العالم و المتعلم شريكان في الأجـر

حديث شريف

الإهداء

إلى من يستحق الحمد على كل حال ويستحق الشكر على ماقدم وأعطى إلى الله رب العالمين.

إلى من علمتنا معنى العطاء إلى من وهبتنا دفئ الحنان إلى من سبب وجودي الصغير في هذا الكون الكبير إلى أمي.

إلى من علمنا معنى التضحية ومعنى العمل إلى من علمني قسوة الحنان وروعة الوفاء إلى أبي.

إلى من اعطانا أدواته الهندسية لنسطر بها دروبنا العشوائية لتنحني العقبات ونصعد الأمجاد إلى اساتذتنا.

ملخص

الهدف من المشروع:

الدراسة التحليلية و التصميمية لبناء نظام تشغيل للحاسب الشخصي و تنفيذ هذا النظام بحيث يكون متعدد المهام، والخيوط، والعمليات، والمستخدمين. ويكون عبارة عن نظام مغلق closedيقلع به الحاسب و يتعامل مع الذاكرة و الأجهزة المحيطية بالإضافة إلى تعامله مع كرت الشبكة وبروتوكول TCP/IP. ويزود هذا النظام دعماً كاملاً للغة العربية حيث يمكن تنفيذ الأوامر باللغة العربية.

مميزات النظام:

يتميز نظام التشغيل بما يلي:

- نسخة عربية كاملة و نسخة أخرى إنكليزية كاملة ويمكن للمستخدم اختيار أحدهما عند الإقلاع ليعمل عليها.
- تعدد الإقلاع: يمكن إقلاعه مع أنظمة أخرى موجودة على الجهاز مثل وندوز و لينكس.
- الإقلاع من قرص مرن أو قرص مضغوط دون الحاجة لتنزيله على الهارد.
- يعمل في النمط المحمي **Protected Mode**.
- متعدد المهام **Multitasking**
- متعدد الخيوط **Multithreading**
- متعدد العمليات **Multiprocessing**
- متعدد المستخدمين **Multiuser**
- يستطيع التعامل مع جميع أنواع الذواكر، حتى حجم أعظمي **4GB**.
- التعامل مع بطاقة الشاشة في النمط النصي **Text Mode** ضمن النمط المحمي للنظام.
- توفير كونسولات متعددة **MultiConsole**.
- دعم لغتين في النمط النصي عربية وانكليزية، مع امكانية تحميل أي لغة أخرى.
- التعرف على القرص الصلب والمرن والتعامل معهما
- دعم كل من نظامي الملفات **EXT2**، **FAT12**.
- استخدام الشبكة في مستوى نواة النظام مما يؤمن سرعة و أمن أكبر
- إمكانية إرسال البيانات لأي جهاز آخر عبر بروتوكول **TCP/IP** وبالتالي يستطيع نظام التشغيل الاتصال مع أي نظام تشغيل آخر يدعم بروتوكول **TCP/IP**.
- أكثر من 30 أمر للتعامل مع النظام متاحة باللغتين العربية والانكليزية، مع شرح وافي عن كل منها.

Abstract

The aim of this project is to provide an analytical study of a novel operating system and then designing and implementing it .This project supports many aspects of modern OSs, such as multitasking, multiprocessing, multithreading, multiusers. In addition to these, this OS can manage the memory and I/O devices, including the network card via TCP/IP protocol. There are two versions of this new OS, The Arabic version and English one.

In spite of the previously mentioned features, this OS supports the following most significant ones :

- Full English and Arabic versions, so it is possible to choose what version to work on during the booting stage.
- Multibooting, so there is ability to boot another OS, like Windows.
- Booting from CD or floppy disk.
- Protected mode OS.
- Managing up to 4GB of memory.
- Multiconsole.
- Supporting two languages in text mode, Arabic and English languages.
- Managing different hard and floppy disk controllers.
- Supporting Ext2 and FAT12 file systems.
- Networking in Kernel level.
- Supporting TCP/IP protocol, so it is possible to connect with any other OS that supports TCP/IP protocol

فهرس

الباب . تهيئة النظام

مُقَدِّمَة

ظهرت أنظمة التشغيل مع بدايات ظهور الحاسب عندما برزت الحاجة لوجود نظام برمجي يقوم بدور الوسيط بين الآلة و المستخدم. و مع تزايد الحاجة لتطوير برمجيات معقدة أصبح من المهم أن يقوم نظام التشغيل بتقديم نموذج بسيط و عالي المستوى للآلة كي يستطيع المبرمجون كتابة برامجهم بسرعة، مما أعطى نظام التشغيل دورا هاما في التمييز بين أنظمة الحاسب المختلفة

و نظرا لأهمية هذا العلم و تأثيره على الكثير على علوم و تقنيات الحاسب الأخرى، قررنا الدخول في هذا المجال الجديد في مشروع يعتبر الأول من نوعه على صعيد مشاريع التخرج، و لعل الهدف الأول هو إزالة الرهبة من كلمة كتابة نظام تشغيل و الشروع ببناء نظام تشغيل عربي يعبد الطريق نحو الدفعات القادمة لتطويره لبناء نظام تشغيل متكامل.

و قد حاولنا أن ننطلق في كتابة نظام تشغيلنا من الصفر أي دون الاعتماد على أنظمة أخرى جاهزة و اعتمدنا في فلسفة بناءه على نفس الفلسفة المتبعة في نظام يونكس.

و نظرا إلى أن المكتبة العربية تعتبر فقيرة في لكتاب يتناول هذه المواضيع فقد رأينا من واجبنا تقديم هذه الدراسة المتواضعة لتعريف الطلاب بهذا العلم الهام، و قد استقينا مادتها بحيث تشمل معظم المواضيع التي يمكن أن تواجه مصممي و مبرمجي أنظمة التشغيل.

و قد اعتمدنا في كتابتنا للدراسة على الشرح الواضح المتسلسل بتصميم و تنفيذ المشروع و بحيث يركز كل فصل على موضوع رئيسي واحد من مواضيع أنظمة التشغيل و يذكر جميع التقنيات و المتعلقة بالموضوع ثم ينتقل للقسم التنفيذي الذي يشرح كيفية تنفيذ هذه التقنيات ضمن نظام تشغيلنا بحيث لا يكون الكلام نظريا جافا و إنما عمليا قابلا للتطبيق.

يتحدث الفصل الثاني عن لمحة عامة عن أنظمة التشغيل و التعريف بها فينتقل من التطور التاريخي لأنظمة التشغيل عبر العقود الماضية إلى أنواع أنظمة التشغيل و من ثم بنية أنظمة التشغيل الشائعة عند تصميم نظام تشغيل و بعد ذلك يتحدث عن أنماط عمل المعالجات في انتل.

يتحدث الفصل الثالث عن إقلاع الحاسب و هو فصل يمكن اعتباره منفصل عن نظام التشغيل و نتكلم فيه عن المبادئ النظرية لعملية الإقلاع و كتابة برامج الإقلاع و محملات الإقلاع المختلفة وصولا لإنشاء قرص إقلاع و يتدرج الفصل في كتابة محمل إقلاع صغير يقوم بداية بطباعة حرف على الشاشة و من ثم تطويره لطباعة جملة و من ثم جعله تفاعليا أكثر بطباعة حرف يدخله المستخدم و يخلص إلى برنامج محمل إقلاع متكامل. و بعد شرح المبادئ النظرية يتم التكلم عن تحقيق ذلك عمليا و شرح مبدأ تعدد الإقلاع الذي يسمح لنظام تشغيلنا أن يتم إقلاعه مع نظام تشغيل آخر مثل وندوز.

يتحدث الفصل الرابع عن أنماط تهيئة المعالج حيث يمتلك المعالج إحدى النمطين المحمي أو الحقيقي و يناقش الفصل ضرورة استخدام النمط المحمي في نظام تشغيلنا لينتقل بعد

ذلك لشرحه بالتفصيل موضحا أشكال العنونة في النمط المحمي و جداول المقاطعات و الوصول إليها.

يتحدث الفصل الخامس عن إدارة الذاكرة منطلقا من المبادئ الأساسية لذلك و يتحدث عن مبادئ عامة مثل التبديل و التقطيع و مفهوم الصفحات و معالجة أخطاء الصفحات.

يتحدث الفصل السادس عن إدارة العمليات و العمليات المختلفة اللازمة مثل إنشاء عملية و إنهاء عملية و حالات العملية و بعد ذلك ينتقل للحديث عن خيوط التنفيذ و أنواعها وصولا لتقنية التضعيف عند محاولة الكتابة.

يتحدث الفصل السابع عن أنظمة الملفات و يبدأ بشرح بنية الملف و من ثم بنية الفهرس و التعامل معهما و بعد ذلك يشرح نظامي الملفات EXT و نظام الملفات FAT النظامان المعتمدان في نظام تشغيلنا و كيفية تنفيذهما في نظام تشغيلنا.

يتحدث الفصل الثامن عن دعم الشبكة حيث أن انتشار الشبكات هذه الأيام جعل من المستحيل غض النظر عن الشبكات و يتحدث هذا الفصل أهمية الشبكة و من ثم الشرح التفصيلي لبطاقة الشبكة و تعريفها و من ثم تطبيق نموذج OSI و طبقاته السبعة لينتقل إلى شرح بروتوكولات الشبكة و تطبيقها في نظامنا التشغيل وصولا لشرح كيفية تعرف جهازين على بعضهما عبر الشبكة.

يتحدث الفصل التاسع عن مقدمة عامة لأجهزة الدخل و الخرج و التعامل معها و يمكن أن نعتبره كمقدمة نظرية تشرح كيفية كتابة مشغلات الأجهزة المختلفة و طبقات أجهزة الدخل و الخرج و هذا القسم لا يتضمن قسما تنفيذيا حيث سيتم شرح كل جهاز دخل/خرج في فصل منفصل.

يتحدث الفصل العاشر عن مقاييس التوقيت من عتاد المؤقتات إلى برمجيات المؤقتات و أنماط عمل المؤقتات بالتفصيل و كتابة برنامج تخديم المؤقت و علاقة المؤقت بالمقاطعة IRQ0 وصولا لساعة الزمن الحقيقي و التعامل مع مسجلاتها.

يتحدث الفصل الحادي عشر عن لوحة المفاتيح و يوضح كيفية التعامل معها كأحد أهم أجهزة الدخل فيشرح بداية البنية العتادية للوحة المفاتيح و من ثم شفرات التحكم و الضغط و الأسكي و شفرة لوحة المفاتيح الموسعة وصولا لبناء خريطة لوحة المفاتيح و كيفية التعامل مع لغات أخرى و التعامل في النمط المحمي.

يتحدث الفصل الثاني عشر عن بطاقة العرض و يشرح بالتفصيل توابع بيوس الاظهارية و خدمات المقاطعة و من ثم ذاكرة العرض الفيديوية و التعامل مع المسجلات في لنمط المنخفض.

يتحدث الفصل الثالث عشر عن سواقة القرص المرن و الصلب فيتكلم عن البنية و أشكال القرص و توابع البيوس للتعامل معه ليصل إلى ميزات الأقراص الصلبة و تجزئتها و جداول التجزئة.

يتحدث الفصل السابع عن إستدعاءات النظام و أهميتها و يتحدث عن تهيئة استدعاء النظام و تمرير البارامترات و كالعادة يشرح المكاتب اللازمة لذلك بالتفصيل.

يتحدث الفصل الخامس عشر عن تنفيذ البرامج في نظام تشغيلنا حيث تم اعتماد الملفات التنفيذية من النوع ELF لتنفيذها و يشرح هذا الفصل بنية هذا النمط بالتفصيل و كيفية تحقيقه.

يتحدث الفصل السادس عشر عن مبدأ التعريب و تطبيقه في نظام التشغيل لدينا و كيفية تمثيل المحارف و تعريب الأوامر.

يتحدث الفصل السابع عشر عن تنفيذ النظام و أمثلة توضيحية له و صور تبين سير عمله.

و في الختام نقول إن كلية المعلوماتية بأمس الحاجة إلى هذا النوع من المشاريع التي تنأى عن كونها تعليمية مقتصرة على التقنيات التي تعلمها و هي لا تزول بزوال التقنيات التي تغطيها و لقد بذلنا قصارى جهدنا في أن تكون الدراسة متكاملة بحيث تعبد الطريق للزملاء الراغبين في تطوير المشروع وصولا لنظام تشغيل عربي متكامل من كل النواحي..

الله ولي التوفيق...

1 -لمحة عامة

1 1 -ما هو نظام التشغيل

يمتلك معظم مستخدمي الحاسب بعض الخبرة في أحد أنظمة التشغيل, و لكن يبقى من الصعب الإجابة على سؤال ما هو نظام التشغيل بدقة؟ ولكن يمكن القول كجواب أولي أن نظام التشغيل هو برنامج يدير عتاديات الحاسب و يوفر أساسات البرامج التطبيقية كما يؤدي دور الوسيط بين مستخدم الحاسب و عتاديات الحاسب و لعل أحد الأوجه المذهلة في نظم التشغيل هو التنوع الواسع في إنجازها تلك المهام.

يقسم النظام الحاسبي عموما إلى أربعة مكونات: العتاديات، نظام التشغيل، البرامج التطبيقية، المستخدمين حيث توفر العتاديات مثل وحدة المعالجة و الذاكرة و تجهيزات الدخل و الخرج الموارد الأساسية للحاسب و تحدد البرامج التطبيقية مثل معالجات النصوص و برامج الجدال الالكترونية و متصفحات الانترنت الأساليب التي تستخدم بها هذه الموارد لحل المسائل الحاسبية للمستخدمين يتولى نظام التشغيل مهمة مراقبة و تنسيق استخدام العتاديات بين مختلف البرامج التطبيقية لمختلف المستخدمين كما في الشكل التالي:

إذا يمكن تشبيه نظام التشغيل بالحكومة الذي يقوم بالوظائف الأساسية و يوفر ببساطة بيئة يمكن لغيره من البرامج القيام بعمل مفيد و لكي نتعرف على نظم التشغيل أكثر لا بد من لمحة تاريخية سريعة لتطورها.

1 2 -لمحة تاريخية عن أنظمة التشغيل

شهدت أنظمة التشغيل تطورا كبيرا منذ ظهورها حتى الآن و سنلقي الضوء في المقاطع التالية على أهم المراحل التي مرت بها أنظمة التشغيل و لما كانت أنظمة التشغيل تاريخيا وثيقة الصلة بالأجهزة التي تعمل عليها فإننا سنطلع على الأجيال المتعاقبة من الحواسب لنرى كيف كانت أنظمة تشغيلها تبدو.

1 2 1 -الجيل الأول 1945-1955 الصمامات المفرغة و لوحات التوصيل

بدأت الحواسب بالظهور في أواسط الأربعينات و كانت بدائية جدا و تستخدم الصمامات و كانت تتألف من آلاف الصمامات التي تملأ عدة غرف و لم يكن آنذاك لغات برمجة (حتى لغة التجميع) و لم يكن هناك أنظمة تشغيل أيضا.

2 2 1 -الجيل الثاني 1955-1965 الترانزستور و الأنظمة الدفعية

تغيرت الصورة جذريا مع ظهور الترانزستورات حيث أصبحت الحواسب أصغر حجما بكثير و أكثر موثوقية و أقل أعطالاً و أصبح هناك تمييز بين المصممين و المنفذين و المشغلين و المبرمجين و عمال الصيانة و بسبب غلاء التجهيزات آنذاك فقد ظهرت فكرة البرمجة الدفعية حيث يتم تجميع مجموعة من الأعمال ثم تحويلها إلى أشرطة مغناطيسية بواسطة حاسب رخيص ثم نقلها إلى الحاسب الرئيسي ليتم معالجتها دفعة واحدة.

1 2 3 -الجيل الثالث 1965-1980 الدارات المتكاملة و البرمجة المتعددة

كانت الشركات في نهاية الستينات تنتج نوعين من أجهزة الحاسب الحواسب/System360 وهوة و الحواسب التجارية الصغيرة نسبيا و كان استخدام و برمجة نوعين مختلفين من الأجهزة مهمة صعبة، خاصة أن معظم الشركات كانت تبدأ بحاسب صغير ثم تتوسع إلى حاسب كبير.قدمت IBM حلا لهذه المشكلة من خلال تطوير جيل جديد من الحواسب يسمى System/360 وهو عبارة عن سلسلة من الحواسب المتوافقة مع بعضها من الناحية البرمجية و تتدرج بالحجم من الكبيرة القوية إلى الصغيرة التجارية و بما أن جميع هذه الأجهزة لها نفس مجموعة التعليمات فإن البرامج المكتوبة لأحدها يمكن أن تعمل عليها كلها نظريا كما أنها مصممة لمعالجة العمليات العلمية بالإضافة إلى العمليات التجارية.و بعد سنوات قدمت IBM سلسلة متوافقة مع 360.

كان 360 الجهاز الأول المعروف الذي استخدم الدارات المتكاملة و كان هذا تطورا كبيرا عن الجيل الثاني الذي استخدمت حواسبه الترانزستورات المنفردة و مازالت هذه الأجهزة مستخدمة حتى الآن لكن بنسخ أحدث و فيما يلي بعض مميزات هذا الجيل:

البرمجة المتعددة حيث في الجيل السابق كان المعالج يبقى خاملا عند القيام بعملية دخل / خرج حتى ينهي هذه العملية و تم حل هذا الأمر في البرمجة المتعددة.

الأرتال أي عملية ترتيل عمليات الطرفيات و وضعها في أرتال و هو ما يدعى ب Spooling.

المشاركة الزمنية أي تقسيم الزمن لخدمة مجموعة لا بأس منها من المستخدمين بنفس الوقت و بشكل تفاعلي.

1 2 4 -الجيل الرابع 1980- الوقت الحاضر الحواسب الشخصية

بدأ عصر الحواسب الشخصية مع ظهور الدارات ذات التكامل الواسع LSI و أصبحت بعد ذلك الحواسب الشخصية رخيصة و بمقدار أي شخص شراءها. صممت IBMجهازها الشخصي IBM PC في أوائل الثمانينات و طرحته في الأسواق مرفقا مع نظام التشغيل MS DOS ومفسر لغة BASIC من شركة MicroUNIX.لناشئة و قد كان أحد أهم القرارات الحكيمة ل بيل غيتس أنه قرر توزيع نظام تشغيله مع الأجهزة من خلال الاتفاق مع شركات تصنيع العداد مثل IBM و غيرها على عكس ما كانت تفعله منافسته شركة Digital Research التي كانت تسوق نظام التشغيل CP/M مباشرة للمستخدم النهائي.أصبح بعد ذلك نظام MS DOS نظام التشغيل الأشهر و قد أضيفت له الكثير من المميزات في الإصدارات اللاحقة و كانت كلها مستوحاة من نظام UNIX.

حتى هذا الوقت كانت جميع أنظمة التشغيل تعتمد على كتابة الأوامر على لوحة المفاتيح و لكن الأمور تغيرت مع ظهور ما يسمى بواجهة المستخدم الرسومية GUI التي صممت في البداية في شركة Xerox على أجهزة Xerox parc المتطورة و لكن Xerox لم تهتم كثيرا بهذه التقنية التي استوحت منها شركة Appleفكرة نظام Lisaالذي لم ينجح بسبب غلاء سعره و لكنها اتبعته مباشرة بنظام Macintosh الذي تميز بسهولة استخدامه فنجح نجاحا باهرا. تأثرت Microsoftبهذا النظام و بدأت بتطوير نظام وندوز الذي لم يكن نظام تشغيل

حقيقي بل كان عبارة عن طبقة رسومية فوق نظام MS DOSو استمر ذلك حتى عام 1995 حيث أطلقت عندئذ نظام وندوز مستقلا أسمته Windows 95 و الذي استخدم نظام DOSفي داخله من أجل الإقلاع و برامج التشغيل القديمة و هكذا بدأت سلسلة الإصدارات من وندوز.

نذكر من المنافسين الآخرين في عالم الحواسب الشخصية نظام التشغيل Unixالذي لاقى نجاحا اكبر على أجهزة محطات العمل و الحواسب القوية مثل مخدمات الشبكات و على الرغم من أن UNIXكان نظام أوامر سطري منذ البداية فإن MIT طورت واجهة رسومية خاصة سمتها X Windows من أجل أنظمة UNIX.

سنكتفي بهذا القدر من السرد التاريخي و دعونا نتحدث قليلا عن أنواع و أشكال أنظمة التشغيل.

1 3 -أنواع أنظمة التشغيل

بعد كل هذا التطور الذي مرت به أنظمة التشغيل، أصبح لدينا مجال واسع من الأنظمة المتنوعة بعضها معروف و البعض الآخر يستخدم في مجالات ضيقة و سنتكلم عن سبعة أنواع.

1 3 1 -أنظمة تشغيل الأجهزة الكبيرة Mainframe

تحتل أنظمة التشغيل الكبيرة المرتبة الأولى من حيث الحجم و تتميز عن الحواسب الشخصية بسعة منافذ الدخل و الخرج فمن الطبيعي أن يحتوي جهاز منها حوالي 1000 قرص صلب و تستخدم ه/360.اسب كملقمات ويب متطورة و ملقمات مواقع التجارة الالكترونية المتطورة.تهتم أنظمة تشغيل هذه الأجهزة بالدرجة الأولى بتشغيل عدة مهام في وقت واحد و تحتوي على كمية هائلة من معالجات الدخل و الخرج من الأمثلة على هذه الأنظمة OS/390 المشتق من OS /360.

2 3 1 -أنظمة تشغيل المخدمات

تعمل هذه الأنظمة على المخدمات التي تكون إما أجهزة شخصية كبيرة جدا أو محطات عمل أو حتى أجهزة كبيرة و تخدم هذه الأنظمة عدة مستخدمين بوقت واحد على شبكة و يسمح للمستخدمين بمشاركة الموارد العتادية و البرمجية و من الأمثلة على هذه الأنظمة UNIX و Windows 2000وظهر مؤخرا على الساحة نظام LINUX.

3 3 1 -أنظمة تشغيل المعالجات المتعددة

تعني المعالجات المتعددة وصل عدة معالجات معا لتامين طاقة معالجة كبيرة و تدعى هذه الأنظمة بالأنظمة المتوازية أو المعالجات المتفرعة و تحتاج لأنظمة تشغيل خاصة بها و عادة ما تكون هي أنظمة مخدمات و مع بعض التعديلات.

4 3 1 -أنظمة تشغيل الحواسب الشخصية

تنحصر مهمة هذه الأنظمة بتقديم واجهة جيدة لمستخدم واحد و من الأمثلة الشهيرة Windowsو Linux و في الواقع يوجد العديد من أنظمة التشغيل الأخرى و لكن ليست مشهورة بسبب الناحية التجارية.

1 3 5 -أنظمة تشغيل الزمن الحقيقي

تعتمد هذه الأنظمة على الزمن بشكل رئيسي مثل أنظمة التحكم بالعمليات الصناعية و تحتوي هذه الأنظمة على نقاط حرجة يجب احترامها و نظم الزمن الحقيقي نوعان Hardو Soft و من الأمثلة على هذه الأنظمة vxworks و QNX.

1 3 6 -أنظمة التشغيل المضمنة

ننتقل الآن الحواسب المحمولة باليد و الأنظمة المضمنة و تتسم غالبا هذه الأنظمة بنفس مزايا أنظمة الزمن الحقيقي لكن لها قيود بالنسبة للحجم و متطلبات الذاكرة و القوة الحسابية و من الأمثلة عليها Palm Osو Windows CE.

1 3 7 -أنظمة تشغيل البطاقات الذكية:

تعمل هذه الأنظمة على البطاقات الذكية ة تحوي على معالج و لها قيود قاسية بالنسبة للقوة الحسابية و حجم الذاكرة و بعض هذه البطاقات يعتمد على لغة الجافا و هذا يعني أن الذاكرة ROM تحتوي على مفسر لآلة جافا الظاهرية JVM.

1 4 -بنية أنظمة التشغيل

1 4 1 -الأنظمة الأحادية

يمكن وصف هذا الأسلوب الذي يعتبر الأكثر شيوعا حتى الآن بأنه فوضى كبيرة "Big Mess" و تعتمد هذه البنية على عدم وجود بنية و يكتب نظام التشغيل على شكل مجموعة من الإجراءات يستطيع كل منها استدعاء أي إجراء آخر عندما يحتاج إليه و لإنشاء البرنامج التنفيذي الفعلي لنظام التشغيل عند إتباع هذا الأسلوب يجب أولا ترجمة جميع الإجراءات أو الملفات التي تحتوي هذه الإجراءات أو الملفات التي تحتوي هذه الإجراءات و من ثم ربطها معا ضمن ملف هدف واحد باستخدام رابط النظام و يكون إخفاء المعلومات مفقودا تماما حيث يستطيع كل إجراء رؤية جميع الإجراءات الأخرى يقترح هذا التنظيم بنية بسيطة لنظام التشغيل:

برنامج رئيسي ينفذ إجراء الخدمة المطلوبة.
مجموعة من إجراءات الخدمة التي تنفذ إستدعاءات النظام.
مجموعة من الإجراءات المساعدة التي تساعد إجراءات الخدمة.

1 4 2 -الأنظمة الطبقية

يتم تنظيم نظام التشغيل على شكل هرمية من الطبقات حيث تقبع كل طبقة فوق الطبقة التي بأسفلها.إن الميزة الرئيسية للأنظمة الطبقية هي الإجتزائية إذ يجري اختيار الطبقات بحيث

تستخدم كل طبقة الوظائف و الخدمات التي تتيحها الطبقات التي هي أدنى منها مستوى فقط وهكذا يمكن إنجاز كل مستوى على حدا حيث يتم إنجاز وظائف العتاديات الأساسية مثلا في المستوى الأول و من ثم يمكن البناء على صحة عمله أثناء إنجاز المستوى الثاني و هكذا.

إن مشاكل هذه الأنظمة تكمن في انه يجب تعريف الطبقات بعناية ويجب أن تكون المتطلبات واضحة و أحيانا تكون هذه الأنظمة أقل فعالية من الأنواع الأخرى فحين يقوم المستخدم بعملية دخل \ خرج فإنه ينفذ استدعاء نظام يستدعي طبقة إدارة الذاكرة التي بدورها تستدعي طبقة وحدة المعالجة و منها إلى طبقة العتاديات و تضيف كل طبقة تحميلا إضافيا على الاستدعاء و بالمحصلة يستغرق وقتا أطول

1 4 3 -الآلات الظاهرية

كما ذكرنا سابقا يتألف النظام عادة من طبقات و العتاديات هي المستوى الأدنى في جميع النظم و النواة هي التي تعمل في المستوى التالي، تستخدم تعليمات العتاديات لإنشاء مجموعة إستدعاءات النظام كي تستخدمها الطبقات الخارجية فبرامج النظام عندئذIBM. قادرة على استخدام إستدعاءات النظام أو التعليمات العتادية و هذه البرامج عادة لا تفرق بين الاثنين.جاءت الأنظمة الظاهرية بحيث تؤمن وجود آلة افتراضية للمستخدم يستطيع خلالها تنفيذ برامجه و هكذا يعطى المستخدمون آلات افتراضية خاصة بهم و من ثم يستطيعون تنفيذ أي نظام تشغيل أو حزمة برمجية متاحة في الآلة الأساسية و كمثال نذكر نظام VM من IBM.

1 5 -أنماط عمل معالجات INTEL

1 5 1 -معالجات 8086

يعمل المعالج في النمط الحقيقي و يسمح بعنونة ذاكرة بحجم 16 خانة, و يعمل في عنونة الذاكرة على أساس تقنية التقطيع. و تحتوي مسجلات المعالج التي هي بحجم 16 خانة أيضا على عناوين المقاطع الذاكرية المباشرة و الإزاحات ضمن هذه المقاطع أي يقوم بالوصول إلى الذاكرة مباشرة. و هذا ما يحققه نظام العمل في النمط الحقيقي إلا أن نمط العمل هذا له بعض السلبيات و لا يؤمن لنا إمكانية الحماية للقطع الذاكرية كما أنه لا يدعم تقنيات تعدد المهام.

1 5 2 -معالجات 80386

تعمل هذه المعالجات أيضا في النمط الحقيقي و بنفس طريقة عمل المعالج 8086 و لكن بشكل أسرع منه، وكما أنها تعمل في نمط جديد لم يكن موجود في معالجات 8086 وأول ما تم إيجاده في المعالج 80286 و تم تحسينه نمط العمل هذا في الإصدار 80386 لكي يعمل على عنونة ذاكرة بحجم عنونة 32 خط عنونة. و يتم العنونة في نمط العنونة هذا على أساس جداول تخطط بنية الذاكرة،و سنقتصر في نظامنا على المعالجات على 80386 فقط لأنها هي المعتمدة في نظام التشغيل المصمم أو لنقل بشكل أدق أن بنية هذه المعالجات هي البنية القياسية المعتمدة.

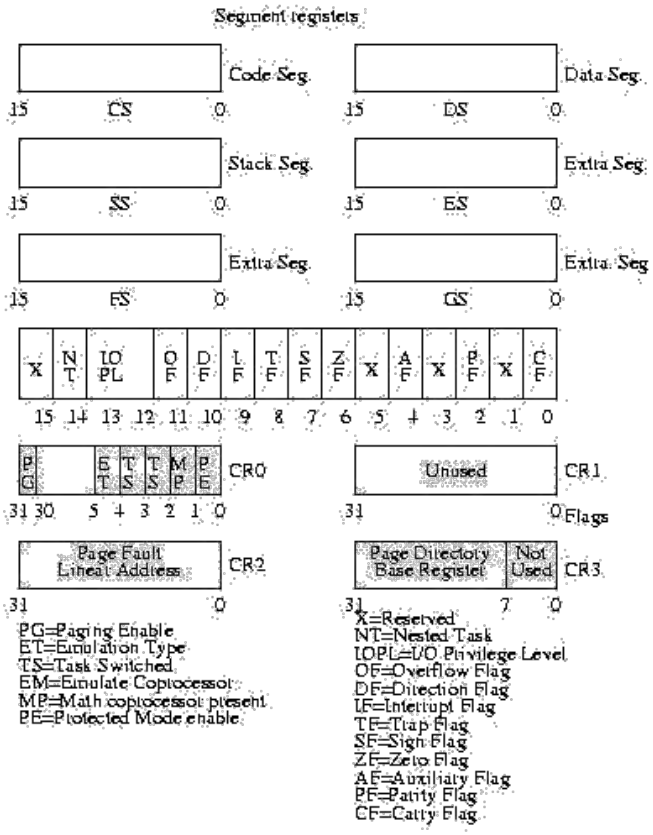
1 5 3 -المسجلات في 80386

كما ذكرنا سنركز في دراستنا هذه على المعالج 80386 و ذلك لأن هذا النوع من المعالجات يعتبر معيارا للمعالجات التي تعمل في النمط المحمي و كل ما أتى بعدها من معالجات من شركة Intel جاء متوافقا و مبنيا على أساس هذه المعالجات و لدينل نوعان من المسجلات في المعالج 80386:

- **مسجلات يمكن للمبرمج الوصول إليها و التعامل معها قراءة أو كتابة:**

- أربع مسجلات أغراض عامة
- مسجل أعلام
- ست مسجلات مقطعية
- مسجلي فهرسة
- مسجل تكديس
- مسجل قاعدي و مسجل التعليمة
- مسجلات الجداول
- **GDTR** مسجل جدول الموصفات العامة.
- **LDTR** مسجل جدول الموصفات المحلية.
- **IDTR** مسجل جدول الموصفات المقاطعات.
- **TR** مسجل تخزين المهمة.
- **CR0 – CR3** مسجلات التحكم,
- **DR0 – DR7** مسجلات نمط التنقيح.

الشكل التالي يبين معظم ما تم ذكره من مسجلات في الأعلى هنالك بعض الأنواع من المسجلات خاصة في النمط المحمي و لا فائدة لها إلا بهذا النمط و سنأتي على ذكرها في حينه.



الشكل 1

مسجلات مخفية لا يمكن الوصول إليها من قبل المبرمج و تستخدم من قبل المعالج و ذلك لأغراض العنونة الذاكرية و التخزين المؤقت لبعض العناوين.

- مسجلات تخزين واصفة مقطع الشفرة الحالي **Code Segment Descriptor Cach Reg**.
- مسجلات تخزين واصفة مقطع المعطيات الحالي: **Data Segment Descriptor Cach Reg**.
- مسجلات تخزين واصفة مقطع التكديس الحالي: **Stack Segment Descriptor Cach Reg**.
- مسجلات تخزين عنوان جدول وصفات المقاطع العامة الـ **GDT** الحالي.
- مسجل تخزين عنوان جدول واصفات المقاطعات الـ **IDT** الحالي.
- مسجل تخزين عنوان جدول الواصفات المحلية **LDT** الحالي.
- سجل تخزين جدول واصفات الحالة **TSS**.

2 -إقلاع الحاسب

مُقَدِّمَة

يعتبر إقلاع الحاسب الخطوة الأولى في بناء نظام تشغيل حيث هو المرحلة التي تأتي بعد تشغيل الجهاز إلى تحميل نظام التشغيل و تسليم القيادة له.عندما يقوم جهاز الحاسب بالإقلاع من قرص مرن تقوم وحدة الدخل و الخرج للنظام BIOS بقراءة القرص و تحميل القطاع الأول إلى الذاكرة عند العنوان 0000:7C00. يدعى القطاع الأول بسجل إقلاع دوس DOS Boot Record (DBR). تقوم البيوس BIOS بالقفز إلى العنوان 0x7C00 و تقوم بتنفيذ التعليمات الموجودة هناك و تكون هذه التعليمات عبارة عن برنامج تحميل الإقلاع boot loader و الذي سيقوم بتحميل نظام التشغيل OS إلى الذاكرة و تبدأ عندئذ عملية إقلاع نظام التشغيل.

تم البحث في هذا المجال بالاعتماد على الكتب و المقالات التخصصية في كتابة برامج الإقلاع و الفروقات بينها و الاحتمالات الموجودة لكتابة برنامج إقلاع و تبين أن بناءنظام تشغيل يتطلب برنامجين رئيسين و هما نواة نظام التشغيل أي نظام التشغيل بحد ذاته و برنامج الإقلاع الذي يقوم بإقلاع الحاسب و تحميل نظام التشغيل إلى الذاكرة و تشغيله.انطلقنا في العمل في جعل الحاسب قادر على الإقلاع من قرص مرن و بالتالي تشغيل نظامي الخاص بي و الذي كان بداية عبارة عن حلقة لا نهائية لا تقوم بشيء مفيد و بعد ذلك شرعنا في بناء نظام تشغيل بسيط يقوم بطباعة حرف واحد على الشاشة و بعد ذلك وضع هذا النظام على قرص مرن والإقلاع منه. ثم قمنا بتطوير نظام التشغيل ذلك بحيث يقوم بطباعة جملة HELLO WORLDعلى الشاشة، ثم فكرنا بعد ذلك بجعله أكثر تفاعلا بحيث يقوم نظام التشغيل بقراءة أحرف من لوحة المفاتيح و طباعتها على الشاشة.الخطوة الأخيرة كانت في كتابة برنامج إقلاع للحاسب مستقل و يقوم ذلك البرنامج بتحميل نظام التشغيل إلى الذاكرة, و بعد ذلك تم البحث في برامج الإقلاع الجاهزة و المستخدمة في نظم التشغيل المختلفة في وقتنا الحاضر ولتنفيذ ذلك تم استخدام لغة البرمجة الأسمبلي لكتابة البرامج و قد استعنت بأداة المنقح لكتابة برامج الأسمبلي في البداية و من ثم نسخ البرنامج للقرص الصلب, و للحصول على طريقة سريعة لكتابة برامج الأسمبلي قمت باستخدام برنامج الأسمبلي "Netwide Assembler" (NASM).

بداية كان العمل يقتصر على ال512 بايت ضمن القطاع الأول من القرص الصلب بحيث أصبح برنامج الإقلاع موجود ضمن القطاع الأول من القرص, و لكن كما نعلم انه من المستحيل أن يقتصر نظام تشغيلنا على 512 بايت فقط فكان الحل بكتابة برنامج إقلاع يقوم ببساطة بتحميل ملف تنفيذي من القرص و يبدأ بتنفيذه و يدعى هذا البرنامج بمحمل الإقلاع Boot Loader, و يكون الملف الذي سنحمله من القرص بالحجم الذي نريده و لن يقتصر حجمه على قطاع واحد. وبعد ذلك قمنا بكتابة برنامج الإقلاع loader, هذا المحمل loader يفترض استخدام نظام الملفات FAT12 و هو النظام الذي تتم به تهيئة القرص المرن. و لذلك من اجل نظاما آخر عليه استخدام محملloader آخر.

في النهاية تم استخدام محمل الإقلاع GRUB الذي يتناسب مع نواة نظام تشغيلنا التي قمنا بتطويرها حيث يعود السبب إلى كون النواة من النوع ELF و بالتالي لا يمكن تحميلها بالمحملات السابقة و أما كتابة محمل خاص جديد خاص بهذه النواة فهو يعتبر ضربا من الجنون، و خاصة إذا علمنا أن محمل الإقلاع GRUBهو عبارة عن برنامج حر و مجاني و مفتوح المصدر و يمكن تطويره بحرية تحت رخصة البرامج الحرة و المفتوحة المصدر.

1. 2 -قطاع الإقلاع The Boot Sector

في هذه القسم سنتعرف على محتويات قطاع الإقلاع و بعد ذلك سنتعلم كيف نكتب برنامج الإقلاع الخاص بنا.عندما يقوم جهاز الحاسب بالإقلاع من قرص مرن تقوم وحدة الدخل و الخرج للنظام BIOS بقراءة القرص و تحميل القطاع الأول إلى الذاكرة عند العنوان 0000:7C00. يدعى القطاع الأول بسجل إقلاع دوس DOS Boot Record (DBR). تقوم البيوس BIOSبالقفز إلى العنوان 0x7C00 و تقوم بتنفيذ التعليمات الموجودة هناك و تكون هذه التعليمات عبارة عن برنامج تحميل الإقلاع boot loader و الذي سيقوم بتحميل نظام التشغيل OS إلى الذاكرة و تبدأ عندئذ عملية إقلاع نظام التشغيل.

بداية سنقوم بإلقاء نظرة إلى داخل سجل الإقلاع, و سيساعدنا في ذلك أداة المنقح DEBUG و الذي يستخدم لعرض محتويات الذاكرة و الأقراص و في حالتنا سنقوم بعرض محتويات سجل الإقلاع لقرص مرن.

1. 1. 2 -المنقح :Debugger

يمكن تشغيل المنقح من محرر الأوامر دوس DOSأو من قائمة ابدأ Startو اختيار تشغيلRUN ثم كتابة DEBUG.

2. 1. 2 -عرض محتويات الذاكرة

يقوم الأمر dبعرض جزء من محتويات الذاكرة RAM.

```
-d
136E:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0110  00 00 00 00 00 00 00 00-00 00 00 00 34 00 5D 13  ............4.].
136E:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
136E:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

2 1 3 -الحصول على المساعدة

من أجل الحصول على المساعدة و الشرح لأمر ما يمكنك كتابة ?.

```
-?
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [value]
unassemble   U [range]
write        W [address] [drive] [firstsector] [number]
allocate expanded memory       XA [#pages]
deallocate expanded memory     XD [handle]
map expanded memory pages      XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

عليك الانتباه و الحذر عند التعامل مع المنقح حيث أنه يمكن أن يقوم بكتابة معطيات جديدة فوق أخرى موجودة في القرص الصلب و إتلافه أو أن يؤدي لضياع معلومات و حذفها من القرص الصلب.

2 1 4 -تحميل سجل الإقلاع

خذ قرص مرن floppy و قم بتهيئته formatثم ضعه في محرك الأقراص المرنة.لتحميل سجل الإقلاع Boot Record لقرصك المرن أدخل الأمر التالي:

-l 0 0 0 1

يقوم هذا الأمر l(الحرف L) بتحميل القطاعات من القرص إلى جزء من الذاكرة RAM أما الأرقام الأربعة التالية فتعبر عن بداية العنوان الذي ستقوم بتحميله إلى الذاكرة حيث:

الرقم الأول: رقم القرص الصلب حيث 0 تشير للقرص المرن الفلوبي floppy.
الرقم الثاني: رقم القطاع من القرص حيث في حالتنا نريد القطاع الأول أي القطاع ذو الرقم 0.
الرقم الثالث: العنوان الذي سنبدأ منه التحميل.
الرقم الرابع:عدد القطاعات التي نريد تحميلها.

و يصبح الأمر السابق يقوم بتحميل القطاع الأول من القرص المرن إلى الذاكرة بدءا من العنوان 0 و بعدد قطاع واحد.

الآن قمنا بتحميل القطاع إلى الذاكرة و نريد عرض محتوياته, يتم ذلك بالأمر التالي:

```
-d 0
```

سيظهر لديك الشكل التالي حيث يتم عرض 128 بايت (0x80 in hex) من سجل إقلاع القرص المرن كالتالي:

```
0AF6:0000   EB3C 90 4D 53 44 4F 53-35 2E 30 00 02 01 01 00
<.MSDOS5.0.....
0AF6:0010  02 E0 00 40 0B F0 09 00-12 00 02 00 00 00 00 00...@.....
0AF6:0020  00 00 00 00 00 00 29 F6-63 30 8.8 4E 4F 20 4E 41......).c0.NO
NA
0AF6:0030  4D 45 20 20 20 20 46 41-54 31 32 20 20 20 33 C9  ME  FAT12
3.
0AF6:0040  8E D1 BC F0 7B 8E D9 B8-00 20 8E C0 FC BD 00 7C....{.........|
0AF6:0050   38 4E 24 7D 24 8B C1 99-E8 3C 01 72 1C 83 EB 3A
8N$}$....<.r...:
0AF6:0060   66 A1 1C 7C 26 66 3B 07-26 8A 57 FC 75 06 80 CA
f..|&f;.&.W.u...
0AF6:0070  02 88 56 02 80 C3 10 73-EB 33 C9 8A 46 10 98 F7..V....s.3..F...
```

كنظرة أولية لن يخبرنا هذا الشكل أي معلومات و لكن يمكن أن أعرف بعض المعلومات مثل أن نظام الملفات هو FAT12 و أن نوع القرص هو MS-DOS 5.0 بدون اسم.

إن الأرقام ضمن العمود اليساري تظهر عناوين الذاكرة RAM أما الأرقام الست عشرية في المنتصف فهي البايتات الموجودة ضمن محتوى الذاكرة حيث أن محتوى الذاكرة يعرض كقيم ست عشرية بالطبع.العمود الموجود في أقصى اليمين يظهر أحرف الأسكي المقابلة للقيم الست عشر و في حال عدم وجود حرف مرئي مقابل للرقم الست عشري فانه يتم عرض الرمز النقطة (.) لاحظ وجود عدة نقاط.....

الخ.ض هذه البايتات التي تراها في هذا الجزء من سجل الإقلاع هي أجزاء من التعليمات في محمل الإقلاع (boot loader) و بعض هذه البايتات تقوم بتخزين المعلومات حول القرص مثل رقم البايتات ضمن القطاع, رقم القطاعات في المسار track... الخ.

2 1 5 -شفرة محمل الإقلاع

دعونا نلقي نظرة إلى شفرة محمل الإقلاع boot loader.

لنكتب الأمر التالي:

```
-u 0
```

تقوم هذه التعليمة بفك تجميع unassembled أي تظهر لك التعليمات بلغة التجميع المقابلة للبايتات السابقة و بدءا من العنوان المحدد و في حالتنا هو الصفر و سنحصل على الخرج التالي:

```
0AF6:0000 EB3C        JMP     003E
0AF6:0002 90          NOP
0AF6:0003 4D          DEC     BP
0AF6:0004 53          PUSH    BX
0AF6:0005 44          INC     SP
0AF6:0006 4F          DEC     DI
0AF6:0007 53          PUSH    BX
0AF6:0008 352E30      XOR     AX,302E
0AF6:000B 0002        ADD     [BP+SI],AL
0AF6:000D 0101        ADD     [BX+DI],AX
0AF6:000F 0002        ADD     [BP+SI],AL
0AF6:0011 E000        LOOPNZ  0013
0AF6:0013 40          INC     AX
0AF6:0014 0BF0        OR      SI,AX
0AF6:0016 0900        OR      [BX+SI],AX
0AF6:0018 1200        ADC     AL,[BX+SI]
0AF6:001A 0200        ADD     AL,[BX+SI]
0AF6:001C 0000        ADD     [BX+SI],AL
0AF6:001E 0000        ADD     [BX+SI],AL
```

تقوم التعليمة الأولى JMP 003 بالقفز إلى العنوان 0x3E. أما التعليمات التالية فهي معلومات القرص التي ذكرناها قبل قليل و لكن هذه التعليمات لا توافقها تماما و إنما بالعنوان x3E0عل ما بوسعه بتحويل البايتات إلى تعليمات أسمبلي و يفسر البايتات بالشكل السابق.إذا إن التعليمة الأولى تقوم بالقفز إلى العنوان الذي يبدأ عنده برنامج الإقلاع boot program والذي يبدأ عادة بالعنوان x3E0 دعونا نطلع على هذه التعليمات بكتابة الأمر:

```
-u 3E
```

هنا يمكنك رؤية بداية سجل MS-DOS يقوم بتحميل نظام التشغيل DOS أو Windows.هذا الكود لنظام التشغيل MS-DOS يقوم بالبحث على القرص الصلب عن الملفات IO.SYS MSDOS.SYS , حيث تحتوي هذه الملفات على شفرة نظام التشغيل, تقوم شفرة محمل الإقلاع boot loader بتحميل هذه الملفات إلى الذاكرة ة البدء بتنفيذها و إما إذا لم يجد محمل الإقلاع هذه الملفات فإنه يقوم بعرض رسالة الخطأ الشائعة.

```
Invalid system disk
Disk I/O error
Replace the disk, and then press any key
```

هذه الرسالة تكتب في سجل إقلاع الدوس بدءا من العنوان 180 كما يبن الشكل:

```
-d 180
0AFC:0180  18 01 27 0D 0A 49 6E 76-61 6C 69 64 20 73 79 73..'..Invalid sys
0AFC:0190  74 65 6D 20 64 69 73 6B-FF 0D 0A 44 69 73 6B 20  tem disk...Disk
0AFC:01A0  49 2F 4F 20 65 72 72 6F-72 FF 0D 0A 52 65 70 6C  I/O error...Repl
0AFC:01B0  61 63 65 20 74 68 65 20-64 69 73 6B 2C 20 61 6E  ace the disk, an
0AFC:01C0  64 20 74 68 65 6E 20 70-72 65 73 73 20 61 6E 79  d then press any
0AFC:01D0  20 6B 65 79 0D 0A 00 00-49 4F 20 20 20 20 20 20   key....IO
0AFC:01E0  53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 7F 01  SYSMSDOS SYS..
0AFC:01F0  00 41 BB 00 07 60 66 6A-00 E9 3B FF 00 00 55 AA.A...`fj..;...U.
```

يظهر الشكل السابق نهاية سجل الإقلاع , هناك ملاحظة هامة جدا و هي أن سجل تليه,ع محدد بالحجم 512 بايت و بالتالي إذا ما تم تحميله إلى الذاكرة بدءا من العنوان 0 سيتوضع البايت الأخير عن العنوان 0x1FF كما أن آخر بايتين ضمن سجل الإقلاع هما 0x1FF , 0x1FE ويحتويان دوما القيمتين0x55 و 0xAA على الترتيب يجب أن دوما وضع هاتين القيمتين ضمن أخر بايتين و إلا سوف لن تقوم البيوس بتحميل القطاع و البدء بتنفيذه.

في الخلاصة نقول إن سجل الإقلاع لدوس يبدأ بتعليمة قفز فوق المعطيات التي تليه , هذه ال60 بايت من المعطيات تبدأ عند العنوان 0x02 وتنتهي عند العنوان 0x3D و تستأنف شفرة الإقلاع عند العنوان 0x3E و تكمل عملها حتى العنوان 0x1FD حيث تنتهي ب بايتين لهما قيمتين محددتين هما 0x55 , 0xAA.

2 2 -إنشاء قرص إقلاع

في هذا القسم سنتعلم كيف نقوم بصنع برنامج إقلاع لقرص مرن و سننطلق في ذلك بتعديل سجل الإقلاع لدوس DOS Boot Record.إن هدفنا هو استبدال شفرة محمل الإقلاع بدون تغيير المعطيات الأخرى في قطاع الإقلاع, لأنه إذا قمنا بتغيير المعطيات إلى شيء آخر غير متاح فسوف لن يعتبر نظام التشغيل دوس DOS أو وندوز Windows القرص الصلب قرصا مقبولا, و سيعطي النظام رسالة خطأ تفيد أن يجب تهيئة القرص المرن.

2 2 1- تعديل سجل الإقلاع

بداية سنحافظ على التعليمة الأولى الخاصة بالقفز jmp 0x3E لأننا نحتاج للقفز فوق معطيات سجل الإقلاع، و للقيام بهذا و البدء بالتعديل بدءا من العنوان 0x3E, دعونا نقوم بتشغيل المنقح و تحميل القطاع الأول من القرص المرن إلى الذاكرة بدا من العنوان 0.

اكتب التعليمة التالي:

```
-u 3E
```

لنبدأ بتعديل المعطيات و ذلك بالأمر:

```
-a 3E
```

```
jmp 3E
```

يقوم المحث prompt بالتغيير إلى العنوان الذي نريد كالتالي:

يتم تنفيذ التعليمة إلى لغة الآلة و توضع في الذاكرة و يشير المحث إلى الموقع الذاكري التالي المتاح بعد إدخال التعليمة, اضغط Enter مرة أخرى لإنهاء شفرة الأسمبلي و يصبح الشكل النهائي للكود كالتالي:

```
-a 3E
0AFC:003E jmp 3E
0AFC:0040
-
```

```
-u 3E
```

الآن يمكنك عرض التعليمات التي أدخلتها كالتالي:

و يكون بالشكل التالي:

```
-u 3e
136E:003E EBFE        JMP     003E
136E:0040 8ED1        MOV     SS,CX
136E:0042 BCF07B      MOV     SP,7BF0
136E:0045 8ED9        MOV     DS,CX
136E:0047 B80020      MOV     AX,2000
136E:004A 8EC0        MOV     ES,AX
136E:004C FC          CLD
136E:004D BD007C      MOV     BP,7C00
136E:0050 384E24      CMP     [BP+24],CL
136E:0053 7D24        JGE     0079
136E:0055 8BC1        MOV     AX,CX
136E:0057 99          CWD
136E:0058 E83C01      CALL    0197
136E:005B 721C        JB      0079
136E:005D 83EB3A      SUB     BX,+3A
```

يمكنك ملاحظة أن التعليمة الأولى تغيرت إلى تعليمة القفز التي وضعناها و بالتالي سنحصل على حلقة لانهائية أي أن برنامج إقلاعنا سيعلق ببساطة لأنه سيتوقف مكانه مستمرا بالقفز عند نفس العنوان, بقي أن نحفظ هذه المعطيات الجديدة على القرص الصلب لان إغلاق المنقح لن يؤدي لآي عملية حفظ و يتم الحفظ كما ذكرنا بالكتابة إلى القطاع الأول من القرص المرن و ببساطة بالأمر التالي:

```
-w 0 0 0 1
```

إن أمر الكتابة write يشبه أمر التحميل load حيث يكتب المعطيات الموجودة في الذاكرة في العنوان 0 للقرص رقم 0 بدءا من القطاع رقم 0 و بطول قطاع واحد 1.عليك الحذر عند استخدام هذا الأمر و خصوصا مع الأقراص الصلبة غير المرنة لأنه يؤدي لكتابة معطيات فوق أخرى موجودة و قد يسبب ضياع معلومات لك.

الآن يمكنك أن تقلع من القرص الصلب, عندما تقوم بإعادة إقلاع الحاسب تقوم البيوس بتحميل المطلوب. الأول من القرص إلى الذاكرة و تبدأ التنفيذ عند بداية القطاع و ستجد عندئذ تعليمة القفز إلى العنوان 0x3E و تنفيذ التعليمة هناك.لآن قم بتنفيذ ما سبق و ستجد أن بعد أعادة الإقلاع سيعلق الحاسب أي لن يفعل شيء و هو المطلوب.

2 2 2 -طباعة حرف على الشاشة

طبعا قد لا تكون راضيا عن النتائج فماذا نستفيد من تعليق الحاسب, لا تتسرع فهذه مجرد بداية لا بد منها تريك أنك يمكنك الاستغناء عن نظام تشغيلك و إعادة إقلاع حاسبك كما تريد, إذا أردنا فعل شيء مفيد فعلينا الاستعانة بتوابع ببوس BIOS و علينا الانتباه أن هناك نوعين من الاستدعاءات منها خاص ببيوس و الآخر خاص ب دوس و طبعا لا يمكن الاستعانة بتلك الخاصة ب(دوسDOS) لأنه بالأصل لا يوجد نظام تشغيل دوس يعمل بالأصل، وتكون قيم المسجلات بالدلالة التالية:

```
AH = 0x0E
AL = ASCII code of the character to be printed
BL = color/style of character
```

أعد الآن التعليمات في هذا الفصل و بدل من تعليمة القفز التي كتبناها سابقا ادخل التعليمات التالية:

```
-a 3E
0AF6:003E mov ah, 0e
0AF6:0040 mov al, 48
0AF6:0042 mov bl, 07
0AF6:0044 int 10
0AF6:0046 jmp 46
0AF6:0048
-
```

تقوم التعليمات السابقة بإسناد القيمة صفر للمسجل AHو القيمة 0x48للمسجل AH حيث تعبر القيمة عن الحرف H, القيمة 7 للمسجل BLو هي تعبر عن لون الخط الأبيض على خلفية سوداء, قم بحفظ قطاع الإقلاع كما سبق (-w 0 0 0 1), أعد الإقلاع مرة أخرى من القرص المرن, هذه المرة ستجد الحرف H مطبوعا على الشاشة قبل تعليق الحاسب.

2 3 -استخدام المجمع NASM

سنتعلم في هذا القسم كيفية استخدام الأسمبلي لكتابة برامجنا, كنا في الفصول السابقة نستخدم أداة المنقح لكتابة برامج الأسمبلي, و لكن كتابة برامج كبيرة يعتبر أمر متعب باستخدام المنقح DEBUG و للحصول على طريقة سريعة لكتابة برامج الأسمبلي سننطلق باستخدام

برنامج الأسمبلي .(NASM) "Netwide Assembler" و يمكنك تحميل البرنامج من الموقع http://nasm.sourceforge.net/index.php كما يمكنك الحصول عليه من القرص المرفق.

2 3 1- كتابة البرنامج

سنستخدم هذا البرنامج لكتابة نفس التعليمات التي قمنا بكتابتها في الفصل السابق.هذا البرنامج ذي الاسم h.asm موجود و يمكنك الإطلاع عليه القرص المرفق, بداية هذه الترويسة للبرنامج.

```
; ----------------------------------------------------
;Simple boot program that prints the letter 'H'
; Disclaimer: I am not responsible for any results of the use of the contents
;of this file
;----------------------------------------------------
```

تبدأ التعليمات في هذه البرنامج بشكل مشابه لما عرضناه في الفصل السابق, أولا تأتي التعليمة التي تقوم بالقفز فوق معطيات سجل الإقلاع و ذلك بالقفز للعنوان begin و بعد القفز فوق هذه المعطيات و هي حوالي 20 بايت من المعطيات لها البنية التالية (حيث قمت باستخلاصها من قرصي المرن بواسطة المنقح) و يمكنك التأكد من ذلك و عليك الانتباه أن الأرقام المؤلفة من أكثر من بايت واحد ستكون معروضة بشكل معكوس byte swapped بسبب بنية Intelو بحيث يخزن البايت الأقل أهمية في المكان ذو العنوان الذاكري الأقل والعكس بالعكس.

```
----------------------------------------------------------------;
 ;data portion of the "DOS BOOT RECORD"
----------------------------------------------------------------;
brINT13Flag    DB     90H           ; 0002h - 0EH for INT13 AH=42 READ
brOEM          DB     'MSDOS5.0'    ; 0003h - OEM name & DOS version(8
chars)
brBPS          DW     512           ; 000Bh - Bytes/sector
brSPC          DB     1             ; 000Dh - Sectors/cluster
brResCount     DW     1             ; 000Eh - Reserved (boot) sectors
brFATs         DB     2             ; 0010h - FAT copies
brRootEntries  DW     0E0H          ;0011 ;h - Root directory entries
brSectorCount  DW     288           ;0013 ;h - Sectors in volume, < 32MB
brMedia        DB     240           ;0015 ;h - Media descriptor
brSPF          DW     9             ; 0016h - Sectors per FAT
brSPH          DW     18            ; 0018h - Sectors per track
brHPC          DW     2             001 ;Ah - Number of Heads
brHidden       DD     0             ; 001Ch - Hidden sectors
brSectors      DD     0     0020 ;     h - Total number of sectors
               DB     0             ; 0024h - Physical drive no.
               DB     0             ; 0025h - Reserved (FAT32(
               DB     29H           ; 0026h - Extended boot record sig
brSerialNum      DD       404418EAH        ; 0027h - Volume serial number
(random(
brLabel        DB     'Joels disk '   ; 002Bh - Volume label  (11 chars(
brFSID         DB     'FAT12   '      ; 0036h - File System ID (8 chars(
----------------------------------------------------------------;
```

لننتقل الآن إلى الشفرة المكتوبة بعد اللائحة Begin حيث ستجد أنها تشبه تماما ما عرضناه في الفصل السابق, فهو يقوم ببساطة بطباعة الحرف H على الشاشة ثم يدخل في حلقة لا نهائية, و في نهاية الشفرة نقوم بالتأكد أن حجم هذا الكود هو 512 بايت و الذي يقابل حجم قطاع واحد و من ثم يأتي السطر ذو الكلمة "times" و الذي يضيف أصفار إلى نهاية الملف بحيث يكون حجم الكود 510 بايت و بعد ذلك نضيف البايتين المعروفين 0x55, 0xAAو اللذان يحددان أن القطاع هو قطاع إقلاع إلى نهاية القطاع.

```
;-----------------------------------------
;Boot program code begins here
;-----------------------------------------
;boot code begins at 0x003E
begin:
        mov   ah, 0x0e      ;Function to print a character to the screen
        mov   al, 'H'       ; Which character to print
        mov   bl, 7         ;color/style to use for the character
        int   0x10          ;print the character

hang:
        jmp   hang          ;just loop forever.

;-----------------------------------------

size    equ   -$entry
%if size+2 > 512
% error "code is too large for boot sector"
%endif
        times  -512)size - 2) db 0
        db    0x55, 0xAA            2;byte boot signature
```

الآن لديك الملف كامل في القرص المرفق, كذلك لديك برنامج NASM قم بفك ضغطه مثلا إلى قرص السي Cوضع داخل مجلده الملف المراد ترجمته ثم قم بترجمة الملف بواسطة المجمع كالتالي:

```
nasm h.asm –o h.bin
```

سيولد لك ملف ثنائي h.bin تأكد أن حجمه هو 512 بايت من خلال خصائص الملف,

قم الآن بنسخ الملف إلى قرصك المرن و بعد ذلك شغل المنقح و اكتب التالي:

```
debug
-n c:\nasm\h.bin
-l 0
```

ستقوم الشفرة السابقة بتحميل الملف إلى الذاكرة بدءا من العنوان 0 ويمكنك التأكد من محتوياته و انه تحمل بشكل صحيح من خلال التعليمتين (d) dumpو (u) unassemble.

حيث عند تنفيذ التعليمة dستلاحظ أن الملف قد أضيف له أصفار حتى عنوان البايتين x1FE and 0x1FF حيث وضع فيهما قيمتين ثابتتين و طبعا هي توجد عند نهاية القطاع.

```
-d
136E:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:0190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
136E:01F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA ..............U.
-
```

يمكنك أن تلاحظ أيضا إن المسجل CXسيحتوي بعد ذلك عدد البايتات المحملة من الملف و طبعا يمكنك عرض محتويات المسجل بالأمر R:

```
-r
AX=0000  BX=0000  CX=0200  DX=0000  SP=FFEE  BP=0000  SI=0000
DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO
NC
136E:0100 0000       ADD     [BX+SI],AL                  DS:0000=EB
-
```

لاحظ أن قيمة المسجل هي 200 بالست عشري و تقابل 512 بالعشري أي إن عملنا بدون أخطاء حتى الآن.أصبح كل شيء جاهز و بقي أن تحمل هذه التعليمات للقرص المرن في مكانها الصحيح:

```
-w 0 0 0 1
```

اعد الإقلاع و سترى نفس النتيجة و هي طباعة الحرف h, في الفصل القادم سنقوم بطباعة "Hello, World".

2 3 2 -طباعة العبارة "Hello, World!"

الآن دعونا ننتقل للغوص في برامج الإقلاع و كتابة نظام تشغيل خاص بنا و ككل المبتدئين في لغات البرمجة لا بد أن بدأت حياتك البرمجية ببرنامج "Hello, World" دعونا نقوم بالشيء بالنسبة لنظام التشغيل و ليكن نظامنا ذو الاسم"Hello, World" operating system. سنقوم بإنشاء تابع يقوم بطباعة نص stringو من خلاله سوف نعرض جملتنا المنشودة. سيكون من الممل أن نقوم بطبع الجملة من خلال طبع الأحرف بشكل متتالي لذلك سنقوم بإنشاء تابع يقوم بطباعة سلسلة نصية تنتهي بالصفر على الشاشة و يكون ذلك من خلال حلقة بسيطة كل الأحرف ضمن السلسلة النصية في نفس الوقت.

```
; ------------------------------------------
; Print a null-terminated string on the screen
; ------------------------------------------
putstr:
        lodsb           ; AL = [DS:SI]
        or al, al       ; Set zero flag if al=0
        jz putstrd      ; jump to putstrd if zero flag is set
        mov ah, 0x0e        ; video function 0Eh (print char)
        mov bx, 0x0007      ; color
        int 0x10
        jmp putstr
putstrd:
        retn
```

دعونا نلقي نظرة سريعة على عمل التابع السابق.نقوم بداية بتحميل العنوان للحرف الأول للسلسة النصية إلى المسجل SIو من ثم نستدعي ببساطة التابع الفرعي putstr. ويمكنك

```
msg    db 'Hello, World!', 0
```

تشكيل السلسلة النصية كالتالي:

و يتم الإشارة لنهاية السلسلة ب الصفر و يبقى أمر طباعتها ببساطة كالتالي:

```
mov si, msg  ; Load address of message
call putstr     ; Print the message
```

بقي أمر واحد عليك إعداده قبل أن يعمل برنامجنا. إن العنوان ل MSG المحمل إلى المسجل SI هو عبارة عن إزاحة عن البداية من بداية القطعة segmentالتي يؤشر لها بالمسجل DS, لذلك قبل أن تستطيع استخدام العنوان MSG يجب أن تقوم بإعداد قطعة المعطيات الحالية current data segment, دعونا الآن نقوم باستخدام العنونة من أسفل الذاكرة RAM ولتحديد قطعة المعطيات بحيث تبدأ من الأسفل نقوم بإسناد القيمة 0 إلى المسجل DS و تقوم التعليمات التالية بذلك:

```
xor    ax, ax ; Zero out ax
mov    ds, ax ; Set data segment to base of RAM
```

لم يبق سوى إعادة الخطوات السابقة للحصول على برنامج إقلاع يطبع الجملة الشهيرة HELLO WORLD. إذا قم بنسخ الملف helowrld.asm إلى مسار NASMعلى سطح القرص Cفي حالتنا , قم بترجمته كما سبق.

2 4 -أكثر من طباعة جملة

طبعا إن طباعة جملة على الشاشة أمر جيد و لكن أي نظام تشغيل يحتاج لشيء من التفاعل مع المستخدم و ليس مجرد عرض المعلومات, دعونا نضفي شيئا من التفاعلية على نظام تشغيلنا الوليد و ذلك بجعله يقرأ دخل من لوحة المفاتيح. سنقوم بقراءة الدخل من لوحة المفاتيح اعتمادا على مقاطعات بيوس BIOSو ليس دوس DOS و من أجل ذلك سنستخدم التابع function 0مع المقاطعة 16 (interrupt 0x16) كالتالي:

```
xor     ah, ah ; we want function zero
int     0x16           ; wait for a keypress
```

يقوم هذا التابع بجعل الحاسب ينتظر حتى أن يتم ضغط مفتاح من لوحة المفاتيح و يخزن شفرة المسح للمفتاح المضغوط في المسجل AH و شفرة الأسكي ASCII code في المسجل AL. سنقوم بهذا الوقت بإضافة شيء من التفاعلية و ذلك في الملف المرفق lesson5.asm.

```
-----------------------------------------;
 ;Boot program code begins here
-----------------------------------------;
 ;boot code begins at 0x003E
begin:
        xor     ax, ax          ;zero out ax
        mov     ds, ax          ;set data segment to base of RAM
        mov     si, msg         ;load address of our message
        call    putstr          ;print the message

loop1:
        xor     ah, ah          ;function 0
        int     0x16            ;get a key from the keyboard

        mov     si, charmsg     ;load address of message
        call    putstr          ;print the message

        mov     ah, 0x0e                ;function print character
        mov     bl, 0x07                ;white on black
        int     0x10

        mov     si, newline     ;print a newline
        call    putstr

        jmp     loop1           ;just loop forever.

-----------------------------------------;
```

أما المعطيات كالتالي:

```
 ;data for our program
msg       db  'Press a key'.
Newline    db 13,10,0
Charmsg   db'Character: ',0
```

و التابع من أجل الطباعة كما سبق مع تعديل بسيط:

```
 ----------------------------------------
;Print a null-terminated string on the screen
----------------------------------------;
putstr:
        push   ax
putstrl:
        lodsb                           ;AL = [DS:SI[
        or      al, al          ;Set zero flag if al=0
        jz      putstrd                 ;jump to putstrd if zero flag is set
        mov     ah, 0x0e                ;video function 0Eh (print char(
        mov     bx, 0x0007              ;color
        int     0x10
        jmp     putstrl
putstrd:
        pop     ax
        retn
----------------------------------------;
```

5 2 -محمل الإقلاع Boot Loader

كل ما كتبناه في الفصول السابقة تم وضعه ضمن القطاع الأول بحيث أصبح برنامج الإقلاع موجود ضمن القطاع الأول من القرص, و لكن كما نعلم انه من المستحيل أن يقتصر نظام تشغيلنا على 512 بايت فقط فما هو الحل ؟ الحل هو بكتابة برنامج إقلاع يقوم ببساطة بتحميل ملف تنفيذي من القرص و يبدأ بتنفيذه و يدعى هذا البرنامج بمحمل الإقلاع Boot Loader, و يكون الملف الذي سنحمله من القرص بالحجم الذي نريده و لن يقتصر حجمه على قطاع واحد. أنها لفكرة جيدة أن نعرج قليلا على نظام الملفات FAT حيث سنفترض أننا نستخدم هذا النظام من الملفات في بحثنا هذا, دعونا الآن نلقي نظرة سريعة إلى عملية تحميل الإقلاع boot loading process.

1 5 2 -عملية تحميل الإقلاع

يحتوي القرص المرن في حالتنا هذه على سجل الإقلاع DOS (القطاع الأول من القرص الذي نعمل عليه), جدول توضع الملفات File Allocation Table (FAT), مجلد الجذر the Root Directory, و من ثم المعطيات المحتواة في الملفات على القرص المرن. بالنسبة للقرص الصلب hard disk فالأمر أكثر تعقيدا, حيث يحتوي على سجل الإقلاع الأساسي Master Boot Record و مبدأ تعدد الأجزاء multiple partitions .

لنفترض أننا قمنا بكتابة نظام تشغيل و قمنا بترجمته إلى ملف سميناه LOADER.BIN و وضعناه على القرص الصلب , إن مهمة محمل الإقلاع هو تحميله كالتالي:

تتم قراءة سجل إقلاع دوس DOS Boot Record (DBR) لتحديد حجم الأشياء التالية: DBR, FAT, Root Directory و بذلك يتحدد موقع كل منهم على القرص.
تتم قراءة مجلدة الجذر Root Directory إلى الذاكرة.
يبحث ضمن المجلد الجذر Root Directory عن الملف ذو الاسم فإذا وجد يمكننا عندئذ البحث ضمن مدخل المجلد directory entry للبحث عن أول عنقود cluster(وحدة توضع الملف file allocation unit) للملف, أما إذا لم يوجد سنعطي رسالة خطأ.

تتم قراءة جدول توضع الملفات من القرص إلى الذاكرة..
نبدأ من العنقود cluster الأول للملف و نستخدم نظام الملفات FAT لوضع جميع العناقيد التي تخص الملف,و تتم قراءتهم من القرص إلى الذاكرة عند موقع محدد.
نقفز إلى ذلك العنوان و نبدأ تنفيذ نظام التشغيل.

يجب أن تتم جميع عمليات القراءة من القرص بواسطة استدعاءات بيوس BIOS, يمكنك الإطلاع على توابع البيوس المستخدمة من أجل قراءة القطاعات من القرص, على كل حال ستجد محمل إقلاع يتعامل مع نظام الملفات FATو هو بالاسم BOOT12.ASM ويمكنك ترجمته وتنفيذه كما تعلمنا سابقا.

2 5 2 -محمل الإقلاع

هناك عدة إعدادات قابلة للتعديل ضمن هذا المحمل loader, هذا المحملloader يفترض استخدام نظام الملفات FAT12 و هو النظام الذي تتم به تهيئة القرص المرن. و لذلك من اجل نظام آخر عليك استخدام محملloader آخر, تتلخص الأشياء التي يمكنك تعديلها بمواقع نظام التشغيل و بنى المعطيات المختلفة لنظام FAT التي سيتم تحميلها إلى الذاكرة, طبعا يمكنك تعديل اسم نظام التشغيل الذي سيقوم المحملloader بتحميله.

افتراضيا يقوم المحمل بتحميل الملف المسمى LOADER.BIN الموجود على مجلد الجذر إلى الذاكرة بدءا من العنوان 0x1000:0000 و يمكنك تعديل العنوان من خلال: %define IMAGE_SEG و بذلك يمكنك ترجمة نظام تشغيلك و نسخه إلى القرص المرن بالاسم LOADER.BIN. دعونا نعتبر أن نظام تشغيلنا هو ذلك النظام الذي يقوم بطباعة جملة Hello World على الشاشةpointer, في الفصل الرابع و سنشغل نظامنا بمحمل الإقلاع هذا, لا يمكننا نفس الملف الذي برمجناه حرفيا في الفصل الرابع و إنما علينا القيام ببعض التعديلات الصغيرة, أحد التعديلات التي تلفت النظر أنه يجب الانتباه أن الملف سيتم تحميله إلى موقع مختلف ضمن الذاكرة و هو 0x1000:0000 عوضا عن 0000:7C00, تعديل آخر أنه يمكننا التخلص و حذف معطيات سجل الإقلاع دوس DOS Boot Record data.سنبدأ شفرتنا بتحديد قطع المعطيات و المكدس data and stack segments و مؤشر المكدس stack pointer , و سنقوم بذلك على النحو التالي:

تخزن قطعة الشفرة الحالية ضمن المسجل CSو تجمع المعطيات الستاتيكية إلى الملف التنفيذي هنا, لذلك سنستخدمه كقطعة معطيات بالإضافة لدوره, و مبدئيا سنستخدم هذا كذلك في قطعة المكدس و من المحتمل أن نغيره مستقبلا.

```
mov   ax, cs ; Get the current segment
```

```
mov   ds, ax ; The data is in this segment
cli                ; disable interrupts while changing stack
mov   ss, ax ; We'll use this segment for the stack too
mov   sp, 0xfffe     ; Start the stack at the top of the segment
sti                ; Reenable interrupts
```

أخيرا يمكننا التخلص من بعض الأسطر في نهاية البرنامج التي تقوم توقيع قطاع الإقلاع و تتأكد أن حجم الملف هو 512 بايت فلسنا بحاجة لذلك الآن و ستجد نظام تشغيلنا

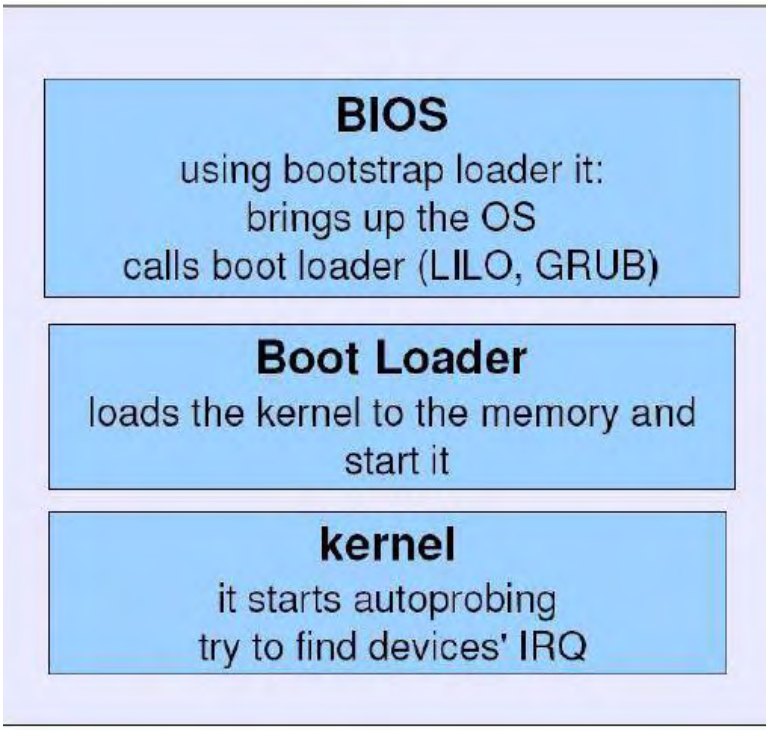
تحت الاسم lesson6.asm في القرص المرفق , قم بترجمة البرنامج و نسخه للقرص المرن باسم آخر

```
nasm lesson6.asm –o lesson6.bin
copy lesson6.bin a:\LOADER.BIN
```

و يصبح برنامج Hellowrld كالتالي:

```
;-------------------------------------------------------------------
;Hello World Operating System
;Disclaimer: I am not responsible for any results of the use of the contents
;of this file
;-------------------------------------------------------------------

;Here is the operating system entry point
;-----------------------------------------
begin:
        mov     ax, cs          ;Get the current segment
        mov     ds, ax          ;The data is in this segment
        cli                     ;disable interrupts while changing stack
        mov     ss, ax          ;We'll use this segment for the stack too
        mov     sp, 0xfffe      ;Start the stack at the top of the segment
        sti                     ;Reenable interrupts

        mov     si, msg                 ;load address of our message
        call    putstr          ;print the message

hang:
        jmp     hang            ;just loop forever.

;-----------------------------------------
;data for our program

msg     db      'Hello, World!', 0

;-----------------------------------------
;Print a null-terminated string on the screen
;-----------------------------------------
putstr:
        lodsb           ;AL = [DS:SI[
        or al, al       ;Set zero flag if al=0
        jz putstrd      ;jump to putstrd if zero flag is set
        mov ah, 0x0e        ;video function 0Eh (print char(
        mov bx, 0x0007      ;color
        int 0x10
        jmp putstr
putstrd:
        retn
```

الآن كل ما هو مطلوب منك نسخ ملف نظام تشغيلك ذو الاسم Loader.bin مع برنامج الإقلاع boot12.bin إلى القرص المرن و من ثم نسخ برنامج الإقلاع boot12.bin إلى قطاع الإقلاع على القرص المرن كما تعلمنا سابقا.أعد الإقلاع و استمتع بالجملة Helo,World!.يلخص الشكل التالي عملية الإقلاع في الحاسب.



الشكل2

2 6 -تعدد الإقلاع :Multi booting

أصبح الآن من البديهي القول أن كل نظام تشغيل يحتاج لمحمل إقلاع لتحميله و تشغيله و في البداية كان يتم كتابة برنامج إقلاع خاص بكل نظام تشغيل و كان لكل حاسب نظام تشغيل خاص به، و مع مرور الزمن شعر المستخدمون بالحاجة لوجود أكثر من نظام تشغيل على أجهزتهم بحيث يستطيعون تنزيل أكثر من نظام تشغيل و التنقل بين نظام و آخر عند إقلاع الحاسب. هذه الضرورة استدعت وجود معيار موحد لنواة نظام التشغيل بحيث تكون داعمة لمبدأ تعدد الإقلاع.يوجد ثلاث مالتالية:ية لواجهة محمل الإقلاع و يقوم هذا المعيار بتحديد ما يجب إضافته و تعديله للمبادئ التالية:

صيغة نظام التشغيل كما يراه محمل الإقلاع.
حالة الآلة عندما يبدأ محمل الإقلاع بتحميل النظام.
صيغة المعلومات التي يتم تمريرها من محمل الإقلاع لنظام التشغيل.

1 6 2 -صيغة نظام التشغيل

إن صيغ نظام التشغيل المدعومة في هذا المعيار هي صيغة الملفات الثنائية bin و التي لها اللاحقة التالية out و أيضا الملفات التنفيذية من النوع ELF و تجدر الإشارة هنا إلى أن النوعان السابقان يستخدمان في الأنظمة المبينة على فلسفة UNIX و منها نظامنا الذي بصدد شرحه.

1 1 6 2 -حالة الآلة

عنـدما يقـوم محمـل الإقـلاع باسـتدعاء نظـام التشـغيل يجـب أن تكـون الآلـة محققـة المواصفات التالية:

- CS must be a 32-bit read/execute code segment with an offset of 0 and a limit of 0xffffffff.
- DS, ES, FS, GS, and SS must be a 32-bit read/write data segment with an offset of 0 and a limit of 0xffffffff.
- The address 20 line must be usable for standard linear 32-bit addressing of memory (in standard PC hardware, it is wired to 0 at
- EAX must contain the magic value 0x2BADB002; the presence of this value indicates to the OS that it was loaded by a MultiBoot-compliant boot loader (e.g. as opposed to another type of boot loader that the OS can also be loaded from).
- EBX must contain the 32-bit physical address of the multiboot_info structure provided by the boot loader (see below boot up, forcing addresses in the 1-2 MB range to be mapped to the 0-1 MB range, 3-4 is mapped to 2-3, etc.).
- Paging must be turned off.
- The processor interrupt flag must be turned off.).

و أمـا بـاقي المسـجلات و الأعـلام فـي المعـالج غيـر معرفـة و هـذا يشـمل علـى وجـه الخصوص:

- ESP: the 32-bit OS must create its own stack as soon as it needs one.
- GDTR: Even though the segment registers are set up as described above, the GDTR may be invalid, so the OS must not load any segment registers (even just reloading the same values!) until it sets up its own GDT.
- IDTR: The OS must leave interrupts disabled until it sets up its own IDT

2 1 6 2 -صيغة معلومات الإقلاع

يحتـوي المسـجل EBX العنـوان الفيزيـائي لسـجل المعطيـات 'multiboot_info' و الـذي مـن خلالـه يقـوم محمـل الإقـلاع بتمريـر المعلومـات لنظـام التشـغيل و عنـدها يمكـن لنظـام التشغيل قراءة المعلومات منه. إن السجل multiboot_info و أقسامه الفرعية يمكن أن تتوضـع في أي مكان في الذ1كرة من قبل محمل الإقلاع و من مسؤولية نظام التشغيل تجنـب الكتابـة علـى ذلك المكان لكي لا تمحى المعلومات.

2 6 2 -صيغة multiboot_info

يأخذ السجل المتعلق بمعلومات سجل الإقلاع الصيغة التالية:

```
        +-------------------+
0       | flags             |   (required)
        +-------------------+
4       | mem_lower         |   (present if flags[0] is set)
8       | mem_upper         |   (present if flags[0] is set)
        +-------------------+
12      | boot_device       |   (present if flags[1] is set)
        +-------------------+
16      | cmdline           |   (present if flags[2] is set)
        +-------------------+
20      | mods_count        |   (present if flags[3] is set)
24      | mods_addr         |   (present if flags[3] is set)
        +-------------------+
28 - 40 | syms              |   (present if flags[4] or flags[5] is set)
        +-------------------+
44      | mmap_length       |   (present if flags[6] is set)
48      | mmap_addr         |   (present if flags[6] is set)
        +-------------------+
```

لن نذكر في هذه العجالة تفصيل حقول السجل حيث يمكنك الحصول عليها من الوثيقة المتعلقة بالمعيارية و سنمر فقط مرورا سريعا بما نحتاج لتعريفه.

2 7 ـ قسم التنفيذ Implementation

2 7 1 ـ تحقيق تعدد الإقلاع Multi booting

فيما يلي لمحات عن تنفيذ و تحقيق تعدد الإقلاع و ذلك من خلال الوحدة multiboot.h.كما ذكرنا أن تعدد الإقلاع يدعم صيغتي out و ELF و بالتالي يجب تعريفهما أيضا ضمن ملف الترويسة لدينا:

2 7 1 1 ـ تعريف الترويسة للصيغة Out

```
//! The symbol table for a.out format.
typedef struct {
        unsigned long tab_size;
        unsigned long str_size;
        unsigned long address;
        unsigned long reserved;
}
aout_symbol_table_t;
```

2 7 1 2 ـ تعريف ملف الترويسة للصيغة ELF

```
//! The section header table for ELF format.
typedef struct {
        unsigned long num;
        unsigned long size;
        unsigned long address;
        unsigned long shndx;
}
elf_section_header_table_t;
```

2 7 1 3 -تعريف سجل الإقلاع

بعد أن وضحنا المبادئ السابقة يمكن ببساطة تعريف سجل الإقلاع على النحو التالي:

```
typedef struct multiboot_info
{
        unsigned long flags;
        unsigned long mem_lower;
        unsigned long mem_upper;
        unsigned long boot_device;
        unsigned long cmdline;
        unsigned long mods_addr;
        union
        {
                aout_symbol_table_t aout_sym_t;
                elf_section_header_table_t elf_sec_t;
        } u;
        unsigned long mmap_length;
        unsigned long mmap_addr;
} multiboot_info_t;
```

2 7 1 4 -التأكد من صحة الإقلاع

بقي أن نتأكد أن عملية الإقلاع تمت بنجاح و يتم ذلك بمقارنة البت المتعلق ب boot_device و ذلك بالكود التالي و الذي سيكون طبعا ضمن البرنامج الرئيسي و بداية نظام التشغيل أي ضمن main على النحو التالي:

```
main ()
//! The multiboot information pointer from GRUB.
extern multiboot_info_t *boot_info;

// Get some multiboot informations.
        boot_info = (multiboot_info_t *)PHYSICAL((dword)boot_info);
        if (boot_info->flags & 0x02)
        {
                printk( KERN_INFO
                        "Total amount of conventional memory: %uKB\n",
                        ((size_t)boot_info->mem_lower) );
        }
        else
        {
                printk( KERN_EMERG
                        "\nError reading multiboot informations... halting
system!\n" );
        }
```

2.7.2 -محمل الإقلاع GRUB

2.7.2.1 -ميزات Grub

محمل الإقلاع GRUB (GRand Unified Bootloader) هو برنامج مجاني مفتوح المصدر للقيام بتحميل نظام التشغيل ويحتوي محمل الإقلاع هذا على الكثير من المميزات وهي:

يقدم واجهة رسومية لا بأس بها.
يقدم محرر أوامر متكامل.
يتعرف على أجزاء القرص الصلب FDISK.
يقدم دعم لعدة أنظمة ملفات مثل ... FAT, EXT2
يمكنه إقلاع أكثر من نظام تشغيل.

2.7.2.2 -الحصول على grub

يمكن الحصول على الملفات المصدرية للبرنامج و الملفات الثنائية المترجمة عن طريق الموقع التالي و باستخدام خدمة FTP:

```
Source code:ftp://alpha.gnu.org/gnu/grub/grub-0.92.tar.gz
Binaries:ftp://alpha.gnu.org/gnu/grub/grub-0.92-i386-pc.tar.gz
```

نحتاج لبعض الأدوات للعمل في بيئة وندوز و هي ARTCOPY or RAWRITE

```
http://www.execpc.com/~geezer/johnfine/index.htm#zero
http://uranus.it.swin.edu.au/~jn/linux/rawwrite.htm
http://www.tux.org/pub/dos/rawrite/
```

أما الصفحة الرئيسية للبرنامج فهي:

```
http://www.gnu.org/software/grub
```

2.7.2.3- متطلبات لا بد منها

من الأسهل القيام بعملية الترجمة و التنصيب من خلال استخدام نظام التشغيل لينكس و عليك التأكد من وجود الأدوات التالية لديك:

GCC:

مترجم لغة السي و يجب أن يكون إصداره GCC 2.95 و ما فوق و يمكنك الحصول عليه من الرابط:

```
http://gcc.gnu.org/.
```

GNU Make:

يجب أن يكون لديك دعم لخاصية Automake و التي تعتمد على GNU Make و يمكنك الحصول عليها من الرابط:

```
http://www.gnu.org/software/make/make.html
```

GNU binutils 2.9.1.0.23:

يجب أن تمتلك الأدوات الخاصة ب GNO من أجل ترجمة بعض ملفات البرنامج و خاصة تلك المكتوبة بلغة المجمع و يمكنك الحصول عليه من الرابط:

```
http://sourceware.cygnus.com/binutils
```

Texinfo 4.0:

يوجد بعض البريمجات (ماكرو) في المستندات و تحتاج لبرنامج قراءة المستندات السابق ويمكنك الحصول عليه من الرابط:

```
http://www.gnu.org/software/texinfo/texinfo.html
```

2.7.2.4- تهيئة التطبيق Configuring the GRUB

تتم تهيئة البرنامج بواسطة الأمر 'configure` ضمن محث الأوامر حيث يقوم بعملية فحص للملفات و الشفرة المصدرية و التأكد أنها كاملة و لا مشكلة بها و غير ناقصة كما يقوم بتهيئة بعض المتحولات و القيم اللازمة خلال عملية التنصيب.

2.7.2.5- ترجمة البرنامج

إن أسهل طريقة لترجمة الحزمة البرمجية هو إتباع الخطوات التالية:

قم بالانتقال للدليل الذي يحتوي مجلد البرنامج و ذلك باستخدام الأمر CD و بعد الوصول هناك قم بكتابة الأمر `./configure' لتهيئة الحزمة البرمجية على نظامك و بعد تنفيذ هذا الأمر سيقوم بعملية التهيئة و طباعة بعض العبارات على الشاشة التي توضح ماذا يجري.

قم بكتابة الأمر `make' للقيام بترجمة الحزمة.
هناك أمر اختياري يمكنك كتابته أيضا و هو `make check' لتشغيل فحص ذاتي للحزمة.
الآن قم بكتابة الأمر `make install' لتنصيب البرنامج و الملفات الثنائية و المستندات اللازمة.
يمكنك حذف الملفات الثنائية الفرعية التي تم توليدها أثناء عملية الترجمة لكي لا تختلط بالشفرة المصدرية للبرنامج و ذلك عبر الأمر `make clean'.
يمكنك حذف الملفات التي قامت عملية التهيئة بتوليدها بكتابة الأمر `make distclean'.
بشكل افتراضي يتم تنزيل الملفات في المسارات /usr/local/bin`/usr/local/man', etc/.
ويمكنك تغيير المسارات بواسطة استخدام الخيار التالي ضمن أمر `configure' و الخيار هو: `--PATH=prefix'

2.7.2.6- الملفات النهائية

بعد الانتهاء من عملية الترجمة ستحصل على الملفات الثنائية التالية.Stage1, stage2, menu.lst حيث يكون حجم المرحلة الأولى Stage1 512 بايت. وهي الملف الواجب كتابته على قطاع الإقلاع.

2.7.2.7- إنشاء قرص إقلاع

الآن بعد أن حصلنا على الملفات الثنائية علينا كتابتها على قطاع الإقلاع في القرص الصلب و تنزيل البرنامج محمل الإقلاع عليه.

2.7.2.8- مراحل تنزيل البرنامج

بداية تحتاج لقرصين مرنين أحدهما تمت تهيئته بنظام ملفات ب\يتعرف عليه محمل الإقلاع GRUB مثل FAT12 أو EXT2 و الآخر عادي غير مهم لأنه ستحذف البيانات الموجودة عليه و سيستخدم كقرص وسيط.

نبدأ مع القرص الأول الذي يتعرف عليه محمل الإقلاع و ننسخ عليه الملفات الثنائية الناتجة معنا و ذلك على الدليل "/boot/grub/" وطبعا هذا يعني أن نقوم بإنشاء المجلدات السابقة لوضع الملفات ضمنها.

قم بدمج الملفين "stage2" and "stage1" إلى ملف واحد و قم بتسميته boot و يمكنك القيام بذلك عبر نظام التشغيل وندوز بالأمر التالي:

```
copy /b stage1 + stage2 boot
```

أما بواسطة لينكس فيتم بواسطة الأمر التالي:

```
cat stage1 stage2 >boot
```

قم الآن بكتابة الملف الناتج الجديد bootإلى القرص الثاني بدءا من قطاع الإقلاع و هذا يعني إتلاف القرص وضياع بياناته بالطبع و يمكن القيام بذلك ضمن وندوز عبر الأمر:

```
partcopy boot 0 168000 -f0
```

و ضمن لينكس عبر الأمر:

```
cat boot >/dev/fd0
```

طبعا يمكنك استخدام برامج أخرى للقيام بعملية النسخ ضمن وندوز مثل مجموعة bootprog.

أعد إقلاع الحاسب تاركا القرص الثاني فيه.

سيتم الإقلاع عبر القرص و ظهور عبارة GRUB و عندئذ قم بإزالة هذا القرص و وضع القرص الأول الذي يحتوي الملفات Stage1,stage2 , menu.lst و اكتب التعليمة:

```
setup (fd0)
```

حيث تشير fd0إلى القرص المرن.

أصبح لدينا الآن القرص الأول قابل للإقلاع و يحتوي على برنامج GRUB عليك الانتباه و المحافظة على الملفات من التعديل أو الحذف.

2 7 3 -إقلاع نواة نظام التشغيل

الآن و بعد أن أصبح كل شيء جاهز كيف سنستثمر هذا المحمل في إقلاع نواة نظام تشغيلنا و أنظمة تشغيل أخرى. بداية يجب أن يكون تصميم النواة نفسه يدعم تعدد الإقلاع Multiboot بحيث تحتوي النواة على Multiboot header كما تم شرح ذلك مسبقا.

ضع النواة في مكان تعرفه.
أعد الإقلاع بوجود القرص.
اكتب التعليمة

```
root (hd0,1)
```

و التي تقوم بالانتقال للقرص الصلب الأول على الحاسب.

أخبر البرنامج عن موقع النواة

```
kernel /krnl.elf
```

2 7 4 -كتابة ملف:menu.lst

إن تجهيز هذا الملف أسهل حيث نضع فيه ما سبق ونضع موقع النواة و عنوانها و ما هي الخيارات التي ستظهر على الشاشة في عملية الإقلاع و فيما يلي مثال على ملف يحتوي على عدة أنظمة تشغيل بحيث يشمل عدة خيارات:

```
# Sample boot menu configuration file
#  default - boot the first entry.
default 0
# if have problem - boot the second entry.
fallback 1
# after 30 sec boot default.
timeout 30
# GNU Hurd
title  GNU/Hurd
root   (hd0,0)
kernel /boot/gnumach.gz root=hd0s1
module /boot/serverboot.gz
# Linux - boot ot gsecond HDD
title  GNU/Linux
kernel (hd1,0)/vmlinuz root=/dev/hdb1
# booting Mach - get kernel from floppy
title  Utah Mach4 multiboot
root   (hd0,2)
pause  Insert the diskette now!!
kernel (fd0)/boot/kernel root=hd0s3
module (fd0)/boot/bootstrap
# booting OS/2
title OS/2
root  (hd0,1)
makeactive
# chainload OS/2 bootloader from the first sector
chainloader +1
# For booting Windows NT or Windows95
title Windows NT / Windows 95 boot menu
root       (hd0,0)
makeactive
chainloader +1
# Colors change:0).
title Change the colors
color light-green/brown blink-red/blue
```

أما بالنسبة لنواة نظام تشغيلنا فيكون الملف كالتالي:

```
#
# Automatically generated by make: do not edit!
#
default 0
title MyOs
root (fd0)
kernel /kernel/mr32.gz
```

حيث يتم كتابة العنوان الذي سيظهر على الشاشة ضمن .title

5 7 2 -النواة متعددة الإقلاع

يقوم البرنامج GRUB بتحميل نواة نظام تشغيل بشرط دعمها لمبدأ تعدد الإقلاع و هو ما سيتم شرحها في الفصل التالي و بالإضافة لشرط آخر أن تكون من النوع ELF أي مترجمة تحت بيئة لينكس و ليس وندوز حيث أن النوع ELFمكافئ للنوع EXEفي وندوز و تكون

النواة محققة لمبدأ تعدد الإقلاع عندما تحتوى على ترويسة تعدد الإقلاع أي Multiboot header و هذه الترويسة تحقق

يجب أن يكون رصف المقاطع بمقدار 32 بايت لكل مقطع.
يجب أن تظهر الترويسة في أول 8 كيل من ملف النواة.

يفهم المحمل ملفات ELF مباشرة فإذا كانت نواتك عبارة عن ELFفيكفي استخدام الترويسة التالية من أجل تعدد الإقلاع multiboot التالية:

```
; NASM syntax
   MULTIBOOT_PAGE_ALIGN   equ 1<<0
   MULTIBOOT_MEMORY_INFO  equ 1<<1
   MULTIBOOT_HEADER_MAGIC equ 0x1BADB002
   MULTIBOOT_HEADER_FLAGSequMULTIBOOT_PAGE_ALIGN                    |
MULTIBOOT_MEMORY_INFO
   CHECKSUM                          equ  -(MULTIBOOT_HEADER_MAGIC  +
MULTIBOOT_HEADER_FLAGS)
   ; The Multiboot header
      align 4
      dd MULTIBOOT_HEADER_MAGIC
      dd MULTIBOOT_HEADER_FLAGS
      dd CHECKSUM
```

بعد إضافة الشفرة السابقة للنواة نعيد ترجمتها و نتأكد أنها متوافقة مع المحمل من خلال الأداة "mbchk" التي تفحص توافقيتها مع مبدأ تعدد الإقلاع.

2 8 -المراجع

http://cse.unl.edu/~jgompert/OS/
http://www.nondot.org/sabre/os/articles
http://www.nondot.org/sabre/os/files/Booting
http://my.execpc.com/~geezer/osd/boot/index.htm
http://www.alkhawarzmi.com
Write Your Own Operating System Tutorial

3 -النمط المحمي

مُقدِّمة

تقدم لنا شريحة Intel80386 العديد من الميزات و الخصائص الجديدة للتغلب على نقاط الضعف في شريحة Intel8086 و التي من بينها عدم قدرة المعالج 8086 على تقديم الحماية المطلوبة عند التعامل مع المقاطع الذاكرية. بالإضافة عدم دعم المعالج لتقنيات تعدد المهام و العمل في نمط الذاكرة الافتراضية و عدم قدرته على التعامل مع المقاطع الذاكرية التي بحجم أكبر من KB64. و كما يوفر لنا معالج الـ 80386 التوافقية الكاملة مع المعالج 8086. و يعتبر إصداري كل 80386 و الـ 80486 متطابقين مع بعضهما البعض تماما.

إن شريحة الـ Intel80286 تم فيها توفير العديد من التحسينات و الميزات التي لم تكن متوفرة في 8086, ومن بين هذه التحسينات التي طرحت في 80286 هي إمكانية عمل المعالج في نمطين مختلفين هما النمط المحمي و النمط الحقيقي بينما كان المعالج 8086 يعمل فقط في النمط الحقيقي. و كما أن معالج الـ 80286 يؤمن توافقية كاملة مع سابقيه من المعالجات و ذلك يتوفر عندما تم منح المعالج قدرة العمل في النمط الحقيقي, و لكن عند انتقال المعالج 80386 للعمل في النمط المحمي فإنه يختلف بطريقة عمله الداخلية اختلافا جذريا عن المعالج 8086 و تصبح العديد من البرامج التي كانت تعمل على معالجات الـ 8086 غير قادرة على العمل في هذا النمط من عمل المعالج –النمط المحمي- و من بين هذه البرامج على سبيل المثال نظام التشغيل الـ DOS.و بنفس الكلام السابق ينطبق على معالج 80386 حيث قمت هذه الشريحة من المعالجات كتطوير لكل من شريحتي 8086 و 80286 حيث حوت على كل ميزات هاتين الشريحتين و أضيفت لها ميزات جديدة. والتي من بينها قدرة المعالج على عنونة مجالات عناوين ذاكرية أكبر من ما كان متاحا في معالجات 8086 و 80386. حيث كان عدد خطوط العنونة في 8086 و 80286 هي 16بت بينما في 80386 أصبحت32بت. و مع قدرة المعالج على تأمين التوافقية التامة مع معالجات 8086 العاملة بنظام النمط الحقيقي و مجال العنونة 16بت. و بالإضافة للميزات التي ذكرناها في 80386 أيضا تم تطوير نمط عمل و هو النمط العمل الحقيقي الوهمي. Virtual Real Mode86.

يتميز هذا النمط V 86 بأنه نظام عمل في النمط المحمي و لكنه يؤمن محاكاة لنمط العمل الحقيقي ضمن نمط العمل المحمي. وبالتالي البرامج المصممة للعمل في النمط الحقيقي تصبح قادرة على العمل في النمط المحمي دون أي تعديل فيها أو تبديل نظام عمل المعالج. يعني من خلال هذا النمط V86 نستطيع أن نشغل نظام DOS مثلا دون أن نغير نظام DOS أو تبديل نظام عمل المعالج.

3 .1 -ميزات استخدام النمط المحمي

3 .1 .1 -الوصول إلى 4GB من الذاكرة

تعد هذه الميزة من أكثر الفوائد المتوخاة من هذا النمط بالمقارنة مع النمط الحقيقي. حيث تستطيع البرامج العاملة في النمط المحمي أن تستخدم ذاكرة بحجم 4 GB و ذلك لكل مقطع على حدا من أنواع المقاطع الأساسية وهي مقطع الشفرة (CODE SEGMENT) و مقطع المعطيات (DATA SEGMENT) و مقطع المكدس (STACK SEGMENT).

من أخرى يمكننا من خلال توظيف بعض الميزات الغير مشهورة في معالج 8086 تمكين هذا المعالج الوصول لذاكرة فوق 1MB و ذلك بغرض التخزين فقط, بينما لا نستطيع الاستفادة من هذه الميزة فيما يخص التعامل مع مقاطع التكديس و الشفرة، و لكن وجود ذاكرة بحجم 4GB في الحاسب أمر غير اعتيادي و غالبا ما تكون أحجام الذواكر أقل من ذلك, هذا الكلام يقودنا للميزة التالية ميزة الذاكرة الافتراضية.

3 .1 .2 -الذاكرة الافتراضية

إن وحدة إدارة الذاكرة في معالجات 80386 يمكنه تنفيذ ما يسمى بتقنية الذاكرة الافتراضية Virtual Memory و هذه التقنية تستخدم لخلق إيحاء للبرامج بأنه يوجد في الحاسب ذاكرة رئيسية حجمها 4GB بينما في الحقيقة تكون حجم الذاكرة الفعلي أقل من ذلك بكثير. و هذه التقنية تقوم على الاستفادة من الذواكر الثانوية مثلا القرص الصلب و تستخدمها كذاكرة تخزين رئيسية للبرامج, أي يمكننا أن نتخيل هذه التقنية أنه تقوم بتوسيع الذاكرة الرئيسية التي حجمها أقل من 4GB تمددها من خلال أجهزة التخزين الثانوية إلى 4GB, و هذا بافتراض انه تتوفر لدي مساحة ذاكرة بحجم 4GB في جهاز التخزين الثانوي.

3 .1 .3 -ترجمة العناوين

في النمط المحمي تعمل البرامج على ما يسمى بالعناوين الذاكرية المنطقية Logical Address و من ثم يأتي المعالج 80386 ليقوم بترجمة هذه العناوين المنطقية إلى عناوين خطية Linear Address التي هي بطول 32 خانة. و من ثم تقوم وحدة إدارة الذاكرة MMU بترجمة هذا العنوان الخطي إلى عنوان فيزيائي حقيقي Physical Address. و في حال لم تكن وحدة إدارة الذاكرة MMU فعالة فإن العنوان الخطي هو نفسه يكون العنوان الفيزيائي، أي إذا كان لدينا على سبيل المثال العنوان المنطقي التالي B800:0010, فإن العنوان المنطقي المقابل لهذا العنوان الفيزيائي هو B8010H و على اعتبار أن وحدة إدارة الذاكرة معطلة أي غير فعالة فعندها يكون العنوان السابق هو نفسه العنوان الفيزيائي.

3 .1 .4 -تحسين في آلية التقطيع الذاكري

- في النمط الحقيقي المساحة الذاكرية لكل مقطع ثابتة و محددة بـ 64KB بينما في النمط المحمي يكون حجم القطعة الذاكرية متغير و يتراوح من 1B إلى 4GB.
- إن أي محاولة من قبل برنامج للوصول إلى الذاكرة فيما هو خارج المقطع الذاكري سوف يؤدي إلى حدوث مقاطعة في المعالج و إظهار رسالة خطأ.

- في حال كان حجم المقطع الذاكري 4GB فإن أي محاولة للوصول إلى ما بعد الـ 4GB فإن المعالج سوف يقوم بعملية التفاف لبداية الذاكرة.
- يمكن للمقاطع الذاكرية أن تبدأ من عنوان ذاكري.
- يمنح القدرة للمبرمج على تخصيص المقاطع الذاكرية فما يتم تحديد كمقطع للشفرة فإن سيكون استخدام محصورا فقط بالشفرة و أي محاولة للكتابة في هذا المقطع الذي يكون فقط للقراءة فسوف يتم مقاطعة عمل المعالج لتظهر رسالة خطأ.

3 1 5 -حماية الذاكرة

يقدم المعالج 80386 ميزة حماية الذاكرة. و هذا يعني تأمين مستويات من الحماية لبعض مناطق الذاكرة حسب أهمية المعلومات الموجودة في هذه المناطق. على سبيل المثال المنطقة التي تكون فيها نواة نظام التشغيل مثلا تعتبر منطقة مهمة جدا و أي محاولة تخريب فيها أو كتابة فيها من قبل البرامج الأخرى يعني أنه سوف يتم التأثير على عمل النظام كاملا و بالتالي يقوم مبرمجو نظام التشغيل عادة بوضع مستوى حماية أكبر للمنطقة الموجودة فيها نواة النظام و هي تكون عادة أعلى مستويات الحماية التي يقدمها المعالج 80386.

3 1 6 -حماية الاجرائيات

يمكن أن نتخيل حماية الاجرائيات بشكل مشابه لما رأيناه في حماية الذاكرة في الأعلى. حيث يتم حماية البرامج المختلفة من بعضها البعض فيتم حظر المنطقة الخاصة ببرنامج ما عن البرامج الأخرى و لا يسمح الوصول لهذه المنطقة إلا من قبل هذا البرنامج.

بينما في نظام التشغيل مثلا يمكن مثلا أن نتيح للنظام أن يصل إلى جميع المناطق الذاكرية الخاصة بالبرامج الأخرى و ذلك لأن نظام التشغيل يملك مستوى من الحقوق التي تخوله للوصول لهذه المناطق على خلاف البرامج العادي و هذا الأمر ضروري و خاصة لدى حصول المقاطعات أو استدعاء إجرائيات النظام أو تمرير الرسائل بين البرامج. و بالتالي يضطر هنا نظام التشغيل أن يكون له حق في الوصول إلى مناطق المعطيات الخاصة بالبرامج الأخرى، بالمقابل نجد أنه يمكن للبرامج أن يكون له حق في الوصول المحدود لمنطقة المعطيات الخاصة ببرنامج نظام التشغيل.كل ما سبق من حديث عن السماحيات و الحقوق يتم تنفيذه من خلال وحدة إدارة الذاكرة MMU و تقنية التصفيح الذاكري Memory Paging.

3 1 7 -مسجلات 32 خانة

إن جميع المسجلات العامة في معالجات 80386 هي بطول 32 خانة. و أسماء هذه المسجلات هي نفسها أسماء المسجلات التي كانت في معالجات 8086 (AX BX CX..) و لكن يضاف إليها للعمل مع عرض 32 خانة حرف A أي تصبح EAX EBX و هكذا و يزيد عن مسجلات المعالج 8086 في 80386 مسجلي FS GS.

هذه المسجلات يتم الوصول إليها في جميع أنماط عمل المعالج و يمكن من خلال النمط الحقيقي استخدام المسجلات العامة بحجم 32 خانة و لكن لا تستخدم في أغراض العنونة و إنما فقط التخزين و تناقل المعطيات بينما الاستفادة الأكبر من حجم 32 خانة هي في النمط المحمي، وعندما أستخدم هذه المسجلات في النمط المحمي بحجم 32 خانة فإن ذلك سيساعدني على اختصار اسطر عديدة من الشفرة البرمجية في لغة التجميع Assembly.

3 .1 .8 -تحسين أنماط العنونة

في النمط الحقيقي عندما كنا نقوم بالوصول الغير مباشر إلى العناوين الذاكرية كان ذلك يتم من خلال ثوابت, و التي كانت تتمثل في المسجلات BX و BP و SI و DI. بينما في النمط المحمي صار لدينا إمكانية تشكيل هذه العناوين من خلال أي مسجل من المسجلات الموجودة في المعالج. و بالإضافة لذلك يمكننا أن نضيف لمسجل الفهرسة عامل انتقال نسبي. و يكون الانتقال من عنوان لعنوان بخطوات أكبر من 1. و هذا يمكنني من كتابة هكذا نوع من الأوامر:

```
MOV   EBX, [EDI][EAX*8]+2
```

3 .1 .9 -دعم لتعدد لمهام

إن معالج 80386 عندما يعمل في النمط المحمي يوفر من خلال هيكليتيه يوفر لنا بنية خاصة تتيح لنا إمكانية حفظ الإجرائية الحالية و الانتقال لتنفيذ إجرائية أخرى و من ثم العودة لتنفيذ هذه الإجرائية, و عملية التبديل هذه بين الاجرائيات تتم من خلال تعليمة تجميع وحيدة. و هذه الميزة هي ميزة رهيبة لدى تصميم أنظمة التشغيل و خاصة أنظمة تشغيل الزمن الحقيقي.

كما يدعم المعالج 80386 بنمطه الجديد ما يسمى بالمعالجات المترابطة حيث انتهاء مهمة معينة سوف يتم الرجوع لمتابعة تنفيذ المهمة الأصلية المتحدرة منها هذه المهمة.باستخدام ما يسمى الـ BackLink. و هذه الميزة سوف نأتي على ذكرها في Multiprocessing.

3 .1 .10 -تتبع الأخطاء في العتاد الصلب

تم تصميم معالجات 80386 بطريقة تمكنك من العمل بنمط الخطوة بخطوة و هذا النمط مفيد جدا للمبرمجين عند تنقيحهم و تتبعهم لسير عمل برامجهم تعليمة بتعليمة. و هذه الميزة متاحة في كلا النمطين المحمي و الحقيقي.

3 .2 -العنونة في النمط الحقيقي

في معالجات Intel8086 يتم تنظيم الذاكرة على أساس حجرات كل حجرة مؤلفة من ثمان خانات أي كل حجرة حجمها بايت واحد و بالتالي كما نعلم عندما نحاول تخزين بايت فإن هذا البايت يتم وضعه في حجرة ذاكرية بشكل مباشر ودون أي التباس و لكن إذا ما أردنا تخزين قيمة بحجم بايتين فإن المعالج هنا سوف يأخذ البايت الأكثر أهمية و يخزنه في العنوان الأول أي الأقل و يأخذ البايت الأقل أهمية و يخزنا في العنوان التالي للعنوان السابق. إن طريقة العمل هذه للمعالج قد تسبب ارتباكا في البداية للمبرمج و لكن ستعتاد على هذه الطريقة بعد فترة، مثلا لو كان لدي القيمة التالية:B800H فإن هذه القيمة ستخزن في الذاكرة على الشكل التالي: 00H ومن ثم B8H، ضمن عائلة منتجات Intel للمعالجات تعرف تقنية فهرسة الذاكرة باسم تقنية التقطيع Segmentation و هذه التقنية تقوم على تقسيم الذاكرة إلى مقاطع، و المقطع هو عبارة عن منطقة من الذاكرة و هذه المنطقة في النمط الحقيقي يكون حجمها ثابت و يساوي 64KB و كما يمكنني أن أقوم بتشكيل 64KB من المقاطع هذه, و لكن هذه المقاطع التي يمكن أشكلها سوف تتداخل فيما بينها و من الممكن أن يكون عنوان ذاكري ينتمي لمقطعين في آن واحد و يكون هذا التداخل بشرط مثلا لو كان لدي مقطعين أن يبدأ المقطع الثاني بعد 16 حجرة من بداية المقطع الأول. أي بعد 16 بايت من المقطع الأول.

وبحسبة بسيطة على الشكل التالي: 16KB*16Byte = 1MB و هذا ما يفسر لنا عدم قدرة المعالج 8086 على عنونة الذاكرة بحجم أكبر من 1MB, و على هذا الأساس نظام التشغيل DOS لا يمكنها أن يرى ذاكرة ما بعد الـ 1MB، و المقاطع يتم ترقيمها بدء من 0000h وحتى FFFFH و كل مقطع نستخدم معه ما يسمى بالإزاحة. و هذه الإزاحة تستخدم للوصول إلى حجرة معينة ضمن نفس المقطع، أي تراكب مقطع + إزاحة يولد لي عنوان حجرة ذاكرية.

الآن بعد أن حصلنا على هذا التراكب فإننا نحتاج لإزاحة هذا العنوان بمقدار نصف بايت حتى نصل للعنوان الفيزيائي المطلوب و هذه المعلية تتم بأكملها من خلال المعالج ودون حاجة لتدخل المبرمج لتنفيذها. أي عملية الترجمة.

Address: 8123:FFEC

8123		* 16
81230	FFEC	+ 16-bit Offset
9121C		Physical Address

الشكل3

3 3 -العنونة في النمط المحمي

الآن بعد أن فهمنا طريقة العنونة في النمط الحقيقي ننتقل لفهم كيفية عنونة الحجر الذاكرية في النمط المحمي. فكما رأينا في النمط الحقيقي تتم عنونة الذاكرة والوصول إلى كل حجرة من حجرات الذاكرة من خلال استخدام مفهوم المقاطع والازاحات ضمن المقاطع, و أيضا في النمط المحمي لدينا نفس الآلية في الوصول إلى الحجرات الذاكرية باستخدام المقاطع و الإزاحات و لكن ببنية مختلفة، و هذه البنية الجديدة تعتمد على الهيكلية الجديدة لمعالج 80386, و أصبحت تسمح لنا هذه البنية بعنونة ذاكرة GB4.

تقوم هذه البنية على ما يلي:

جدول الواصفات Descriptor Table
مؤشر إلى جدول الواصفات Descriptor Table Register
ناخب الواصفات ضمن الجدول.Selector Descriptor.

من هذه البنية يمكننا تعرف آلية العنونة في النمط المحمي, فجدول الواصفات يحتوي على حقول و كل حقل من هذه الحقول و الذي يسمى واصفة Descriptor تحتوي على المعلومات التالية:

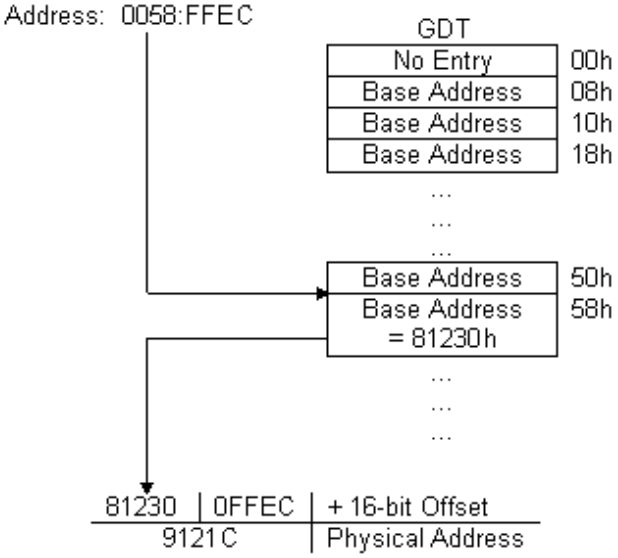
عنوان بداية المقطع
حجم المقطع
أمور أخرى لتحديد مواصفات المقطع

سنأتي على ذكر حقول الواصفة بالتفصيل فيما بعد أما الآن سنقتصر على شرح الآلية بشكل عام و لكن نضع الآن بالحسبان أن حجم كل حقل من حقول الجدول هو 8Byte أي كل واصفة حجمها 8Bytes.يتم تخزين عناوين المقاطع في الواصفات و هذه الواصفات تكون في

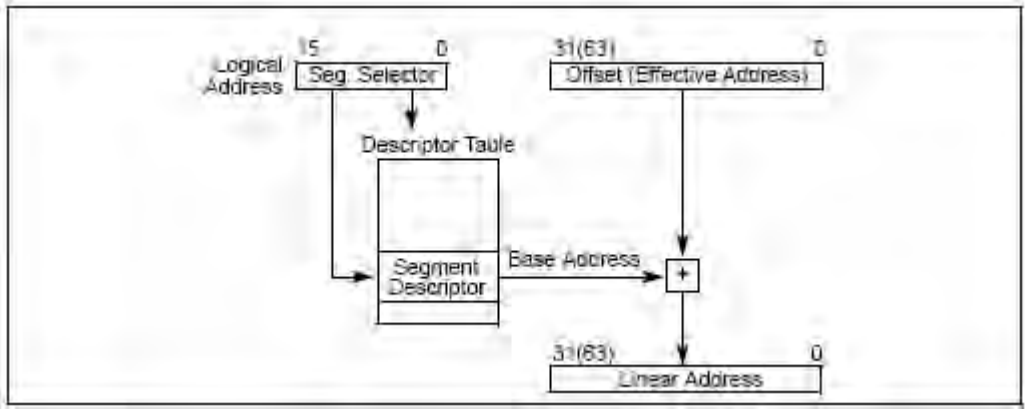
جدول الواصفات و يتم الوصول لكل واصفة من خلال مؤشر الواصفات, و يتم تحديد جدول الواصفات الذي سوف يتم انتخاب واصفاته من خلال مؤشر جدول الواصفات. إلى حد الآن قمنا بانتخاب المقطع الذي سنعمل عليه.

إن هدفنا الأخير من العملية هو الوصول إلى عنوان حجرة محددة في الذاكرة. و نحن حسب ما سبق وصلنا إلى عنوان مقطع ذاكري أي مجموعة من الحجرات و لكي نصل إلى الحجرة فإنه يلزمنا إزاحة عن رأس المقطع الذي تم انتخابه.

إن الآلية السابقة في عنونة الذاكرة هي كما في النمط الحقيقي آلية تتم تلقائيا من قبل المعالج. و كل ما علينا القيام به حتى تتحقق هذه الآلية هو تهيئة جداول الواصفات و مؤشرات الجداول بالقيم البدائية، و يوضح الشكل التالي لنا بشكل أكبر آلية العنونة في النمط المحمي، حيث نجد من الشكل السابق توضيح لكيفية الانتقال من العنوان المنطقي Logical Address إلى العنوان الفيزيائي في بيئة النمط المحمي.



الشكل4



الشكل5 آلية العنونة في النمط المحمي

الآن لكي نرسخ المعلومات بشكل أفضل نعود للحساب الرقمي.

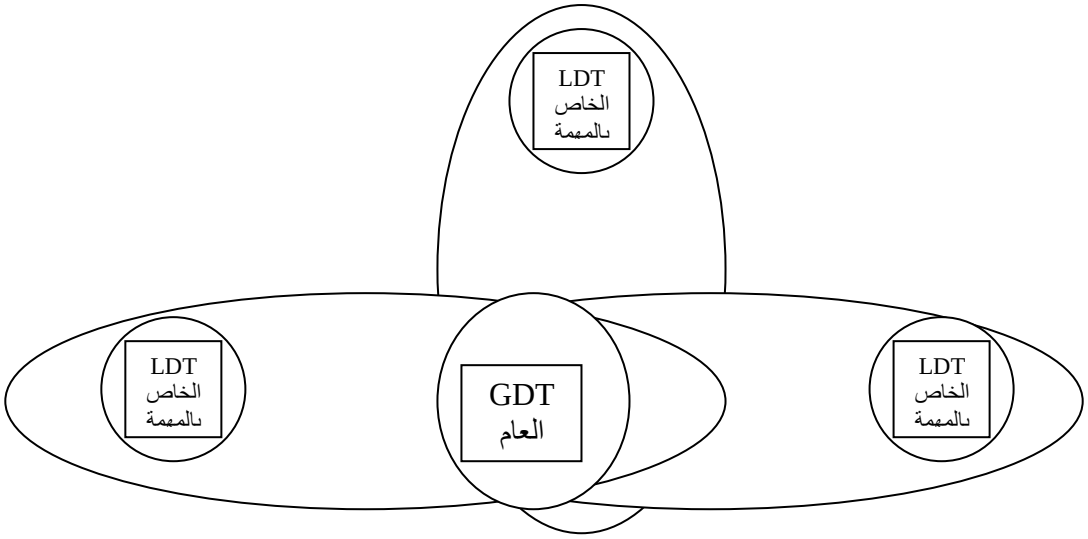
لدينا جدول بالواصفات حجم كل حقل من حقول هذا الجدول 8Bytes
لدينا ناخب للوصول إلى هذه الواصفات بحجم 16-3 = 13 خانة
يمكننا أن ننتخب من خلال الناخب 2^{13} = 8129 واصفة –أي حقل من حقول الجدول-

بالتالي يصبح لدينا حجم جدول الواصفات 8 * 213 = Byte65536 أي حجم جدول الواصفات 64KB.

قبل أن ننتقل للفقرة التالية اذكر أنه لدي نوعين من جداول الواصفات:

جدول الواصفات العام (Global Descriptor Table) GDT.
يستخدم جدول الواصف العام من أجل عنونة المقاطع الذاكرية المتشارك عليها من قبل جميع مهمات النظام أو لنقل من اجل مقاطع نظام التشغيل, أي هي بمثابة المتحولات العامة في النظام التي يمكن لجميع المهام الوصول إليها
جدول الواصفات المحلي (Local Descriptor Table) LDT.
تستخدم جداول LDT من أجل عنونة المقاطع الخاص بكل مهمة من المهام, حيث لكل مهمة يمكن أن نفرد جدول بالمقاطع خاص بها لوحدها و للمهمات المتحدرة من هذه المهمة و لا يمكن لباقي المهام الوصول لجدول هذه المهمة و بهذه الطريقة نكون قد حمينا المهام من بعض البعض.

يوضح الشكل التالي تشارك المهام على جدول الواصفات العام و كيف يتم إفراد جدول واصفات محلي خاص بها.



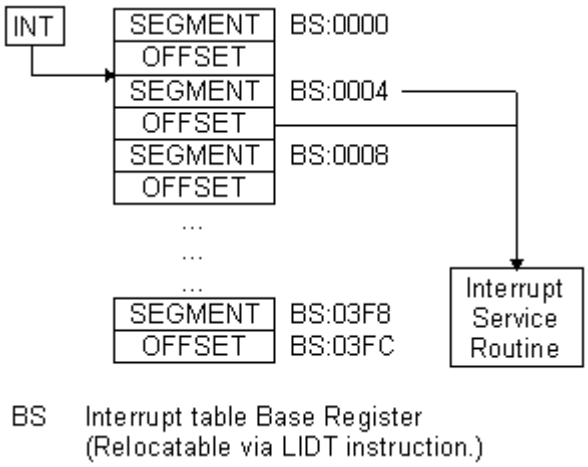
الشكل6

و كملاحظة أخيرة نقول و هي أن المعالج في لحظة واحدة لا يستطيع أن يتعامل إلى جدول واصفات واحد و لكي نقوم بالتبديل بين جداول الواصفات فعلينا القيام بتغير قيمة مؤشر جدول الواصفات GDTR أو LDTR و هي عبارة عن مسجلات موجودة في المعالج.

3 4 -جدول مقاطعات النمط الحقيقي

تستطيع معالجات 80386 التعامل مع 256 مقاطعة مختلفة و هذه المقاطعات تبدأ عناوينها الذاكرية بدء من العنوان 0000.0000 و هذا العنوان هو عنوان البداية لجدول أشعة المقاطعات. إن هذا الجدول يحتوي على حقول كل حقل بحجم 4Byte و يحتوي كل حقل على مؤشر لبداية برنامج خدمة المقاطعة حيث يشكل المؤشر كل من عنوان القطعة و الإزاحة ضمن القطعة، وبالتالي نحن نجد أنه لدينا 256مقاطعة*4Byte = 1024Byte حجم جدول أشعة المقاطعات في النمط الحقيقي، و لفهم آلية عمل جدول أشعة المقاطعات في النمط الحقيقي لنرى هذا المثال.

في حال قدوم المقاطعة رقم 2 INT2 فإن المعالج يوقف عمله بشكل مؤقت بعد أن يقوم بحفظ حالته من حفظ لمحتويات مسجل IP و CS و محتويات مسجل المكدس و أعلام المعالج و من ثم ينتقل لتنفيذ الكود الموجود في العنوان المحتوى ضمن الحقل 0000.0008H في جدول أشعة المقاطعات. حيث يسمى هذا الكود ببرنامج خدمة المقاطعة Interrupt Service Routin أو اختصارا بالـ ISR و سنمر على هذا المصطلح فيما بعد بشكل كبير في النمط المحمي. و عادة ما يتم إنهاء برنامج خدمة المقاطعة بوضع تعليمة الـ IRET في آخر البرنامج و ذلك للعودة لإخبار المعالج بالعودة لمتابعة تنفيذ البرنامج الأخير الذي كان ينفذه قبل قدح المقاطعة.



الشكل7

3 5 -المقاطعات الصلبة

إن المقاطعات الصلبة في الحاسب الشخصي هي عبارة عن إشارات خارجية تأتي من قبل الأجهزة المحيطة إلى المعالج لتقوم بإخباره عن أحداث معنية قد وقعت. حيث لدى ورود هذه الإشارات إلى المعالج فإن المعالج يترك كل شيء يقوم بتنفيذه ويلتفت للمقاطعة و يقوم بالاستجابة لها من خلال تنفيذ برنامج مخصص لدى كل ورود لهذه المقاطعة يتم تنفيذه. و بعد أن ينهي المعالج تخديمه لهذه المقاطعة يعود ليتابع ما كان يقوم به.

إن المقاطعات الصلبة يمكن أن ترد للحاسب من مجموعة من المصادر الخارجية, فعلى سبيل المثال لوحة المفاتيح تقوم بإرسال مقاطعة للحاسب لدى كل عملية ضغط أو تحرير للمفتاح على لوحة المفاتيح كما أن المنفذ التسلسلي لدى كل عملية استقبال تتم عليه يقوم بتوليد مقاطعة

لعمل المعالج و أيضا القرص الصلب والطابعة وكما أن المعالج يولد مقاطعة بنفسه و هي المقاطعة INT0 التي تحدث لدى قيام المعالج بتقسيم عدد على الرقم صفر. وفي النمط المحمي عملية معالجة المقاطعات هي من أهم الأمور و أي خطأ في كتابة برنامج معالجة المقاطعة من المحتمل أن يسبب مشاكل كبيرة في عمل الحاسب و النظام بشكل عام.إن عملية استقبال الإشارات و الأحداث من الأجهزة المحيطية و إرسالها إلى المعالج تتم من خلال دارة التحكم بالمقاطعات PIC 2859.

لدينا حسب معالجات INTEL ما يسمى بخطوط المقاطعة IRQ و هي مرتبة من IRQ0-IRQ15 و كل خط مقاطعة من هذه الخطوط يرتبط بشعاع مقاطعة, فمثلا مقاطعة لوحة المفاتيح IRQ1 ترتبط بشعاع المقاطعة رقم 9 أي INT9. إن عملية توزيع خطوط المقاطعة على أرقام المقاطعات يتم التحكم بها من خلال برمجة شريحة التحكم بالمقاطعات PIC8259 و بالتالي نستطيع توزيع خطوط المقاطعة على أشعة المقاطعات وفقا للعناوين التي نريدها و هذا ما سنقوم به بعد قليل لدى عملية التهيئة للنمط المحمي.

و أيضا شريحة التحكم بالمقاطعات مسئولة عن عملية ترتيب أولويات عمل المقاطعات في الحاسب, حيث يتم برمجة هذه الشريحة لكي تتحكم بأولويات المقاطعات وفقا للترتيب الذي يريده المبرمج, وعادة في أنظمة التشغيل يكون ترتيب أولويات المقاطعات يبدأ بالتسلسل من الـ IRQ0 ذات الأولوية الأعلى و انتهاء بالمقاطعة IRQ15 ذات الأولوية المنخفضة. فمثلا لدى ورود مقاطعة المؤقت (TIMER) IRQ0 و بعدها في حال ورود مقاطعة لوحة المفاتيح IRQ1 فإن حسب مبدأ الأولويات فإن مقاطعة لوحة المفاتيح لن يتم الاستجابة لها و لن يتم تخديمها إلا بعد أن يتم تخديم مقاطعة المؤقت. أي بعبارة أخرى يمكننا أن نقول أن مقاطعة المؤقت قامت بمقاطعة مقاطعة لوحة المفاتيح.

في الحواسب الشخصية يوجد لدينا عادة شريحتي تحكم بالمقاطعات PIC1 PIC2 و يتم وصل هاتين الشريحتين من خلال ما يسمى بالـ CASCADING. حيث كل شريحة من هذه الشرائح تتحكم بعمل ثمان مقاطعات من مقاطعات الحاسب, حيث الشريحة PIC1 تتحكم بالمقاطعات من 0-7 و الشريحة PIC2 تتحكم بالمقاطعات من 7-15، و يبين الجدول التالي أهم المقاطعات الصلبة في الحاسب الشخصي:

Interrupt	IRQ Number	Description
00H	-	Divide by zero or divide overflow
02H	-	NMI (Non-maskable Interrupt)
04H	-	Overflow (generated by INTO)
08H	0	System timer
09H	1	Keyboard
0AH	2	Interrupt from second PIC
0BH	3	COM2
0CH	4	COM1
0DH	5	LPT2

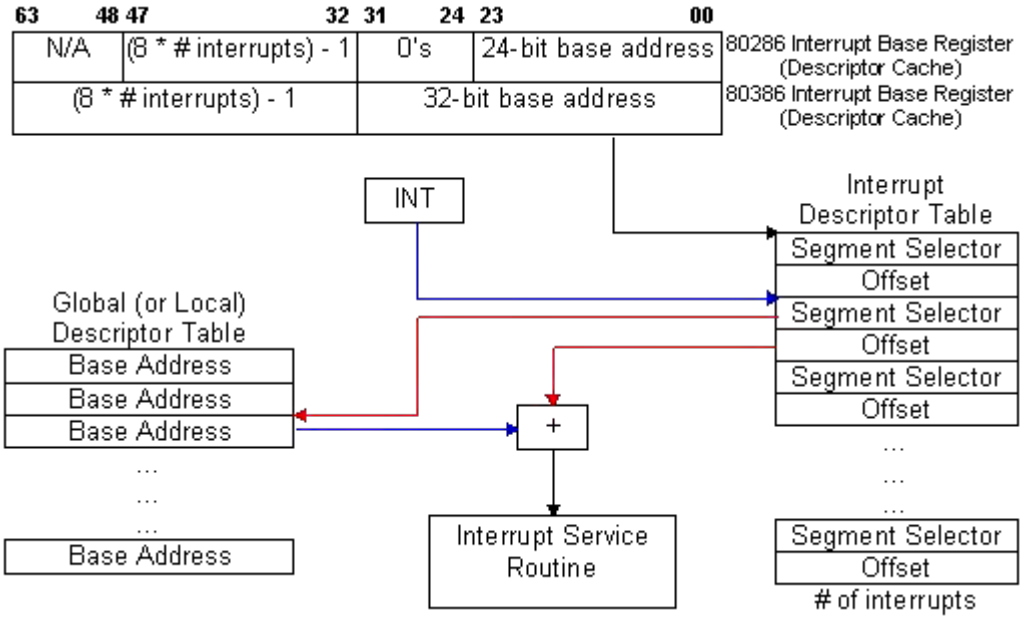
0EH	6	Floppy Disk
0FH	7	LPT1
70H	8	Real Time Clock
71H	9	General I/O
72H	10	General I/O
73H	11	General I/O
74H	12	General I/O
75H	13	Coprocessor
76H	14	Hard Disk
77H	15	General I/O

جدول1 المقاطعات الصلبة في الحاسب الشخصي

ومن الممكن أيضا حجب عمل مقاطعة و إيقاف ورودها للحاسب و ذلك من خلال برمجة الشريحة PIC و ذلك من خلال مسجل المقاطعات المحجوبة الموجود في الشريحة NMI.

3 6 ـالمقاطعات في النمط المحمي

إن عملية عنونة المقاطعات في النمط المحمي من ناحية النتيجة لا تختلف عنها في النمط الحقيقي و لكن آلية عنونة و الوصول إلى المقاطعات مختلفة، فكما رأينا في النمط الحقيقي كان لدينا جدول بأشعة المقاطعات يحتوي هذا الجدول على عناوين البرامج لـ 256 مقاطعة في الحاسب و هو جدول أشعة المقاطعات. و لكل حقل من حقول هذا الجدول كانت بحجم 32 خانة, 16 خانة للمقطع الموجود فيه برنامج خدمة المقاطعة و 16 خانة للإزاحة التي يبدأ عندها برنامج خدمة المقاطعة، أما في النمط المحمي فالبنية العامة اختلفت و أصبح لدينا عوضا عن جدول أشعة المقاطعة, أصبح جدول بواصفات المقاطعات IDT (Interrupt Descriptor Table) ويحتوي هذا الجدول على 256 مقاطعة أيضا. ويعتمد هذا الجدول في عنونة المقاطعات على آلية عنونة المقاطع في النمط المحمي أي على البنية GDT، ولننظر إلى الشكل التالي الذي يوضح آلية عمل المقاطعة:



الشكل8

فمن خلال الشكل السابق يمكننا الآن بكل بساطة استنتاج سرد آلية عمل المقاطعة في النمط المحمي:

لدى ورود المقاطعة إلى المعالج فإن المعالج يأخذ رقم المقاطعة INT.

إن كل حقل من حقول جدول واصفات المقاطعات حجمه 8Byte تماما مثل جدول واصفات المقاطع، وبالتالي لكي نصل إلى الواصف المطلوب من خلال رقم المقاطعة فعلينا ضرب رقم المقاطعة بـ 8 و من ثم يقوم المعالج بجمع الناتج بالعنوان المخزن في المؤشر الذي يشير إلى جدول المقاطعات IDTR، وبهذا نكون قد وصلنا للواصفة ضمن جدول واصفات المقاطعات.

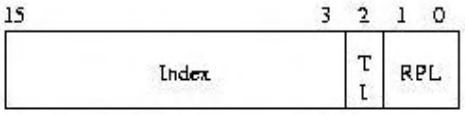
الآن من خلال الواصفة التي وصلنا إليها يكون مخزن قيمة ناخب المقطع الذي يوجد فيه برنامج خدمة المقاطعة و بالإضافة للإزاحة التي يبدأ عندها برنامج خدمة المقاطعة. و بذلك نكون قد وصلنا إلى برنامج خدمة المقاطعة، ويقوم المعالج بتنفيذه.

نجد حسب الآلية السابقة و التي أيضا تتم بشكل تلقائي من قبل المعالج نجد أن المقاطعات أصبحت عناوينها مختلفة عن عناوين مقاطعات BIOS الأساسية و أننا لم نعتمد على عناوين جدول أشعة المقاطعات التي يتم تعبئتها بشكل قياسي من قبل نظام الإقلاع الأساسي BIOS. كما أن بنية برامج خدمة المقاطعة التي يقوم بوضعها نظام BIOS لا يمكن استخدامها في النمط المحمي. و ذلك لأن برامج المقاطعة هذه مكتوبة للعمل على أساس نظام عنونة 16BIT بينما برامج المكتوبة للنمط المحمي تعمل على أساس 32Bit و بالتالي عمليات القفز البعيد و عمليات الاستدعاءات البعيدة للتوابع سوف لن نستطيع استخدامها بغض النظر عن نمط العمل و بالتالي نحن مطالبين بإعادة كتابة برامج خدمة المقاطعة، ولا نستطيع أن نتجاوز هذه الخطوة أي إعادة كتابة برامج خدمة المقاطعات و هذا ما سنراه لاحقا. و أيضا لاعتبارات أخرى يعتبر استخدام مقاطعات الـ BIOS ضرب من المستحيل لدى البرمجة في النمط المحمي.

3 7 ـ ناخبات المقاطع

إن عملية فهم المقاطع هي المدخل الرئيسي لفهم نظام عمل النمط المحمي في المعالج. فكما وجدنا مما سبق أن هنالك تشابه في المنظور العام بين المقاطع في النمط المحمي و المقاطع في النمط الحقيقي و لكن الاختلاف هو في آلية الوصول إلى هذه المقاطع.

إن مسجل المقطع الذي كان في النمط الحقيقي يستخدم للوصول إلى المقطع الذاكري أستعيض عنه بواصفة المقطع كما وجدنا سابقا. و هذه الواصفة الموجودة في جدول في الواصفات يتم انتخابها من جدول الواصفات بواسطة ناخب الواصفات Selector. و هذا الناخب هو عبارة عن مسجل من 16 خانة.



الشكل9 ناخب الواصفة Selector

- 13 خانة الأولى: تستخدم للوصول إلى واصف المقطع ضمن جدول الواصفات.
- 3 خانات الأخيرة موزعة على الشكل التالي:
- RPL (0 - 1): Requested Privilege level.
- TI (2) Table indicator.

مستوى السماحية المطلوبة Requested Privilege level RPL

ذكرنا في مقدمة الموضوع عن ميزات النمط المحمي أنه يقدم حماية للمقاطع الذاكرية كما يقدم حماية للمهمات من بعضها البعض. و يقدم النمط المحمي أربع مستويات للحماية تدعى بالـ 4Rings هذه المستويات الأربعة تحدد إمكانية الولوج إلى منطقة معنية من الذاكرة من قبل مهمات أخرى، فمثلا المنطقة التي توجد فيها نواة نظام التشغيل يجب أن يتم ضبطها على أعلى مستوى حماية و هو المستوى 0 و بينما خدمات نظام التشغيل الأساسية و برامج خدمة المقاطعات و برامج معالجة الأرتال يتم ضبطها على المستوى 1 بينما التطبيقات المستخدمين يتم ضبطها على مستوى حماية أقل.

إن عمليات ضبط مستويات الحماية تتم للمقاطع الذاكرية من خلال واصفات المقاطع حيث هنالك بعض الحقول التي سنسردها لاحقا تحدد مستوى الحماية لكل مقطع ذاكري, بينما الـ RPL التي نلاحظها هنا فهي تحدد مستوى الحماية لعملية الولوج إلى هذه المقاطع, و بكلام آخر عند محاولة مهمة الولوج إلى مقطع ذاكري فإن هذه المعالج يقارن مستوى الحماية للمقطع المطلوب و التي تكون مضبوطة في واصفة المقطع يقارنها مع مستوى الحماية لعملية الولوج التي تكون موجودة مع الخانات الثلاثة الأخيرة من ناخب المقطع Selector فإن كان مستوى الحماية لعملية الولوج أقل من مستوى الحماية للمقطع فإن عملية الوصول تعتبر غير شرعية و تظهر رسالة خطأ و إلا فإنه يتم السماح للمهمة بالولوج و يتم تحميل المسجلات بالقيم المطلوبة.

محدد الجدول Table indicator TI:

تستخدم هذه الخانة من الناخب لتحديد فيما إذا كنا جدول الواصفات الذي نريد الوصول إليه هو جدول واصفات محلي LDT أو جدول واصفات عام GDT.

جدول الواصفات المحلي LDT: TI = 1
جدول الوصفات العام IDT: TI = 0

3 8 -واصفات المقاطع

لقد تكلمنا حتى الآن كثيرا على واصفات المقاطع, و كل ما عرفناه عن هذه الواصفات لحد الآن هو الأمور التالي:

الواصف هو مسجل يحتوي على عنوان و حجم المقطع الذاكري. و حجم هذا الواصف 8Byte.
الواصف عبارة عن مسجل يحتوي على قيم لضبط مواصفات المقطع و مستوى حماية المقطع وسماحيات الولوج إليه.
الواصف هو حقل من جدول كبير يدعى بجدول الواصفات و يتم تخزين الواصف الحالي في مسجلات مخفية لا يمكن الوصول إليها من قبل المبرمج.
هنالك نوعان من الواصفات, واصفات للمقاطع واصفات للمقاطعات و تسمى بالبوابات Gates.
الآن لندخل في تفاصيل الواصفات. و لنفصل بينة الواصف وحقوله و لنبدأ بالجدول التالي حيث يبين الجدول التالي واصفات المقاطع في معالجات 80386.

جدول2 جدول واصفات المقطع

البايت 0	البايت 1	البايت 2	البايت 3	البايت 4	البايت 5	البايت 6	البايت 7
حجم المقطع 7-0	حجم المقطع 15-8	عنوان بداية المقطع 7-0	عنوان بداية المقطع 15-8	عنوان بداية المقطع 23-16	السماحيات	• مؤشرات • تتمة حجم المقطع 19-16	عنوان بداية المقطع 31-24

حجم المقطع 0-19: و هي تحدد حجم المقطع و هذا ما يمنح النمط المحمي ميزة المقاطع ذات الحجم المتغير.

عنوان بداية المقطع 0-31: و هي تحدد عنوان بداية المقطع الذاكري. ونلاحظ الحجم الكبير للعنونة و هذا ما يمنح النمط المحمي إمكانية عنونة 2^{32} =GB4.

السماحيات: نوضحها في الجدول التالي:

جدول3 جدول سماحيات الواصف

البت 0	البت 1	البت 2	البت 3	البت 4	البت 5 - 6	البت 6
إمكانية	للقراءة \	إمكانية تمدد	تنفيذي	1	السماحيات	إتاحية المقطع

الولوج	للكتابة	المقطع				

إمكانية الولوج: يتم تفعيل قيمة هذه الخانة في حال الكتابة أو القراءة من المقطع.

قراءة \ كتابة: لتحديد فيما إذا المقطع هو مقطع فقط للقراءة أو مقطع للقراءة و الكتابة معا

إمكانية تمدد المقطع: يمكن أن تكون في هذه الحالة الإزاحة قيمتها أكبر من قيمة حجم المقطع و يمكن في هذه الحالة أيضا أن يتم تمدد المقطع نحو الأسفل.

تنفيذي: لتحديد فيما إذا هذا المقطع مقطع شفرة تنفيذية =1, أو مقطع معطيات\مكدس = 0.

الحماية: لتحديد مستوى حماية المقطع و إمكانية الولوج إليه.

اتاحية المقطع: عندما تكون قيمة الحقل =1 فهذا يمنع الوصول إلى المقطع.

المؤشرات: نوضحها في الجدول التالي:

جدول4 جدول مؤشرات الواصفة

البت 4	البت 5	البت 6	البت 7
0	0	الحجم الافتراضي	وحدة قياس حجم الذاكرة

وحدة قياس الذاكرة: عندما يكون 1 فإن وحدة قياس الذاكرة تصبح 4KB و بالتالي يصبح المعالج قادر على وضع حجم المقطع الذاكر بحجم GB4
في حال كانت الخانة 0 يكون وحدة قياس الذاكرة بـ 1KB و بالتالي يكون الحجم الأعظمي للمقطع الذاكر KB64.

الحجم الافتراضي: تتحكم هذه الخانة فيما إذا كان المعالج سيعمل على أساس 16خانة أو على أساس 32خانة.

البت 4 من بايت السماحيات: في حال كانت قيمة الخانة 1 فهذا يعني أننا نستخدم مقطع ذاكري عادي إما مقطع معطيات \مكدس أو مقطع شفرة.
في حال كانت قيمة الخانة 0 فهذا يعني أننا نستخدم مقطع من نوع خاص أو ما يسمى مقطع نظام
أنواع مقاطع النظام:

مقطع حالة المهمة TSS: Task State Segment يستخدم هذا المسجل لدعم تقنية تعدد المهام, و معالجات 80386 لها 4 أنواع من مسجلات TSS

البوابات Gates:
تستخدم في المقاطعات وكما أنها تستخدم بشكل أدق في العمليات التي تتطلب الانتقال بين

مستويات الحماية المختلفة للقطع الذاكرية. و واصفات البوابات لها حقول مختلفة عن حقول واصفات المقاطع.

3 9 -واصفات المقاطعات -البوابات-

بالنسبة لواصفات المقاطعات لنرى الجدول التالي و من ثم نفصل في حقول الواصف:

جدول5 واصفات البوابات

البايت 7	البايت 6	البايت 5	البايت 4	البايت 3	البايت 2	البايت 1	البايت 0
الإزاحة	الإزاحة	السماحيات	كلمة العدد	الناخب	الناخب	الإزاحة	الإزاحة
31-23	23-16		4-0	15-8	7-0	15-8	7-0

السماحيات:

جدول6 سماحيات جدول المقاطعة

البت 7	البت 6-5	البت 4	البت 3-2-1-0
النوع	السماحيات	0	اتاحية المقطع

النوع: تحدد أنواع واصفات النظام و الجدول التالي يوضح هذه الأنواع وقيمة الحقل وفقا لهذه الأنواع

Type	Segment function	Type	Segment function
0	(invalid)	8	(invalid)
1	Available '286 TSS	9	Available '386 TSS
2	LDT	10	(undefined, reserved)
3	Busy '286 TSS	11	Busy '386 TSS
4	'286 Call Gate	12	'386 Call Gate
5	Task Gate	13	(undefined, reserved)
6	'286 Interrupt Gate	14	'386 Interrupt Gate
7	'286 Trap Gate	15	'386 Trap Gate

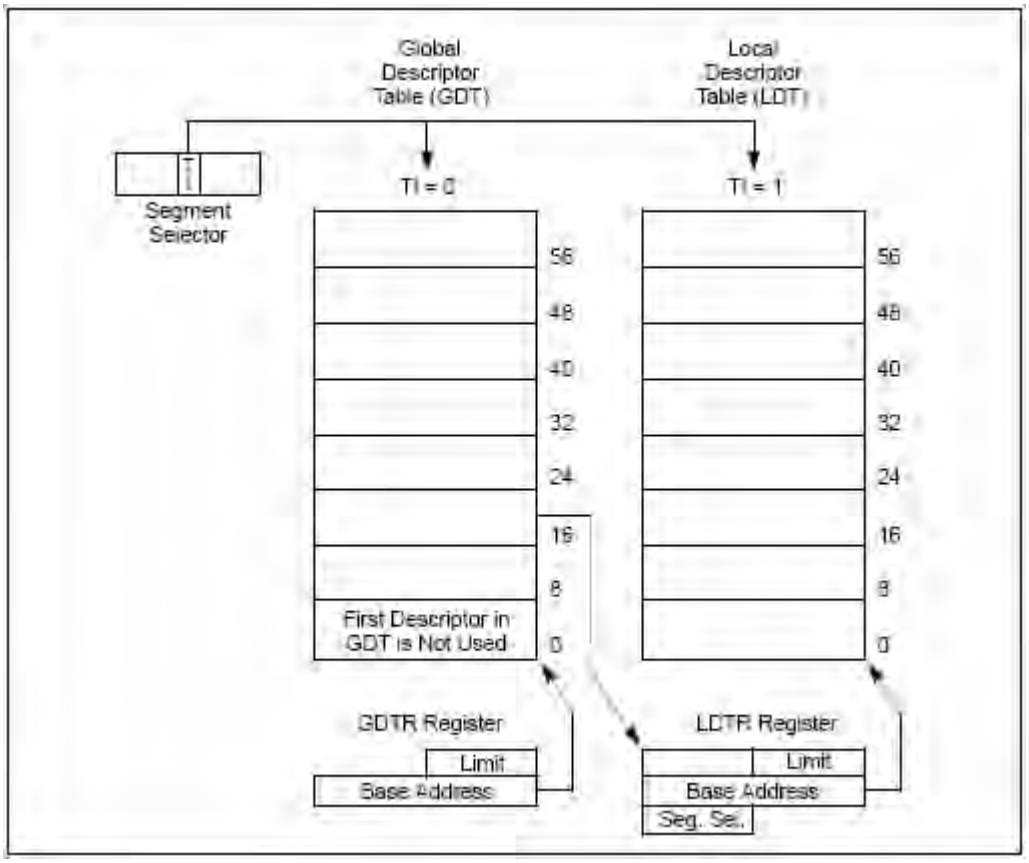
الشكل10

3 10 -جداول الواصفات

أيضا تكلمنا كثيرا عن جداول الوصفات و مما عرفناه لحد الآن عن جداول الواصفات ما يلي:

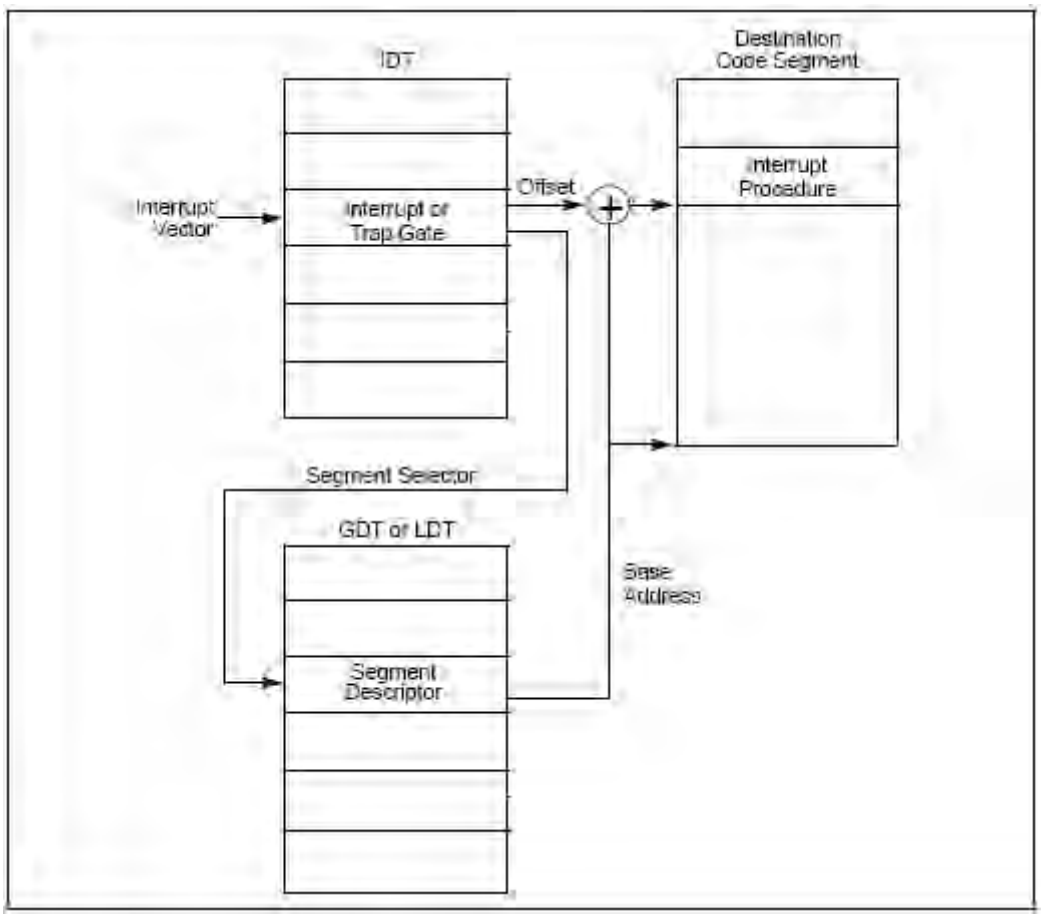
جدول الواصفات هو عبارة عن جدول حقوله عبارة عن واصفات لمقاطع ذاكرية كل حقل بحجم 8Byte

يتم الوصول إلى هذه الجداول من خلال مسجل جدول الواصفات.
حجم الجدول 8192Byte. و يتم تهيئة محتويات هذه الجداول يدويا من خلال المبرمج لدى بدء عملية إقلاع النظام.
و أيضا مما وجدناه أنه لدي ثلاث أنواع من الجداول:
جدول الواصفات العام. (Global Descriptor Table) GDT و يتم عنونة هذا الجدول من خلال مسجل الواصفات العام GDTR. و الذي يهيئ بقيمة عنوان بداية جدول الواصفات العام.
جدول الواصفات المحلي.(Local Descriptor Table) LDT و يتم عنونة هذا الجدول من خلال مسجل الواصفات المحلي LDTR و الذي يهيئ بقيمة عنوان بداية جدول واصفات المحلي للمهمة.



الشكل11

جدول واصفات المقاطعات.(Interrupt Descriptor Table) IDT و يتم عنونة هذا الجدول أيضا من خلال مسجل الـ IDTR.



الشكل12

3 10 1 -الحماية في النمط المحمي

كما نلاحظ إن تسمية النمط المحمي بهذا الاسم تعود لمستويات الحماية التي يؤمنها معالج 80386 لمقاطع الذاكرة. و إن كل مقطع ذاكري أو لنقل بشكل أعم برنامج له مستوى حماية يتراوح من الصفر و حتى الثلاثة كما وجدنا سابقا. و مر معنا أن المستوى الحماية 3 هو أخفض مستوى حيث تمنع البرامج التي تم ضبطها عند هذا المستوى من تنفيذ بعض العمليات الحساسة و التي من المحتمل أنها تؤثر على عمل النظام. كما أن البرامج التي تكون مضبوطة عند مستوى الحماية المنخفض لا تستطيع أن تصل إلى المعطيات للمقاطع التي تكون مستويات حمايتها أعلى. و هذا ما مر معنا سابقا.

إن كل واصف مقطع لديه مستوى من السماحيات يدعى بالـ Descriptor Privilege Level أو اختصارا بالـ DPL يحدد من خلاله مستوى سماحيات المقطع الذي يشير إليه. و أيضا 386 يحدد أي من البرامج يحق لها أن تقوم بعمليات الإدخال أو الإخراج.إن التعليمات المتاحة للبرامج التي تعمل على المستوى 0 من الحماية أو السماحية PL0 يحق لها التالي من التعليمات:

- HLT
- CLTS
- LGDT

•LIDT

•LLDT

•LTR

•LMSW

•MOV (to/from control/debug/test registers)

On 486 systems

•INVD

•WBINVD

•INVLPG

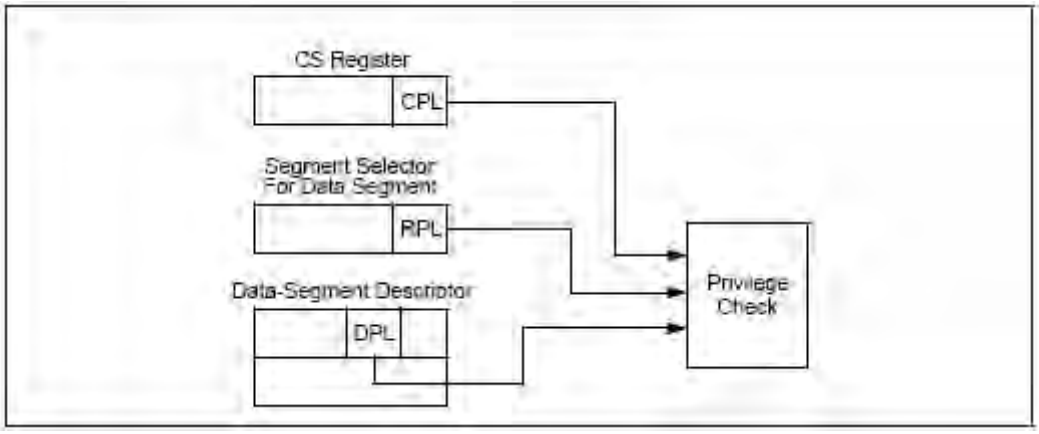
On Pentium and above

•RDMSR

•WRMSR

•RDTSC

إن مستوى سماحية البرنامج أو المقطع الذاكري هي مساوية لقيمة الـ RPL في مسجل CS الذي يحوي على ناخب المقطع. و هذه الخانة تدعى بالـ CPL أي مستوى السماحية الحالية Current Privilege Level. الوصول:نك تعديل مسجل CS مباشرة و بالتالي لا يمكن تعديل قيمة RPL.



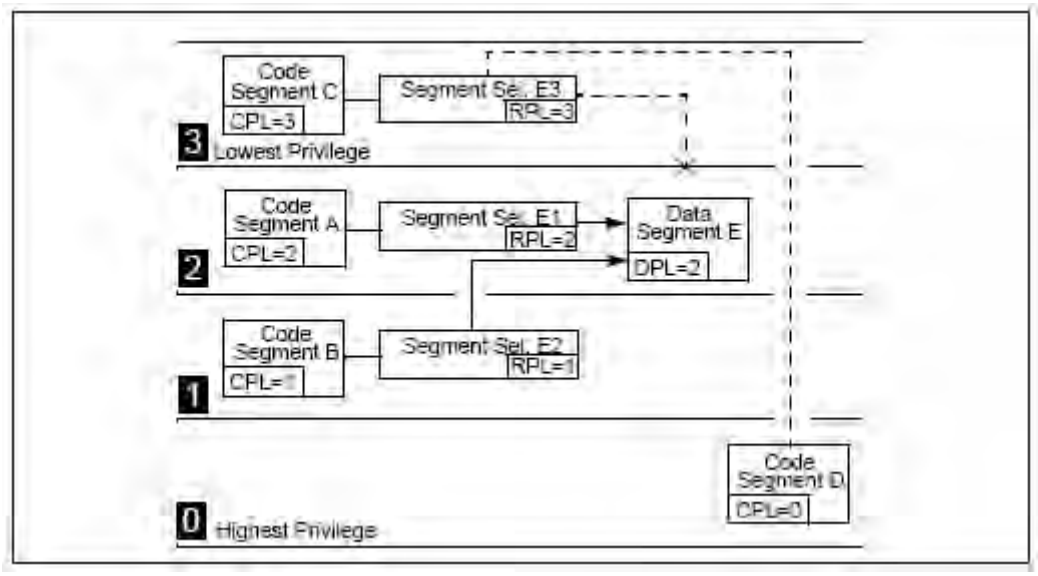
الشكل13

3 .11 -الوصول

إن عملية تحميل مسجلات لا تتم بشكل اعتباطي وأي ناخب بقوم بانتخاب واصف مقطع معين من الجدول فإنه لا يقوم مباشرة بتحميل المسجلات بقيم الواصف. وإنما قبل ذلك هنالك عملية فحص و تأكد من أحقية البرنامج الحالي في تحميل الواصف المنتخب. ويتم ذلك من خلال مقارنة CPL مع DPL (حيث أن سماحيات الواصف الحالي تدعى Descriptor Privilege Level DPL).

في حال كان DPL>CPL فإن عملية التحميل تفشل و ترسل مقاطعة أو رسالة خطأ للمعالج.

في حال كان DPL<CPL فإن عملية تحميل الواصف في المسجل تتم بنجاح و يتم الولوج للقطعة.



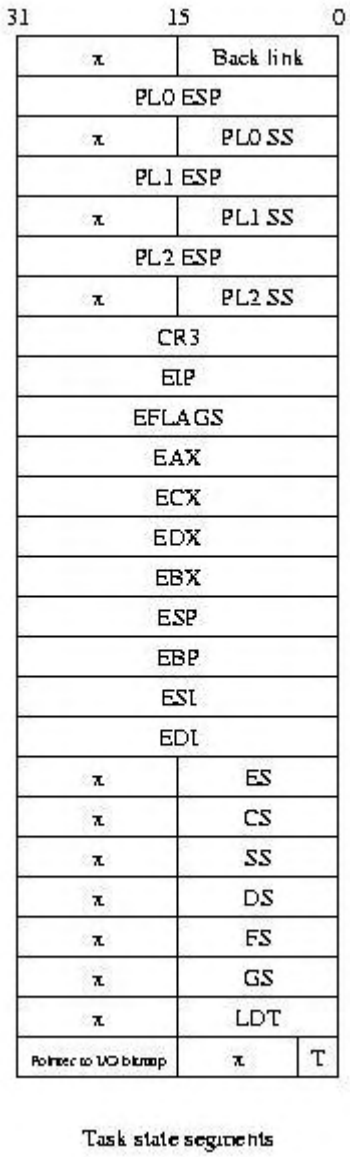
الشكل 14

3-11-1 -آلية عمل تقنية تعدد المهام

كما ذكرنا من البداية أن من الميزات التي يقدمها النمط المحمي هي دعم لتقنية تعدد المهام و تنفيذ أكثر من مهمة واحد على التفرع. و ذلك من خلال تقسيم زمن عمل المعالج بين هذه المهام.

يستخدم معالج 80386 مقطع حالة المهمة TSS في دعم تقنية تعدد المهام, حيث يشير واصف TSS إلى ذاكرة تخزين مؤقتة بحجم Byte104, وبالإضافة لتعدد المهام يستخدم الـ TSS في معالجة المقاطعات الصلبة. إن ناخب TSS غير قابل للقراءة أو الكتابة. و بشكل عام يتم خلق جدول TSS والذي هو عبارة عن مؤشر لذاكرة TSS المؤقتة. ويكون عادة ناخب مقطع TSS موجود في جدول الواصفات العام GDT و لا علاقة له بجدول IDT أو LDT. و لكن كما سبق وذكرنا عندما يستخدم TSS في معالجة المقاطعات الصلبة فإن بوابة المهمة Task Gateتكون موجود في جدول IDT. و بعد كل ذلك أقول أن المعالج يقوم باستخدام ناخب TSS داخليا.

عندما نحتاج لعملية تبديل المهمات فإن المعالج يقوم بحفظ حالة المعالج في مقطع TSS و حالة المعالج في الغالب تكون هي عبارة عن محتويات مسجلات المعالج.و عندما تحديث عملية تبديل المهمة فإن عملية حفظ حالة المعالج في TSS تتم بشكل تلقائي, و كما أن هذه العملية تتم بسرعة و بأقل عدد من دورات عمل المعالج. و قبل أي أن تتم تهيئة وبدء أي مهمة في العمل فإن من الواجب على نظام التشغيل أن يقوم بملء حقول معينة من ذاكرة TSS.



15 الشكل

3 11 2 -الاستثناءات في معالجات 80386

يولد معالج 80386 ثلاث أصناف من المقاطعات:

الأفخاخ Tarps
الأخطاء Faults
الإخفاقات Aborts

إن ما يميز كل نوع من هذه الأنواع الثلاث للمقاطعات هو محتوى المكدس بعد كل من هذه الأحداث، حيث أن الفخاخ لا تقوم أبدا بدفع شفرة الخطأ في المكدس, بينما الأخطاء تقوم على الأغلب بدفع رقم الخطاء إلى المكدس أما الإخفاق فتقوم بتمرير رقم الخطاء إلى المكدس دائما. و إن الفخاخ تستخدم في العادة بنفس طريقة استخدام المقاطعات المرنة في النمط الحقيقي و عادة ما يتم قدح الفخاخ لدى حدوث المقاطعات البرمجية و كما نعلم المقاطعات البرمجية تحدث لدى

استدعاء التابع INT، وعند حدوث الفخ في البرنامج فإن قيمة المسجل CS:EIP يتم شحنها بالتعليمة التالية للتعليمة التي أحدثت الفخ.و بالنتيجة النظام لا يستطيع أن يستعيد عمله بعد حدوث الفخ.

تقع الأخطاء في الغالب لدى حدوث عمليات خاطئة. ولدى حدوث الخطأ, فإن مسجل CS:EIP يتم شحنه بالتعليمة التي سببت الخطأ, والأخطاء هي بالنتيجة عبارة عن مجموعة من الأحداث الخطيرة و أكثر هذه الأخطاء شهرة هي خطأ GPF(General Protection Error)، بينما الإخفاقات تعبر عن أحداث خطأ في البرنامج و لكن برامج خدمة هذه النوع من المقاطعات لا يمكنها أصلاح هذا النوع من المشاكل و مثال على هذه الإخفاقات هو خطأ Double Fault.يوضح الجدول التالي أصناف المقاطعات الثلاث التي يولدها المعالج و أرقامها و تصنيفاتها.

جدول7 المقاطعات و الاستثناءات

Description	nt #	ype	Return Addr points to faulting instruction	rror Code	This interrupt first appeared in this CPU
Division by 0			Yes		8086
Debug Exception		ault	*1	o	8086
NMI	1	*1	No	No	8086
Breakpoint	2	*2	No	No	8086
Overflow	3	Trap	Yes	No	8086
Bounds	4	Trap	Yes	No	80186
Invalid OP Code	5	Fault	Yes	No	80186
Device not available	6	Fault	Yes	No	80186
Double Fault	7	Fault	No	No	80286
Copr. segment overrun	8	Abort	Yes	Yes	80286 *3
Invalid TSS	9	Fault	Yes	No	80286
Segment not present	10	Fault	Yes	Yes	80286
Stack fault	11	Fault	Yes	Yes	80286
General Protection	12	Fault	Yes	Yes	80286
Page fault	13	Fault	Yes	Yes	80386
Floating point error	14	Fault	Yes	Yes	80386
Alignment check	16	Fault	Yes	No	80486
Machine check	17	Fault	No	Yes	Pentium *4
Software interrupts	18	Abort	No	Yes	All
	0-255	Trap		No	
*1	On the 386-class CPUs, debug exception can be either traps, or faults. A trap is caused by the Trap Flag (TF) being set | in the flags image, or using the debug registers to generate data				

	breakpoints. In this case the return address is the instruction following the trap. Faults are generated by setting the debug registers for code execution breakpoints. As with all faults, the return address points to the faulting instruction.
*2	Non-maskable.
*3	Removed from the 80486, now generates exception 13 on all future processors.
*4	Model dependant. Behavior may be different or missing on future processors.

3 11 3 -وحدة إدارة الذاكرة

يوجد نوعين لإدارة الذاكرة:

التقطيع
التصفيح

من بين هذين النمطين فإن استخدام تقنية التقطيع هي الأكثر شيوعا و استخدام و قد تناولنا هذه الآلية في عنونة الذاكرة في فقرة سابقة.تقنية التصفيح هي تقنية اختيارية لمستخدم نظام التشغيل. كما أن ميزة استخدام آلية التصفيح تفيد في تقييد الوصول إلى أجزاء معينة من الذاكرة بينما في المقابل فإن من الصعوبة بمكان القيام بذلك عند استخدام آلية التقطيع. ويمكنك تفعيل العمل بآلية التصفيح من خلال مسجل CR0.

تستخدم المقاطع لتخزين البرامج التي هي قيد التنفيذ حاليا حيث يتم تخزين هذه البرامج و توزيعها على أصناف المقاطع الثلاثة, مقطع الشفرة ومقطع المعطيات ومقطع التكديس. وكما يستطيع نظام التشغيل أيضا باستخدام ناخب مقطعي وحيد أن يصل لـ 4GB من الذاكرة وذلك كما وجدنا سابقا لدى دراسة بنية الواصف.إن استخدام تقنية التصفيح كما سنرى لدى دراسة موضوع إدارة الذاكرة, لا يقتصر فقط على تفعيل الخانة 31 من المسجل CR0 في المعالج, و إنما هذا يتطلب منك أن تقوم ببناء:

فهرس أدلة الصفحات Page Directory Entries (PDE)
جدول أدلة الصفحات Page Directory Table (PTE)

إن PDE وPTE هي عبارة عن جداول تستخدم لتحويل العناوين وترجمة العناوين المنطقية إلى عناوين فيزيائية، وكما وجدنا سابقا في آلية الحماية في النمط المحمي, فإن فقط البرامج ذات مستوى الحماية PL0 تستطيع الوصول إلى وحدة إدارة الذاكرة MMU وتفعيلها, بينما برامج التي لها مستوى الحماية PL3 والتي تكون في العادة برامج المستخدم فإنها لا تستطيع و لا تعلم بوجود وحدة إدارة الذاكرة أصلا، و تصنف صفحات الذاكرة عادة كصفحات نظام أو كصفحات مستخدم User/System، حيث البرامج ذات مستوى الحماية PL0 يمكنها أن تصل إلى صفحات النظام بينما برامج ذات المستوى PL3 فيمكنها الوصول إلى صفحات المستخدم و لا تستطيع الوصول إلى صفحات النظام.

إن معظم الأنظمة لا تتوافر فيها في الحالات العادية ذاكرة رئيسية بحجم 4GB ولذلك لدى بناء دليل الصفحات فإنه لا نقوم بتعيين إلا الصفحات المتوفرة في الذاكرة بينما الصفحات غير المتوفرة في الذاكرة نقوم بالإشارة إليها على أنها Not Present.علمنا لحد الآن أنه لدينا PDE و PTE حيث يحتوي كل حقل من حقول PDE مؤشر إلى بداية جدول دليل صفحات PTE و كل PTE يشير إلى عنوان بداية صفحة، الآن كيف يتم تحديد PDE ؟ يتم تحديد PDE من خلال مسجل التحكم CR3.

3 11 4 -نمط العمل الوهمي VM8086

كما نوهنا من البداية إلى أن معالج 80386 يعمل في ثلاث أنماط عمل مختلفة ومن بينها النمط الافتراضي 8086. V86. في هذا النمط V86 فإن المعالج يعمل تماما كما يعمل معالج 8086 مع بعض الاختلافات البسيطة. و تكون وحدة إدارة الذاكرة MMU فعالة في النمط V86. و تسمى المهمة التي تعمل في النمط V86 بمهمة V86.وذلك لدى تفعيل خانة VM في مسجل EEFLAGES. و من ميزات استخدام المهمة V86 عن المهمة في النمط الحقيقي بالإضافة إلى خصائص الأمن فإن عملية التبديل من النمط المحمي إلى النمط الحقيقي تأخذ زمن اكبر بكثير من عملية الانتقال من النمط المحمي إلى النمط V86. كما أن تنفيذ البرامج في النمط V86 أسرع بكثير مما هو عليه في النمط الحقيقي.

و كما هو في النمط الحقيقي فإن المعالج لدى عمله في النمط V86 إنه يقوم على عنونة الذاكرة باستخدام المقاطع و الإزاحة عوضا عن استخدام الناخب و الإزاحة وكما أن المعالج في النمط V86 يستطيع عنونة ذاكرة 1MB فقط.و كما أن مستوى سماحية البرامج التي تعمل في النمط V86 تعمل بمستوى سماحية PL3. كما أن البرامج في هذا النمط يمكنها القيام بعمليات الإدخال و الإخراج فقط في حالة كانت سماحية IOPL3 أيضا. و في حال كانت السماحية IOPL أقل من 3 فإن المعالج يولد رسالة الخطأ General Protection Fault في حال قيام البرنامج بتنفيذ التعليمات التالية:

.CLI, STI, INT, IRET, LOCK, PUSHF and POPF.

3 11 5 -معالجة المقاطعات في النمط المحمي

إن عملية تخديم المقاطعات هي من أهم العمليات التي يجب أن يؤمنها نظام التشغيل, و يقصد بعملية تخديم المقاطعات هي البرنامج الذي ينفذ بعد وصول المقاطعة إلى المعالج أي هي استجابة النظام للمقاطعة, و هذه الاستجابة تكون على برامج كما قلنا و تدعى هذه البرامج ببرامج خدمة المقاطعة Interrupt Handler. و في هذه الفقرة سوف نحاول أن نلقي نظرة على كيفية تخديم المقاطعات في النمط المحمي و النمط V86.

إن عملية تخديم المقاطعات في النمط V86 هي عملية معقدة جدا. فعلى الرغم من أن المعالج يعمل في النمط V86 فإن معالجة المقاطعات مازالت تتم في النمط المحمي. و يستفاد من النمط V86 أيضا في استخدام مقاطعات BIOS التي لا نستطيع استخدامها في النمط المحمي كما أنه يمكننا في النمط V86 يمكننا أن نستخدم التعليمات CLI و STI.

3 11 6 ـخطوات العمل في النمط المحمي

يتطلب العمل في النمط المحمي منك بعض الخطوات التهيئة الأساسية. و التي يتم تنفيذها في النمط الحقيقي, و يتم عادة تهيئة النمط المحمي لدى عملية إقلاع نظام التشغيل. و خطوات تهيئة النمط المحمي تنفذ عادة بتعليمات المجمع و هذه الخطوات هي على الشكل التالي:

تهيئة جدول الواصفات العام GDT و يتم تهيئة الواصف الأول في الجدول بالقيمة Null.
تهيئة جدول واصفات المقاطعات IDT (اختيارية في النمط المحمي) و يتم تهيئة الواصف الأول في الجدول بالقيمة Null.
إيقاف عمل المقاطعات.
إعادة برمجة شريحة التحكم بالمقاطعات لإعادة توزيع المقاطعات الصلبة.
تحميل المسجل GDTR ببداية جدول GDT.
تحميل مسجل IDTR ببداية جدول IDT.(اختيارية للنمط المحمي)
وضع 0 في الخانة الأولى PE من المسجل CR0 لنقل المعالج إلى النمط المحمي.

تنفيذ تعليمة قفز طويل لتحميل كل من مسجلي CS و EIP بقيمته البدائية و التي تكون عادة ثاني واصف في جدول الواصفات العام و هي تشكل الناخب الحالي و أيضا يستفاد من هذه الخطوة أي القفز الطويل في تفريغ رتل جلب التعليمات في المعالج من تعليمات النمط الحقيقي التي تكون على أساس 16خانة.

الآن صرنا في بيئة النمط المحمي. و أول ما نقوم به هو تحميل مسجلات DS وSS بناخبات مقطع المعطيات و مقطع المكدس و على الترتيب.

تنصيب و تهيئة مكدس النمط المحمي.
تفعيل المقاطعات.(اختيارية في النمط المحمي.)

3 12 -قسم التنفيذ

3 12 1 -تهيئة النمط المحمي

الآن سوف نقوم بتطبيق ما درسناه نظريا في السابق عن النمط المحمي و سنقوم بتطبيقه على مشروع نظام التشغيل لكي نقوم بإدخال النظام في نمط العمل المحمي Protected Mode.تبدأ تهيئة النمط المحمي عند بدء إقلاع نظام التشغيل حيث يتم إعداد متطلبات النمط المحمي عندما يكون المعالج يعمل في النمط الحقيقي و يتم تهيئة كل مستلزمات الانتقال إلى النمط المحمي و من ثم يتم الانتقال إلى النمط المحمي.كما وجدنا سابقا فهنالك مجموعة من الخطوات للدخول في النمط المحمي. هذه الخطوات يتم تطبيقها ضمن مكتبة start.s و التي يتم تنفيذها مباشرة بعد انتهاء نظام الإقلاع Grub وتسليم قيادة الحاسب إلى نظام التشغيل.

الآن ضمن مكتبة start.s يتم تنفيذ ما سبق ذكره من خطوات على التتالي و ذلك باستخدام تعليمات لغة التجميع طبعا.

و الخطوتين الأولى و الثانية التي هي عبارة عن تهيئة لجداول GDT و IDT يتم تنفيذه على الشكل التالي:

```
gdt:
//;Dummy descriptor 0x00.
        .word 0                         //;Limit 15:0
        .word 0                         //;Base 15:0
        .byte 0                                 //;Base 23:16
        .byte 0                                 //;Access byte (descriptor type(
        .byte 0                                 //;Limits 19:16, Flags
        .byte 0                                 //;Base 31:24

//;Data descriptor 0x08.
        .word 0xFFFF                    //;Limit 15:0
        .word 0                         //;Base 15:0
        .byte 0                                 //;Base 23:16
        .byte 0x92                      ; // Data, Present, Writeable (1,0,0,1,0010(
        .byte 0xCF                      //;G=1, D=1, 0, AVL=0, 1111=F:
                                Limit/Length (1,1,0,0,1111)
byte 0                                  //;Base 31:24

//;Code descriptor 0x10.
        .word 0xFFFF                    //;Limit 15:0
        .word 0                         //;Base 15:0
        .byte 0                                 //;Base 23:16
        .byte 0x9A                      //;Code,        Present,        Non-conforming,
Exec/read                                       (1,0,0,1,1110)
.byte 0xCF                      //;G=1,       D=1,        0,      AVL=0,      1111=F:
                                Limit/Length (1,1,0,0,1111(
        .byte 0                                 //;Base 31:24
gdt_end:
```

نجد من الشفرة السابقة المكتوبة بلغة NASM أنه قمنا بتهيئة جدول GDT بثلاث واصفات رئيسية هي:

واصفة الابتدائية و التي تكون قيمها Null
واصفة مقطع المعطيات
واصفة مقطع الشفرة

و كل واصف من هذه الواصفات قمنا بتهيئته بالقيم المناسب حسب الطريقة التي نرغب بها. مثلا واصفة مقطع المعطيات و كما وجدنا سابقا من شرح لحقول الواصفات نجد أننا قمنا بتهيئته بالقيم التالية:

عنوان بداية المقطع يبدأ من العنوان 0x0000.0000H
حجم المقطع الذاكري 0xFFFFFH أي الحجم 1MB*4KB = 4GB.
الأعلام موضحة في التعليق الذي بجانب الشفرة هي و السماحيات.

الخطوة الثالثة والرابعة كانت عبارة عن إيقاف عمل المقاطعات ومن ثم إعادة برمجة الشريحة لكي يتم توزيع العناوين الجديدة على المقاطعات.

عملية إيقاف المقاطعات يتم تنفيذها من خلال التابع الموجود في مكتبة Interrupt الموجودة في /os/arch/i386/kernel.

```
void disable_IRQ( uint8_t IRQ)
```

عملية إعادة برمجة شريحة التحكم بالمقاطعات يتم تنفيذها من خلال التابع الموجود في مكتبة Interrupt الموجودة في /os/arch/i386/kernel.

```
Void reprogram_PIC()
```

الخطوة الرابعة و الخامسة كانت عملية تحميل مسجلات GDTR و IDTR بقيمها الفعلية.

```
gdtr:
        .word (gdt_end - gdt - 1)          ; // GDT limit.
        .long (gdt - ADDRADJUST)           ; // GDT linear address.
```

حيث نقوم في هذا الجزء بتهيئة محتويات مسجل الـ GDTR و التي تتألف من حجم جدول GDT و عنوان بداية الجدول.

```
        // ;Reload global descriptor table (GDT)
        lgdtl   (gdtr)
```

و نقوم بتعليمة LGDT بتحميل مسجل GDTنلاحظ أننا إلى الآن لم نقم بتهيئة جداول IDT هذه سنأتي عليها بطريقة أخرى.

الآن نقوم بتحويل عمل المعالج إلى النمط المحمي من خلال مسجل الـ CRO بتنفيذ التعليمات التالية في ملف start.s

```
        ; // Enable paging.
        movl   %cr0, %eax
        orl    $0x80010000, %eax ; // cr0.PG = cr0.WP = 1
        movl   %eax, %cr0
```

و الخطوة الأخيرة هي تنفيذ عملية القفز و تتم عملية القفز الطويلة من موقع التعليمة الحالية إلى موقع تابع k_main_ في نظام التشغيل و الذي يمثل تابع mainفي برنامج نظام التشغيل.

```
// ;Jump to the kernel main routine.
jmp    k_main
```

ملاحظة: يتم التعرف على k_main_ من خلال التعليمة التالية في بداية الملف start.s.

```
.extern k_main
```

3 12 2 -تنصيب مقاطع النظام

حتى الآن و ما قمنا به سابقا كان عملية تهيئة لابد منها للانتقال للعمل في النمط المحمي, أما الآن فسنقوم بتنفيذ نفس العمليات السابقة و لكن من ضمن نظام التشغيل أيضا لكي يكون هنالك إمكانية في التحكم من ضمن نظام التشغيل بجدول الواصفات العام. و بالتالي نقوم الآن بتنفيذ:

عملية تنصيب المقاطع في جدول الواصفات العام بأنواعها و مستويات سماحياتها.
عملية إزالة مقاطع من جدول الواصفات العام.

و للقيام بتنفيذ ما سبق فإنه يلزمنا أن نعرف بينة معطيات تشير إلى جداول الواصفات و بنية معطيات للواصفات و هذا يتم في المكتبة mem.c. حيث يتم تنصيب مقاطع النظام و مقاطع المستخدم بأنواعها المعطيات و الشفرة. و يتم ذلك من خلال التابع:

```
void __INIT__ install_GDT()
```

كما نقوم بإزالة المقاطع من جدول الواصفات من خلال التابع

```
void remove_GDT_entry(word sel)
```

و الذي نمرر له دليل الواصف أو ناخب الواصف الذي نرغب بحذفه من جدول الواصفات و يتم حذف الواصف بتصفير قيم الواصف ضمن الجدول فقط. أي نجعله null.

3 12 3 -إعداد المقاطعات في النمط المحمي

من خلال ما سبق لم نقم بتهيئة جدول واصفات المقاطعات و قد تم تنفيذ هذه الخطوة خارج مكتبة start.s و ذلك لتأمين ديناميكية أكبر في عملية تنصيب المقاطعات و تغيير برامج خدمة المقاطعة بشكل ديناميكي.يتم تنصيب المقاطعات في الجدول بعد أن يستلم الإجراء k_main_ التنفيذ و يستدعي الإجراء install_IDT(). و يتم تنفيذ جميع عمليات تنصيب واصفات المقاطعات في مكتبة interrupt.c.

و تنصيب الواصفات يتم من خلال التابع التالي:

```
void setup_IDT_entry(byte i, word selector, dword offset, byte attribs,byte
paramcnt )
```

فكما نلاحظ من مدخلات هذا التابع فإنه يتم تمرير قيمة الناخب و رقم الواصف و الإزاحة للمقاطعة و خصائص الواصف:

و نقوم بتنصيب أربع أنواع من واصفات المقاطعات أو البوابات و هي:

بوابة المقاطعات.

```
void set_intr_gate( unsigned int n, void *addr)
```

بوابة المهمات.

```
void set_task_gate( unsigned int n, unsigned int gdt_entry)
```

بوابة الفخ.

```
void set_user_gate( unsigned int n, void *addr)
```

بوابة المستخدم.و التي هي عبارة عن بوابة الفخ و لكن مخصصة لعمليات الولوج للمستخدم. و يتم تعيين مستوى السماحية لها على DPL=3.

```
void set_user_gate( unsigned int n, void *addr)
```

3 12 4 -إسناد برامج خدمة المقاطعة

برامج خدمة المقاطعة تكون عناوينها موجودة في مصفوفة سجلات, حيث يتألف كل سجل من حقل وحيد فيها مؤشر إلى بداية برنامج خدمة المقاطعة handler.

```
irq_handler[].handler = handler
```

3 13 -المراجع

- **http://www.osdever.net** **2004** مجموعة من المقالات عن النمط المحمي

· All 17 Pmode Chapters by Alexei A Frounze
· Xosdev chapter 1 Chapter1 of Series by mr xsism
· Xosdev chapter 2 Chapter 2 of Series by mr. xsism []
· Protected Mode by Chris Giese []
· Working in the Protected Mode Enviroment by rashant · The World of Protected Mode by Gregor Brunmar []

Working with protected mode2000, 2001 Prashant TR
http://members.tripod.com/protected_mode/prashant/protmode.htm

http://x86.ddj.com/articles/pmbasics/tspec_a1_doc.htm
Protected Mode By Chres Gies.
Protected Mode Basics by Robert Collins

كتاب برمجة الحواسيب الشخصية بلغة التجميع 1996
الفصل التاسع : مقدمة إلى معالجة لوحة المفاتيح و الشاشة.
الفصل الحادي عشر : المعالجة المتقدمة للوحة المفاتيح.

الباب □. نواة النظام

4 -إدارة الذاكرة Memory Management

مُقَدِّمَة

تعتبر الذاكرة من أهم الموارد وأغلاها في نظام الحاسب، وقد كانت في الأيام الماضية قليلة جدا (لا تتجاوز الـ 1MB)، ومع اتساع هذه الذاكرة فإن المبرمج أصبح يستخدم جميع هذه الذاكرة المتوفرة لديه، وذلك حسب المبدأ الشهير:

"Programs expand fill the memory available to hold them." أي أن البرامج ستتسع حتى تبلغ الذاكرة الموجودة في جهازك، وهذا بالفعل ما يتحقق، فالبرامج تتطور بشكل أساسي متزامن مع النمو في الذاكرة وتطور بنيتها.ومع ذلك فهناك أنواع من الذواكر تتوضع في هيكلية معينة في الجهاز memory hierarchy، وهذه الذواكر تتفاوت بين الذواكر السريعة المرتفعة الثمن والذواكر المتوسطة السرعة والمتوسطة الكلفة، والذواكر البطيئة الرخيصة التي لا تستخدم إلا في تخزين الأحجام الكبيرة التي لا تستخدم كثيرا من البرامج.

إن القسم الذي يدير هذه الهيكلية من أنظمة التشغيل هو الذي يسمى قسم إدارة الذاكرة memory manager. وهذا القسم يعتبر من أهم الأقسام فعلى سبيل المثال: ضعف هذا القسم في بعض أنظمة التشغيل جعل من الصعب أن تتطور هذه الأنظمة بحيث تحوي واجهة تخاطبية مع المستخدم.

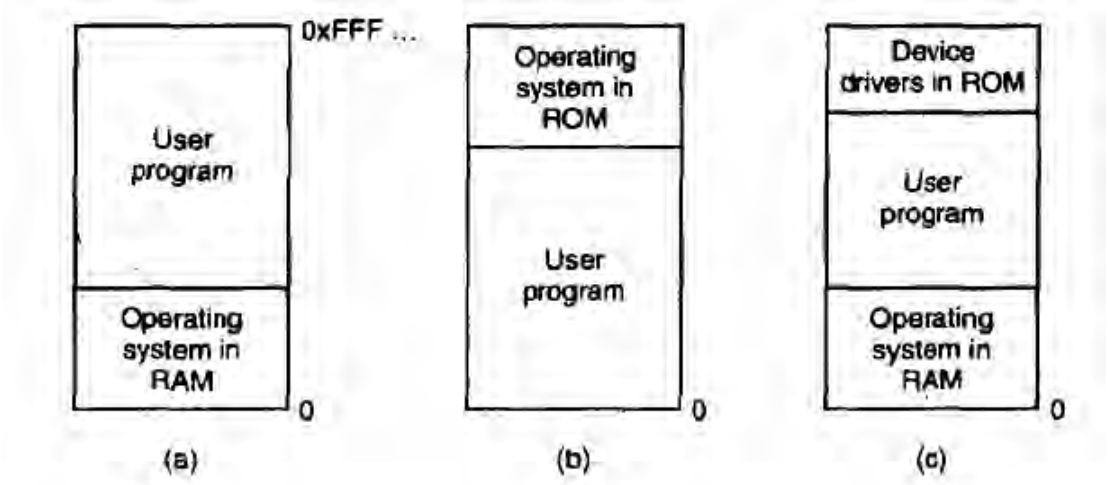
إن المهمة الأساسية لهذا القسم هو تتبع الأقسام التي استخدمت في الذاكرة والأقسام التي لم تستخدم فيها، وتحجز الذاكرة للعمليات (Process) التي تحتاج للذاكرة، وتحرر الذاكرة عندما لم تعد هذه العملية بحاجة لها، وتنظيم الانتقال في المعلومات بين الذاكرة والقرص الصلب، وذلك في حال كانت الذاكرة صغيرة جداً، ولا تتسع لجميع العمليات الفعلية في الوقت الحالي.

4 .1 -مبادئ أساسية في إدارة الذاكرة

تقسم إدارة الذاكرة إلى قسمين أساسيين: إدارة الذاكرة التي تقوم بتنظيم حجز الذاكرة إضافة إلى نقل الـ Processes سندرسه.مرة إلى القرص وبالعكس في حال امتلاء الذاكرة (swapping and paging) والقسم الآخر الذي يقوم بتنظيم حجز الذاكرة فقط.و دعونا بداية نلقي نظرة سريعة على القسم الثاني الذي لأنه سيكون خارج المجال الذي سندرسه.

4 1 1- برنامج وحيد على الذاكرة
Monoprogramming without Swapping or Paging:

في هذا النموذج لا يحتوي النظام إلا على برنامج واحد يقوم بالاشتراك مع نظام التشغيل في الذاكرة ويكون توضع هذه الذاكرة في ثلاثة أشكال كما هو موضح في الشكل ، حيث لا يعمل إلا برنامج وحيد في لحظة واحدة على النظام، وفي حال طلب أوامر من الدخل تتجه هذه الأوامر مباشرة إلى البرنامج، وعند الانتهاء من البرنامج، يقوم نظام التشغيل بتحرير الذاكرة ومن ثم يظهر محث الأوامر، وذلك من أجل تلقي الأوامر لتحميل برنامج جديد.



الشكل16

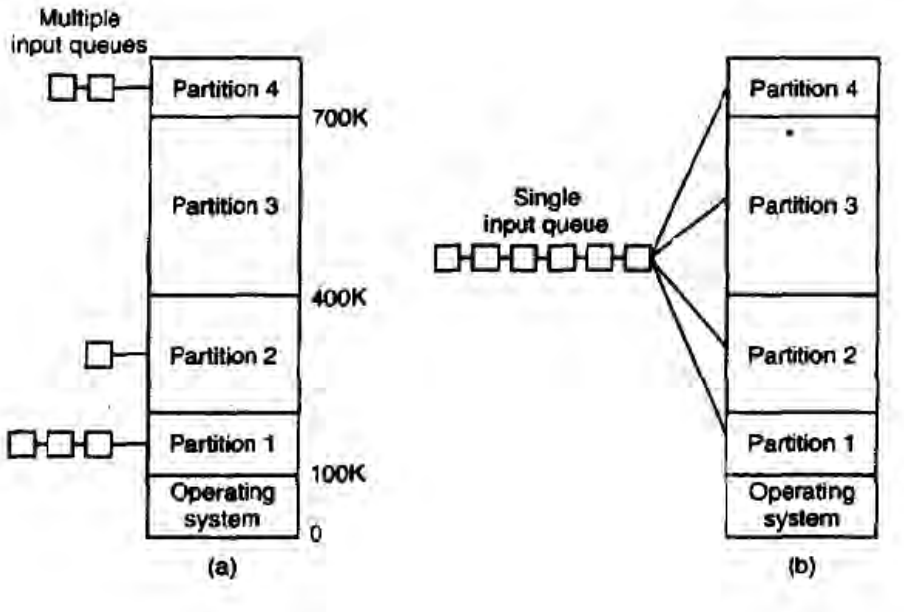
يظهر الشكل الأعلى أسلوب بدائي جدا في إدارة الذاكرة، وهنا نلاحظ عدة توضعات بين ذاكرة البرنامج ونظام التشغيل.

4 1 2- عدة برامج على الجهاز مع حجم ثابت محدد مسبقا
Multiprogramming with Fixed Partitions:

في هذه الحالة والتي تستخدم مع الحواسب الصغيرة في هذه الأيام نلاحظ أن من الممكن أن يكون أكثر من برنامج على الجهاز في نفس اللحظة، وهذا يعني أن البرنامج الذي يعمل على الجهاز في هذه اللحظة كان في انتظار طلب من القرص أو من أجهزة الدخل والخرج، فعندها سيملك البرنامج الآخر (الموجود على الذاكرة) CPU وهذا يوفر إمكانية استخدام وحدة المعالجة CPU بشكل كامل. على كل حال، فإن هذا النموذج يكون قادرا على تشغيل أكثر من برنامج في نفس الوقت.يعمل هذا النظام على تقسيم الذاكرة إلى عدة أقسام (بحيث يكون عددها ثابت n) ويكون الحجم من الذاكرة المعطى لكل قس معلوم، وتتم هذه الأمور بشكل يدوي في أول التشغيل. ويعطى لكل قسم حجم من الذاكرة معلوم مسبقا، ويختلف عن الآخر.

يعتبر هذا النموذج أفضل من النموذج السابق، ولكن له الكثير من العيوب، فلا يتم مثلا حجز الذاكرة على الحجم الذي يتطلبه البرنامج، وإنما يتم حجز المكان له بشكل ثابت، مما يجعل

البرامج الصغيرة تتوضع في بعض الأحيان في أقسام كبيرة الحجم، في حين تتوضع البرامج الكبيرة في أقسام صغيرة في بعض الأحيان، ولكن من الأفضل الذكر أن هذا النموذج سهل البرمجة على مستوى نظام التشغيل، ولقد تم استخدام هذا النموذج في نظام التشغيل OS/360 من شركة IBM، ويطلق عليه اختصارا MFT (Multiprogramming with OS/MFT)



الشكل17

3 1 4 -التبديل Swapping

يعتبر النموذج السابق سهل البرمجة وذو فعالية عالية في كثير من المهام، ولكن مع أنظمة الـ Timesharing أو الأنظمة التشغيلية والتي تحوي واجهات رسومية GUI(والتي تحتاج إلى ذاكرة كبيرة) في الأجهزة لشخصية ذات الحجم الذاكري الصغير نسبيا، فالأمر يكون مختلفا، لأنه في أغلب الأحيان لا تكون هناك ذاكرة كافية للعمليات التي تعمل في الوقت الحالي على الجهاز من العملية الحالية على القرص، ولذلك فإن المهام الإضافية ستكون على القرص حتى يأتي الوقت الذي يتم فيه تفريغ الذاكرة ويتم تحميها بشكل ديناميكي، ومن أجل هذا تم إيجاد تقنية الـswapping.

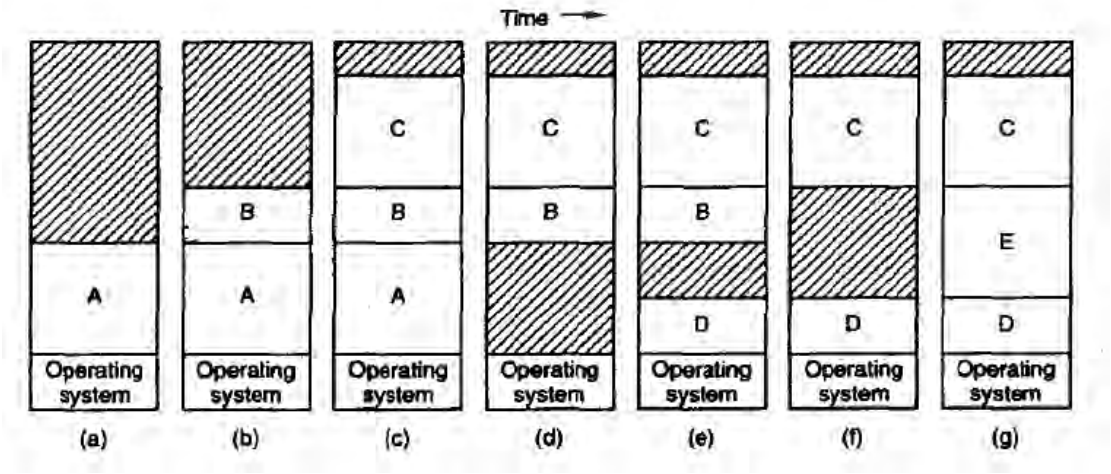
هناك حلان لهذه المشكلة على الأجهزة الشخصية:

Swapping: وهي الاستراتيجية الأبسط من هذه الاستراتيجيات، وفيها يتم جلب العملية وقت التنفيذ المخصص لها، وتنفيذها لفترة محددة(وهي الفترة المخصصة لهذه المعالجة Process time sharing)، ثم بعد ذلك بعد ذلك إعادتها إلى القرص.

Virtual Memory: والتي تمكن البرنامج من العمل حتى وإن كان جزء منه فقط يقع في الذاكرة الحقيقية.

والآن سنركز على دراسة الـ Swapping، ففي المثال التالي الموضح في الشكل فإن العملية A تكون فقط على الذاكرة، وبعدها يتم جلب العمليتين B & C من القرص(في حال تم إخراجهن سابقا من الذاكرة إلى القرص) أو إلى التنفيذ (في حال كان ذلك أول استدعاء لهن)، ثم

بعد ذلك يتم إنهاء العملية A أو يتم نقلها إلى القرص، وبعد ذلك يتم إدخال الـ D وإخراج الـ B، وأخيرا يتم إدخال الـ E.



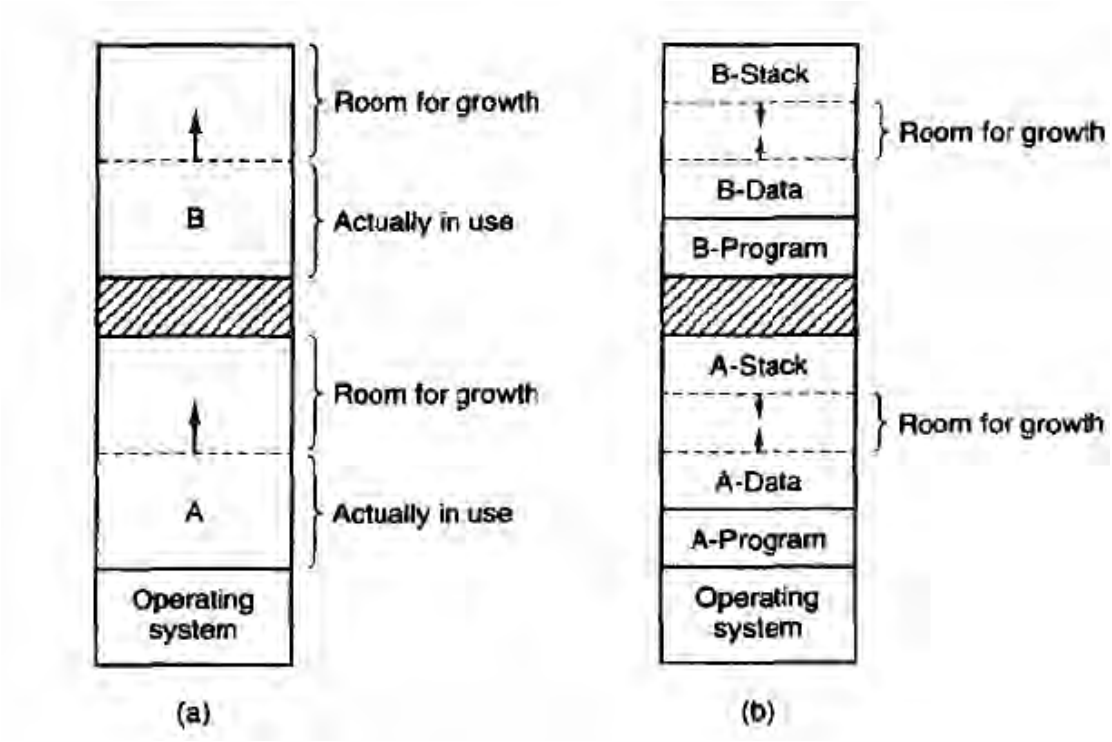
الشكل18

ويتم فيه تغيير مواضع الذاكرة عندما يتم نقل العمليات من الذاكرة وإليها، حيث أن المناطق المظللة تعبر عن مناطق فارغة.

إن الفرق بين نموذج MFT، ونموذج الأجزاء المتغيرة، هو أن الأخيرة تتغير فيها عدد الأقسام وإحجامها ومواقعها بشكل ديناميكي، بينما الأولى يكون فيها هذه المعايير ثابتة.إن النموذج المرن يحقق أفضلية عليا في استخدام الذاكرة، وذلك من خلال الديناميكية التي يوفرها، ولكن من الجهة الأخرى فإنها تحقق نوع من التعقيد في الحجز وتحرير الذاكرة، بالإضافة إلى تتبع العمليات التي تتم على الذاكرة.أما بالنسبة لمشكلة الفجوات التي يشكلها الحجز الديناميكي، فمن الممكن أن نتجاوزها عن طريق ما يسمى الـ Memory Compaction والتي تقوم بإزاحة الأجزاء المليئة لتلتصق ببعضها وتبدو على أنها جزء واحد، مما يقضي على الفجوات المتشكلة من الحجز الديناميكي، ولكن في أغلب الأحيان لا يتم استخدام هذا الحل، لأنه يستغرق كمية كبيرة من طاقة الـ CPU والتي تقوم بهذا العمل في معالج من 16-bytes في الميلي ثانية الواحدة (وذلك في آلة من 32-MB) في 2 ثانية بالنسبة للذاكرة ككل.

تكمن المشكلة في الحجز الديناميكي على انه يقوم بالحجز على الذاكرة بناء على الحجم المطلوب تماما ليس على انه حجم ثابت، بينما في حال كان حجز جزء ثابت من الذاكرة لا يتغير، فإن الأمر سيكون ابسط مما هو عليه في حالتنا السابقة، حيث ستحجز المطلوب دون زيادة ولا نقصان أقل.فإذا كان ازدياد حجم العملية على الذاكرة ممكنا، على سبيل المثال عن طريق الحجز الديناميكي للذاكرة (أي في قسم الكومة كما في لغات البرمجة) فإن المشكلة ستكون عندما تتوسع هذه العملية لتشمل قسم أكبر من الذاكرة. فإذا كانت هناك فجوة مجاورة لهذه العملية فإن الحجز سيكون على هذه المنطقة ويكون الامتداد للذاكرة المتعلقة بهذه العملية على هذه الفجوة. بينما على النظير الآخر، إذا كان هناك عملية أخرى مجاورة لهذه العملية التي تريد التوسع، فهنا ستكون المشكلة، فإما أن يكون هناك انتقال منطقة العملية إلى فجوة تتسع للحجم الجديد المتطلب من الذاكرة، أو سيتم نقل العملية المجاورة القرص عن طريق عملية التبديل Swapping، وفي أسوأ الأحوال ستكون الذاكرة كلها ممتلئة، إضافة إلى أنه لم يعد هناك مساحة على القرص الصلب، عندها سيقوم نظام التشغيل بقتل هذه العملية أو يقوم بعملية انتظار.

إذا كان من الممكن توقع حجم الذاكرة الذي ستأخذه العملية أثناء عملها، فالحل المقترح أن يقوم مدير الذاكرة بحجز حيز إضافي على الذاكرة في حال تشغيل هذه العملية، أو عند إعادتها من القرص الصلب إلى الذاكرة، والشكل التالي يوضح عملية الحجز الإضافي على الذاكرة.



الشكل19

حيث في الشكل (a) يكون حجز منطقة فارغة من أجل ازدياد حجم المعطيات، أما الشكل (b) يتم حجز مكان فارغ من اجل الازدياد في حجم المعطيات والازدياد في حجم المكدس.

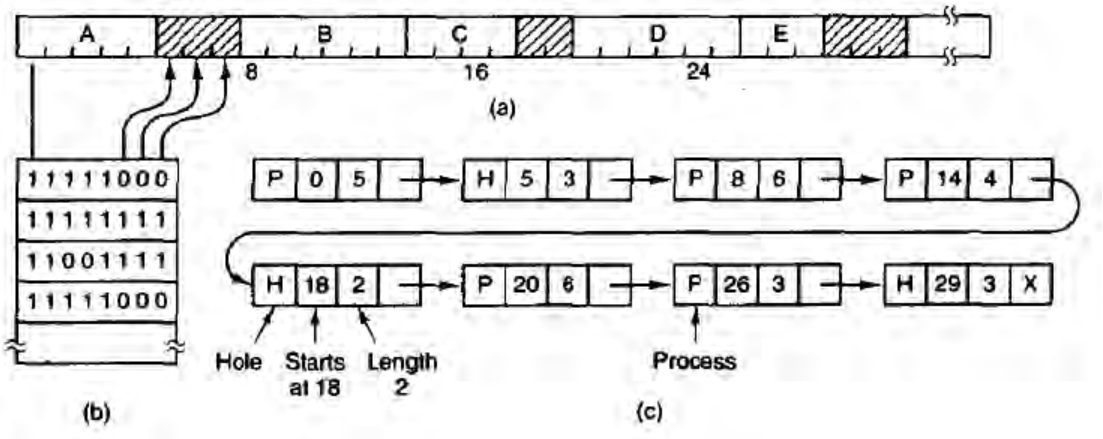
4 2 ـإدارة الذاكرة

عندما تتم عملية الحجز الديناميكية، فمن واجب نظام التشغيل أن يدير هذه الذاكرة. وعلى هذا الأساس فهناك طريقتين لتتبع عملية حجز الذاكرة ومراقبتها، وذلك عن طريق المصفوفات المؤلفة من الـ bits واللوائح المترابطة:

4 2 1 ـإدارة الذاكرة من خلال الـ Memory Maps

من خلال هذا النموذج يتم تقسيم الذاكرة إلى وحدات، ربما تكون هذه لوحدات بضعة كلمات ذاكرية، ومن الممكن أن تكون عدة كيلوات من الذاكرة، يتم من خلال هذه الخريطة الذاكرية معرفة الوحدات المحجوزة من خلال هذه الخارطة، فكل bit يعبر عن وحدة من هذه الوحدات الذاكرية، فإذا كانت قيمة هذا البت الذاكري هو (0) فإن هذه الحجرة الذاكرية حرة، أما إذا كانت هذه الحجرة ممتلئة (قيمتها مساوية لـ 1) فإن هذه الوحدة تكون محجوزة، وطبعا من

الممكن أن يكون الأمر معكوسا، بحسب أسلوب النظام.يظهر الشكل 19 التالي جزء من الذاكرة وخريطة البتات الموافقة لها.



الشكل20

الشكل يظهر شكل الذاكرة مع خمسة عمليات ومع الأماكن الفارغة، يشير كل قسم إلى المكان الفارغ من الذاكرة، حيث يعني القسم المظلل أن الذاكرة فارغة، وهذا يعني أن نفس المكان المعبر عن هذه الحجرة في الخارطة مساوي للصفر، حيث يعبر الشكل (b) عن هذه الخارطة، بينما يعبر الشكل (c) عن التمثيل الذاكري عن طريق اللوائح المترابطة

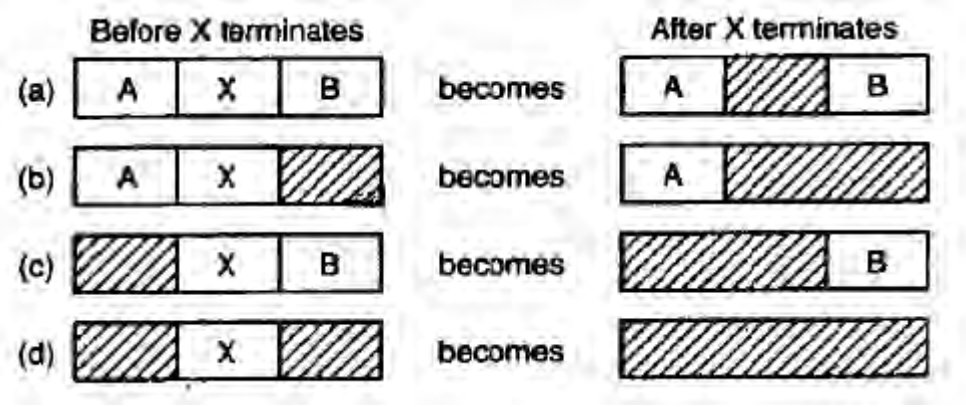
كلما كبر حجم الذاكرية الفيزيائية فإن هذا يعني أن خارطة البتات ستكبر، وهذا يعني أن هناك كمية كبيرة من الذاكرة سيتم صرفها على هذه الخريطة، ولكن ليكن الوحدة هذه ذات حجم 4-byte فهذا يعني أن كل 32 بت يتم تمثيله ببت واحد، وهذا يعني أنه في حال كان لدي 32n بت من الذاكرة فإن هذا يعني بالمقابل أن لديه n بت من الخريطة، مما يعني أن 1/33 من الذاكرة. وفي حال تم تكبير حجم هذه الوحدة فإن هذا يعني أن الخارطة البتية ستصغر، ولكن بالمقابل سيكون هناك هدر للذاكرة من خلال الفجوات المشكلة من كل وحدة ذاكرية.

تعتبر الخارطة البتية طريقة سهلة ليتم إتباع العمليات التي تحصل على الذاكرة من حجز وتحرير وذلك من خلال حجم ثابت من الذاكرة، وذلك لأن هذا الحجم يتم من خلال حجم الوحدة الذاكرية المتبعة، ومن خلال حجم النظام. والمشكلة الأساسية في هذه الطريقة أنه في حال طلب k وحدة من الذاكرة فإن هذا يعني أن على مدير الذاكرة البحث على k وحدة فارغة متجاورة، وذلك بسبب البحث الخطي عن هذه الحجرات في الخارطة.

4 2 2 -إدارة الذاكرة من خلال الـ Linked List

هناك طرق أخرى لعملية مراقبة الذاكرة وعمليات الحجز والتحرير، وهي من خلال اللوائح المترابطة، حيث كل لائحة إما أن تعبر عن مكان محجوز في الذاكرة لإحدى العمليات، أو عن فجوة بين مكانين محجوزين لعمليتين مختلفتين.كما هو في الشكل (3-5) وفي القسم (c) فإن هذه اللائحة تقوم بتمثيل الجزء الذاكري الموجود في الشكل (a)، كل لائحة من هذه اللوائح يحوي عدة قيم، فالـ (H) يعبر عن الفجوة، في حين (P) يعبر مكان مخصص في الذاكرة، وكذلك مكان بداية هذه الذاكرة، وطول هذه المنطقة من الذاكرة، ومؤشر إلى اللائحة التالية.

تـتم عمليـات الحجـز والتحريـر علـى الـذاكرة وتمثـل هـذه العمليـات علـى اللائحـة بخوارزميات معينة لن نذكرها هنـا، ولكن من الممكن أن تمثل عن طريق الشكل التـالي، حيث يعبر الشكل عن عملية إلغاء حجز القسم (X) من الذاكرة وبنـاء علـى وضـع الجيران فإن عملية التحرير هذه ستقوم بدمج الفراغات المجاورة مع الفراغ الناتج عن عملية التحرير، وبالتالي سيتم إنقاص هذه السلسلة بلائحة واحدة.



الشكل21

يعبر عن عملية تحرير الذاكرة المتعلقة بالعملية X

هناك عدة خوارزميات متبعة في عملية الحجز، من بينها الخوارزمية First Fit ومن خلالها يقوم مدير الذاكرة بالبحث عن فجوة فارغة حجمها أصغر من حجم الذاكرة المطلوبة، وأول ما تم العثور عليها فإن مدير الذاكرة يقوم بكسر هذه اللائحة إلى قسمين وحجز الأول بما يناسب الحجم المطلوب، والإبقاء على القسم الآخر على أنه مساحة فارغة قابلة للحجز مرة أخرى. وتعتبر هذه الخوارزمية هي الأسرع من حيث البحث، لأنها عملية المسح بحثا عن الذاكرة تكون أقل ما يمكن.

هناك خوارزمية أخرى قائمة على تعديل خفيف في خوارزمية الـ First Fit وهي خوارزمية الـ Next Fit، فهي تشبه كثيرا الخورازمية الأولى إلا أنها تختلف عنها في أنها تقوم في البحث بالمرة الثانية من المكان الذي توقفت فيه في المرة السابقة، ولا تعود بالبحث من البداية، ولقد أثبتت الدراسات من شخص يدعى Bays في عام الـ 1977 أن خوارزمية الـ Next Fit أبطأ من حيث الأداء من خوارزمية الـ First Fit.وهناك خوارزمية أخرى معروفة والتي تدعى Best Fit، والتي تقوم بالبحث في كامل السلسلة عن الفجوة الأكثر ملائمة فيها (أي الفجوة الأصغر والتي تلائم الحجم المطلوب)، وذلك بدلا من كسر فجوة كبيرة إلى عدة أقسام وذلك من اجل استخدام هذه الفجوة الذاكرية لاحقا.

إن خوارزمية الـ Best fit أبطأ من خوارزمية الـ next fit وذلك لأنها تقوم بعملية مسح كامل للسلسلة، ولكن الأمر الأكثر غرابة أن الخوارزمية الأولى تخلف وراءها الكثير من الشقوق الصغيرة التي لا يمكن استخدمها في كثير من الأحيان بسبب صغرها، في حين أن الخوارزمية الثانية تقوم بترك الفجوات الكبيرة التي من الممكن أن تستخدم لاحقا.

إن المشكلة الأخيرة يمكن أن تحل من خلال استخدام خوارزمية ال Worst Fit والتي تقوم بمسح اللائحة كاملة ولكن لا تقوم بالحجز إلا في الفجوات الأكبر حجما حتى يتم استخدام الفجوات الأصغر الناتجة عن هذه الفجوات مرة أخرى (حيث من المفترض أن تكون هذه الفجوة

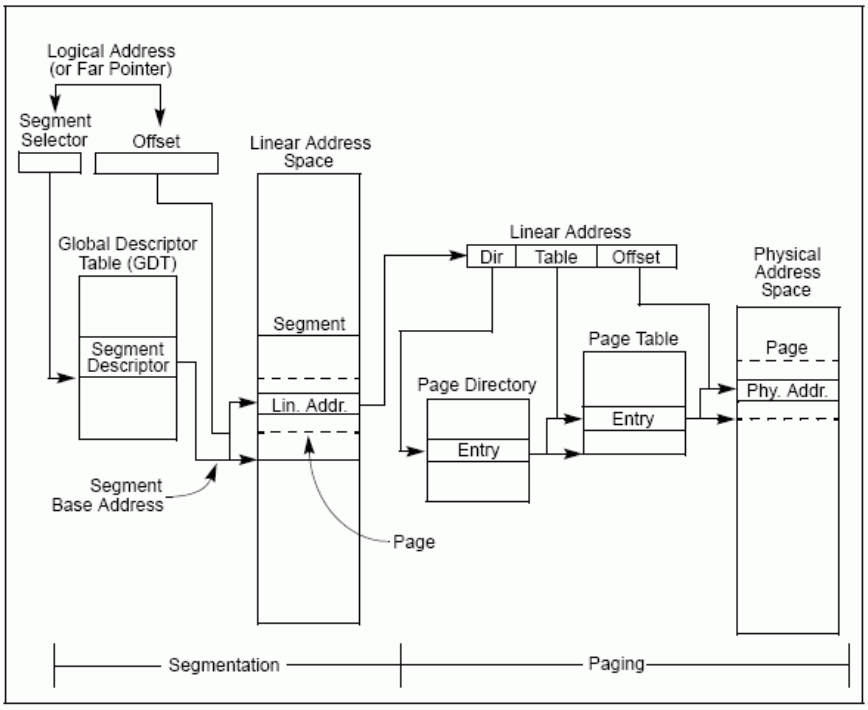
كبيرة كفاية لتستخدم مرة أخرى).ولقد أشارت المحاكاة المتبعة في هذه العملية أنها أيضا فكرة غير جيدة.

يتم تسريع هذه الخوارزميات الأربعة السابقة من خلال جعل لائحة خاصة بالعمليات ولائحة خاصة بالمعالجات، ومع أن هذه العملية تقوم بتسريع عملية حجز الذاكرة إلا أنها تقوم من وجهة نظر أخرى بتبطيء عملية التحرير، إذ يتم فيها تخير الجزء الذاكري، ومن ثم إلحاق هذا الجزء في المكان المناسب من سلسلة الفجوات.يمكن تسريع هذه العملية أيضا من خلال ترتيب هذه اللائحة المترابطة تصاعديا، فيصبح حينها عملية البحث عن طرق خوارزمية الـfirst fit تشبه عملية البحث من خلال best fit وتصبح الغاية من الخوارزمية الأخيرة محققة من خلال الـ first fit ولا داعي للبحث في جميع لوائح السلسلة.

4 3 -الذاكرة الافتراضية Virtual Memory

لقد كانت هناك مشكلة كبيرة لدى المبرمجين في التعامل مع البرامج الكبيرة في أنظمة لها ذواكر صغيرة، عندها كان المبرمجون مضطرين إلى تقسيم البرامج إلى عدة أقسام (Modules)، ففي البداية يتم تحميل احد هذه الوحدات إلى الذاكرة، ثم بعد أن يقوم المعالج بتنفيذ هذه الوحدة يقوم البرنامج بنزع هذه الوحدة ومن ثم وضع الوحدة التالية في عملية التنفيذ، وهكذا فإن هذه العملية تقوم بتوفير إمكانية عمل البرنامج الذي يحتاج مثلا إلى 16-MB في نظام لديه 4-MB. ولكن المشكلة أن هذه العملية كانت تتعب المبرمجين، إذ كان على المبرمج أن يقسم البرنامج إلى عدة أقسام صغيرة يمكن تحميلها إلى هذه الذاكرة، وكانت تسمى كل وحدة عادة بالاسم Overlays، أضف إلى ذلك فإن عملية نقل الوحدات من الذاكرة إلى القرص وبالعكس كان يقع أيضا على عاتق المبرمج.

لقد تم حل هذه المشكلة وتخفيف الصعوبات التي كانت على المبرمج، وذلك عن طريق الاختراع الذي سُمي Virtual Memory. إن الفكرة الأساسية لهذه العملية هي أن البرنامج يحتاج إلى مكان ذاكري للشفرة وللمتحولات والمكدس بما يزيد على الذاكرة الموجودة في النظام، وبالتالي فإن نظام التشغيل يقوم بوضع جزء من هذه البرنامج على الذاكرة والباقي يكون على القرص الصلب، حيث يكون القسم الموجود على الذاكرة منتقى بنوع من العناية فيكون هو القسم المتعلق بالعملية الجزئية التي تنفذ فيه في الوقت الحالي.كذلك فإن هذه العملية من الممكن أن يعمل على نظام تشغيل يعتمد مبدأ الـ Multiprogramming في وقت واحد. فيكون الحال في عملية جلب قسم الشفرة المتعلقة بالبرنامج في وقت معين هو نفسه الحال الذي يكون في عملية طلب قراءة أو كتابة من الذاكرة.

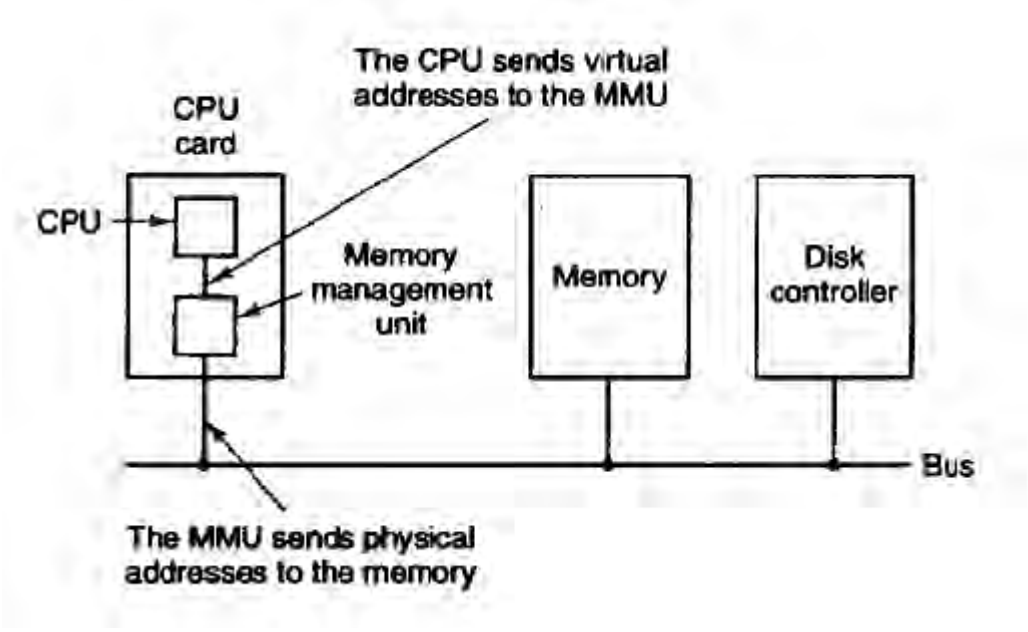


الشكل 22

4 4 -مفهوم الصفحات Paging

إن معظم النماذج التي تعتمد على الـ Virtual Memory على تقنية الـ Paging والتي سنتحدث عنها الآن، حيث يتم فيه إعطاء عنوان منطقي غير العنوان الخطي الذي اعتاد عليه المبرمجون في المعالجات القديمة، أو في منطق بناء أنظمة التشغيل القديمة، حيث يتم في أغلب الأنظمة الحديثة إرسال هذه العناوين المنطقية إلى وحدة تسمى وحدة إدارة الذاكرة MMU (Memory Management Unit) والتي تقوم بتحويل هذا العنوان إلى عنوان فيزيائي، حيث أن خريطة العناوين المنطقية تكون اكبر بكثير من خريطة العنوان الفيزيائية الحقيقية.

إن كل نظام لديه حجم من الذواكر مختلف عن النظام الآخر، وبما أن على نظام التشغيل أن يتعامل مع النظام بشكل مستقل عن طبيعته الفيزيائية وبغض النظر عن الحجوم الذاكرية التي لديه، فهذا يعني أن عليه أن يفترض وجود ذاكرة ذات حجم ثابت (وإن كانت افتراضية) ويتعامل معها بأسلوب ثابت بغض النظر عن الحجم الحقيقي للذاكرة، وهذا الأمر يتم تحقيقه من خلال ما يسمى بالذاكرة الافتراضية التي تعطي خريطة وهمية تدل على حجم وهمي ثابت من الذاكرة، ويتعامل نظام التشغيل مع هذه الخريطة بدلا من تعامله مع الخريطة الفيزيائية بشكل مباشر، وتقوم وحدة الإدارة بالتحويل بين الخريطة الافتراضية والخريطة الفيزيائية كما هو موضح في الشكل.



الشكل23

الشكل وهو يعبر عن مكان وعمل الوحدة الذاكرية MMU

لقد تم اعتماد هذه التقنية كثيرا في بناء أنظمة التشغيل، وأخيرا تم دعمها في مجال الـ Hardware، وبما أن معالجات Pentium من شركة Intel هو الأكثر استخداما فإننا سنعتمد على النموذج الخاص به في تمثيل الصفحات Paging وشكل الأعلم (Flags) التي يتم استخدمها في هذا المعالج للتعامل مع الذاكرة و تجدر الملاحظة إلى أن المعلومات التي ستذكر لاحقا كلها مأخوذة من الوثائق المتعلقة بمعالج الـ Pentium من شركة Intel. أو من كتاب Understanding The Linux Kernel.

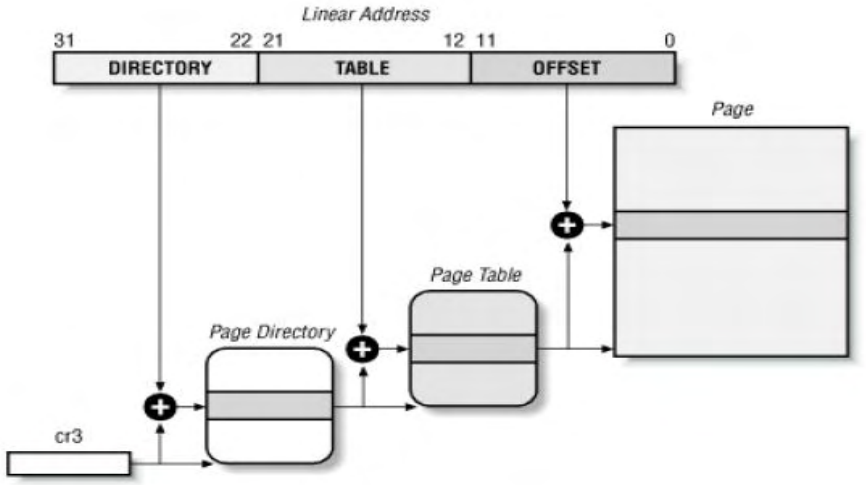
وبشكل مختصر فإن هذه العملية (Paging) هي عملية التحويل من العنوان الخطي إلى العنوان الفيزيائي، حيث يتم التأكد بالإضافة لذلك من صلاحيات الدخول معتمدين على العنوان الخطي، وإذا كان الدخول إلى العنوان المطلوب غير ممكنا، فهذا يولد Exception يدعى Page fault Exception. ومن اجل الفعالة يتم تقسيم الذاكرة إلى مجموعات ذات أحام ثابتة تسمى pages، وتكون فيها العناوين الذاكرية الخطية المتجاورة ذات عناوين فيزيائية متجاورة.يتم التحويل من العنوان الخطي إلى العنوان الفيزيائي عن طريق بنية معطيات ندعى الـ Page Tables والتي يتم تخزينها في الذاكرة الأساسية من النظام، ويجب أن تهيئ بالقيم المناسبة قبل أن يتم تحويل نمط المعالج إلى Paging Unit.

في معالج Intel يتم تفعيل الـ Paging من خلال الـ PG flag في المسجل cr0 في المعالج، وعندما يكون PG = 0، يتم ترجمة الـ Linear Address إلى Physical Address.تقوم وحدة الـ Paging بالتعامل مع 4-KB من الذاكرة على أنها page واحدة، وذلك في معالج الـ Intel، ويتم تقسيم العنوان الخطي المؤلف من 32-bit إلى ثلاث أقسام، والتي هي:

Directory: ويحتل الخانات العشر الأكثر أهمية في هذا العنوان.
Table: والذي يحتل الخانات العشر التي تقع في الوسط.
Offset: ويحتل الخانات الـ 12 الأقل أهمية.

يتم التحويل من العناوين الخطي إلى العنوان الفيزيائي بمرحلتين، الأولى باستخدام البنية Page Directory، والثانية باستخدام البنية Page Table.يتم تخزين العنوان الفيزيائي المتعلق في الـ Page Directory في المسجل cr3، يدل الحقل في العنوان الخطي الذاكري المتعلق بالـ Page Directory على المدخل في هذه الصفحة، والذي يشير إلى أحد الـ Page Table، كما أن الحقل في العنوان الذاكري الخطي المتعلق بالـ Page Table يدل على المكان الذي يحوي العنوان للصفحة الهدف، ويدل الحقل الأقل أهمية في العنوان الفيزيائي على المكان المطلوب تحديده تماما في هذه الصفحة، وبما أن عدد هذه البتات في الحقل الأخير يساوي الـ 12 فإن طول الصفحة يجب أن يكون 4096 أي 4-KB وهذا ما يكون تماما في معالجات الـ Intel.

يشير الشكل التالي إلى العملية التي تم شرحها في الأعلى:



الشكل 24

الشكل (3-7) يدل على شكل الـ Paging في معالج الـ Intel 80X86

إن طول كل من الحقول المتعلقة بالـPage Directory والـ Page Table هي عبارة عن حقول من طول 10-bit أي أنها تكون قادرة على عنونة 1024 عنوان، وهذا يعني أن كل Page Directory يمكنه أن يعنون 232=4096 x 1024 x 1024 أي تماما كما هو الحال في عملية العنونة الفيزيائية.

إن كل من الـ Page Directory والـ Page Table له نفس البنية من حيث الأعلام، وكل من هذه المداخل تحوي على البنى التالية:

Present flag: إذا كان هذا العلم مساو للواحد فإن هذا يعني أن هذه الصفحة موجودة في الذاكرة الحقيقة، و إلا فإنها موجودة على القرص، وعند الطلب وعدم إيجادها فإن هذا سيؤدي إلى قذف استثناء والذي يسمى بالـPage fault.

Accessed flag: يدل هذا العلم على أن هذه الصفحة قد تم الدخول لها للحصول على المعلومات، وهذا العلم يتم استخدامه في أنظمة التشغيل من أجل معرفة الصفحات الأنسب التي يمكن نقلها غلى القرص في الـSwapping (حيث يكون الصفحات الأقل استخداما هي الصفحات

التـي يكـون فيهـا هـذا العلـم مسـاو للصـفر، أي أنهـا الأنسـب للنقـل إلـى القـرص فـي عمليـة الـ (Swapping).

Dirty flag: وهذا العلم يتم استخدمه في الـ page Table وهو يدل على أن هذه الصفحة قد تم التغيير في قيمها بعد تحميلها من القرص أو مكان آخر، وهذا يعني أن هذه الصفحة يجب إعادة حفظها قبل حذفها، ويستخدم هذا العلم في عملية الـ Swapping ومعرفة الصفحات الأنسب لها. ولا يتم إعادة تهيئة هذا العلم إلا من قبل نظام التشغيل المستخدم، أي ليس للمعالج أي تأثير على هذا العلم.

Read/Write flag: والتي تحوي الصلاحيات في عمليات القراءة والكتابة.
User/Supervisor flag: والتي تحوي على المستوى السماحيات في الدخول إلى الصفحات والـ Page Table.
PCD & PWT flags: والتي تحدد البنية التي ستعالج فيها الصفحات والـ Page Table في الـ hardware cache.
Page Size flag: وهذا العلم لا يتم تطبيقه إلا على الـ Page Directory، فإذا تمت هذه العملية فإنها تشير إلى أن حجم الصفحة أصبح 4-MB.

4 5- (TLB)Transition Lookaside Buffer

إن عملية طلب التحويل بين العنوان الخطي و العنوان الفيزيائي تستغرق وقتا لا بأس به، وقد أثبتت الدراسات في هذا المجال أن عملية طلب الصفحة والتحويل إلى العنوان الفيزيائي لهذه الصفحة يحدن بشكل متكرر بعد أول طلب لها من البرنامج، وهذا يعني أن عملية طلب الصفحة نفسها سيتكرر عدة مرات، مما يجعل هناك هدرا للوقت، من اجل هذا قام مصممو أنظمة التشغيل بنوع من الذاكرة المؤقتة التي تحفظ العناوين الفيزيائية التي تطلب في لحظة ما، وهذه العناوين تحفظ وتدار في وحدة من الوحدات الفيزيائية على النظام، والتي تسمى بالـ Transition Lookaside Buffer(TLB)، حيث يدير هذا النظام عملية حفظ وإدارة العناوين، ولقد أثبتت التجارب أن عملية طلب العنوان بشكل متكرر تتم بنسبة 2/3، تفيد هذه العملية في استرجاع العناوين الفيزيائية للعناوين المطلوبة سابقا بشكل سريع، وتكون هذه الوحدة متوضعة في الـ MMU.

4 6- معالجة أخطاء الصفحات

سنذكر الآن بنوع من التفاصيل عن المعالجة التي تتم للأخطاء، حيث يكون تسلسل الأحداث كما يلي:

ينقل العتاد التحكم إلى النواة، حيث يخزن عداد البرنامج في المكدس، في بعض الأجهزة يتم أيضا تخزين التعليمة الحالية في مسجلات خاصة في المعالج.

يبدأ تنفيذ إجراء فرعي في لغة التجميع لتخزين المشاركات العامة والمعلومات الحساسة الأخرى، وذلك لمنع نظام التشغيل من تدميرها، تستدعي هذه الشفرة نظام التشغيل كإجراء عادي.

يكتشف نظام التشغيل حدوث خطأ صفحة، ويحاول اكتشاف الصفحة الظاهرية المطلوبة. عادة تكون هذه المعلومات موجودة في أحد مسجلات العتاد، إذا لم تكن هذه المعلومات متوفرة يجب على نظام التشغيل استرداد عداد البرنامج، وإحضار التعليمة وتفسيرها بشكل برمجي لمعرفة ما كانت تفعله أثناء حدوث خطأ.

حالما يكتشف العنوان الظاهري الذي سبب الخطأ، سيختبر نظام التشغيل هذا العنوان لمعرفة إذا كان صالحا أم لا، وإذا كان نوع الوصول متفق مع نوع الحماية (مثلا إذا كانت العملية تقوم بمحاولة الكتابة على صفحة معدة للقراءة فقط، عندها يجب توليد خطأ حماية). إذا فشل الاختبار، ترك إشارة إلى التعليمة أو تقتل. أما إذا كان العنوان صالح، ولم يحدث خطأ حماية، فإن النظام يبدأ بالبحث عن إطار صفحة فارغ. إذا لم يكن هناك إطار فارغ، يتم تشغيل خوارزمية استبدال الصفحات لاختبار صفحة من أجل طردها.

إذا كان إطار الصفحة المختارة متسخاً، تتم جدولة الصفحة من أجل نقلها إلى القرص، ويحدث تبديل للسياق مما يؤدي إلى إرجاء العملية التي سببت الخطأ وترك عملية أخرى تعمل حتى انتهاء النقل إلى القرص. على كل حال يعلم الإطار على أنه مشغول حتى لا يتم استخدمه من عملية أخرى.

حالما يصبح إطار الصفحة نظيفا (إما مباشرة أو بعد نقله إلى القرص)، يبحث نظام التشغيل عن عنوان القرص للصفحة المطلوبة، ويقوم بجدولة عملية قرص لقراءة الصفحة. تبقى العملية متوقفة أثناء تحميل الصفحة وتعمل عملية أخرى أثناء ذلك.

عندما تعلن مقاطعة القرص وصول الصفحة المطلوبة، يتم تعديل جدول الصفحات كي يشير إلى موقع الصفحة الجديدة، ويعلم الإطار عن طريق الأعلام أنه في حالة طبيعية.

تعاد التعليمة التي سببت الخطأ إلى الحالة التي كانت عليها عندما بدأت ويعاد تهيئة عداد البرنامج بحيث يشير إلى هذه التعليمة.

تجدول العملية التي تسبب الخطأ، ويعود النظام التشغيل إلى إجراء شفرة التجميع التي استدعته.

يقوم هذا الإجراء بإعادة تحميل المسجلات ومعلومات الحالة الأخرى ويعود إلى فضاء المستخدم لمتابعة التنفيذ كان شيئا لم يحدث.

4 7 -التقطيع مع التصفيح (نظام الـ MULTICS)

لقد تم في فصول سابقة ذكر التقطيع وكيف يتم استخدمه في عمليات العنونة الذاكرية، ولكن المسالة تكمن في كيفية الجمع بين عمليتي التقطيع والتصفيح.فإذا كانت المقاطعة كبيرة فإنه يصعب أو حتى يستحيل إبقاؤها بأكملها على الذاكرة الرئيسية، تقودنا هذه المشكلة إلى فكرة تصفيح هذه المقاطع،بحيث توضع الصفحات اللازمة فقط على في الذاكرة. تم استخدما التقطيع مع التصفيح في كثير من أنظمة التشغيل الهامة، سنشرح في هذا القسم أول هذه الطرق التي جمعت بين التقطيع والتصفيح والتي تسمى MULTICS، وبعدها سنتكلم عن عملية الربط فيها في معالجات الـ Intel Pentium.

يعمل هذا النظام على أجهزة Honywell 6000 وخلافها ويعطي لكل برنامج ذاكرة ظاهرية حتى 2^{18} مقطع (أكثر من 250000) يمكن أن حول كل منها إلى 65536 كلمة بطول 36 بت، وبهذا الأساس قرر مصممو MULTICS اعتبار كل مقطع كذاكرة ظاهرية وتصفيحها بحيث يجمع مزايا التصفيح (حجم المقطع ثابت وعدم الحاجة إلى إبقاء كل المقطع في الذاكرة) مع مزايا التقطيع (سهولة الحماية والتقطيع إلى وحدات نمطية والحماية المشتركة).يملك كل برنامج MULTICS جدول مقاطع فيه واصف واحد (Descriptor) لكل مقطع.وبما انه من الممكن وجود أكثر من ربع مليون سجل في هذا الجدول، لذلك يوضع جدول المقاطع نفسه ضمن مقطع يتم تصفيحه. ويحوي واصف المقطع إشارة تدل إذا كان المقطع في الذاكرة الرئيسية أم لا. إذا كان أي جزء من المقطع موجودا في الذاكرة، يعتبر المقطع انه موجود في الذاكرة. إذا كان المقطع في الذاكرة، فإن الواصف يحوي عنوانا بطول 18 بت يشير إلى جدول الصفحات الخاص به. بما أن العنوان الفيزيائي بطول 24 بت، والصفحات متراصفة عند حدود كل 64 بايت في ذلك النظام (يعني ضمنيا أن كل عنوان بداية كل صفحة فيه 000000 في البتات الستة الدنيا منه) فإننا نحتاج فقط إلى 18 بت في الواصف لتخزين عنوان جدول الصفحات.

يحوي الوصف أيضا حجم المقطع وبتات الحماية وبعض الأشياء الأخرى، ويوضح الشكل في الجزء b واصف مقطع في نظام الـ MULTICS. لا يوضع عنوان المقطع في الذاكرة الثانوية ضمن واصف المقطع، ولكنه يوضع في جدول آخر يستخدم من قبل معالج أخطاء المقاطع.كل مقطع عبارة عن فضاء عناوين ظاهري اعتيادي ويصفح في نفس الطريقة المتبعة في الطرق المشروحة سابقا. والحجم الاعتيادي في هذا النظام هو 1024 بايت (على الرغم من أن بعض المقاطعات الداخلية التي يتم استخدامها في نظام الـ MULTICS غير مصفحة أو مصفحة باستخدام واحدات مؤلفة من 64 بايت فقط لتوفير الذاكرة الفيزيائية).يتألف العنوان في نظام الـ MULTICS من جزأين: المقطع والعنوان ضمن المقطع. يقسم العنوان في المقطع أيضا إلى رقم الصفحة وعنوان كلمة داخل الصفحة، كما في الشكل. وعند ورود إشارة إلى الذاكرة، تنفذ الخوارزمية التالية:

يستخدم رقم المقطع لإيجاد واصف المقطع.

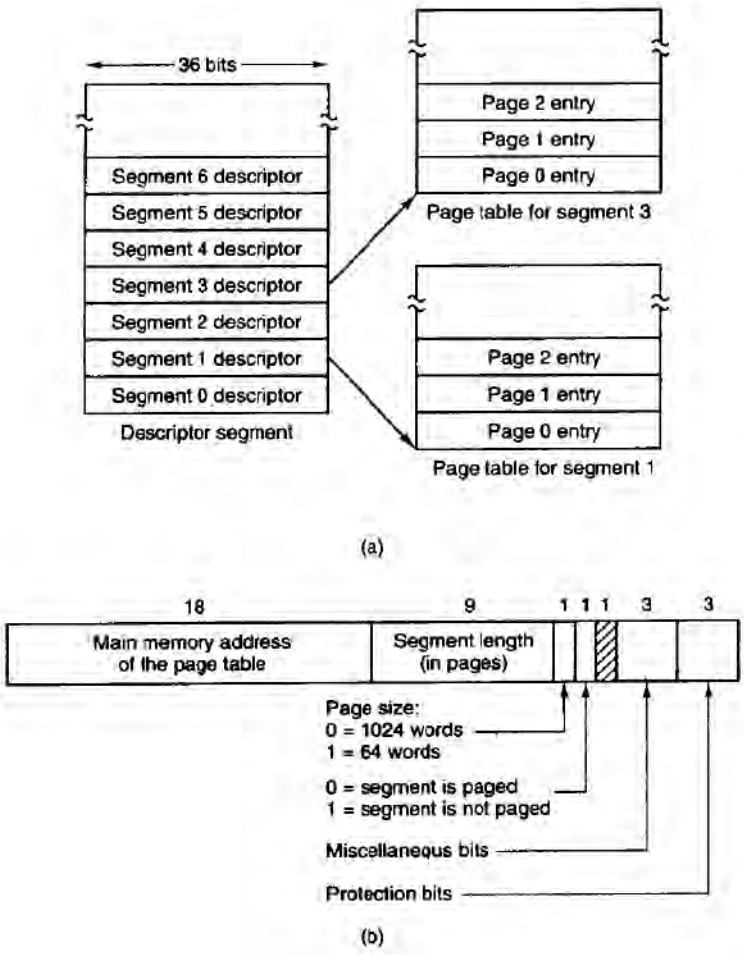
يجري اختبار إذا ما كان جدول صفحات المقطع موجودا في الذاكرة. إذا كان موجودا يتم تحديد مكانه، إما إذا لم يكن موجودا، يحدث خطا مقاطع. وإذا كان هناك خرق لمستوى الحماية، يحدث خطأ (مقاطعة).

يفحص سجل جدول الصفحات الموافق للصفحة الظاهرية المطلوبة. إذا كانت الصفحة غير موجودة في الذاكرة، يحدث خطأ صفحة. أما إذا كانت في الذاكرة، يستخلص عنوان بداية الصفحة من سجل جدول الصفحات.

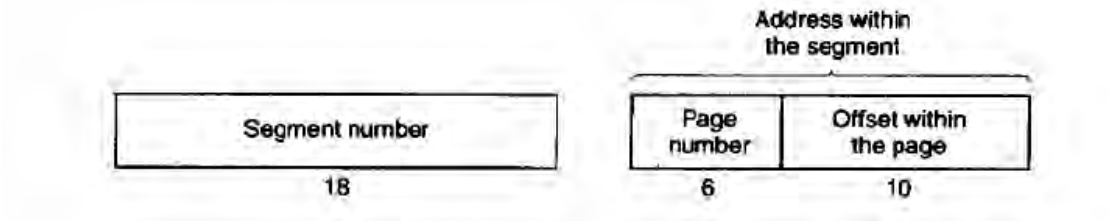
تضاف الإزاحة إلى عنوان بداية الصفحة للحصول على عنوان الذاكرة الرئيسية الذي يشير إلى الكلمة المطلوبة.

أخيرا، تتم عملية القراءة والكتابة.

يوضح الشكل هذه العملية و لتبسيط الأمور قم تجاهل أن واصف المقطع يخضع أيضا للتصفيح. إن ما يحدث فعلا هو أن مسجلا ما (مسجل قاعدة الواصف) يستخدم لتحديد موقع جدول صفحات مقطع الواصفات، الذي يشير بدوره غلى الصفحات التي تحوي مقطع الواصفات. حالما نحصل على واصف المقطع المطلوب، تسير عملية العنونة كما في الشكل الذي بعده.

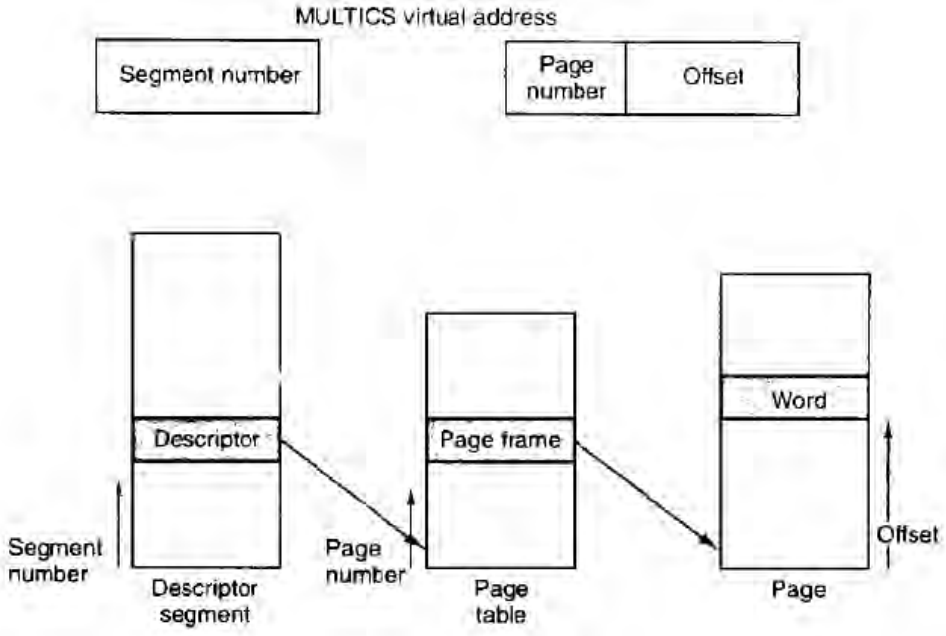


الشكل 25



الشكل26

من الممكن أن تظن بان هذه العملية لن تتم بالسرعة المطلوبة في حال تم تنفيذها من قبل نظام التشغيل. ويحوي عتاد الـ MULTICS مسجلا جانبيا TLB عالي السرعة بطول 16 كلمة والذي يمكن البحث في جميع سجلاته على التوازي عن مفتاح معين. وهو مبين في الشكل. عند ورود عنوان إلى المعالج، يقوم عتاد العنونة أولا باختبار إذا ما كان العنوان الافتراضي موجودا في الـ TLB. إذا كان كذلك، فإنه يحصل على رقم إطار الصفحة مباشرة من الـ TLB ويشكل العنوان الفعلي للكلمة المشار إليها دون الرجوع غلى مقطع الواصفات أو جدول الصفحات.



الشكل 27

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

الشكل 28

يحتفظ الـ TLB بعناوين الصفحات أل 16 الأكثر استخدما مؤخرا. البرامج التي تكون مجموعها العاملة أصغر من حجم الـ TLB ستصل إلى التوازن بوصول مجموعتها العاملة بأكملها في الـ TLB وستعمل بالتالي بكفاءة عالية. وإذا مل تكن الصفحة في TLB سيستخدم الواصف وجداول الصفحات فعليا لإيجاد عنوان إطار الصفحة ويحدث TLB بحيث يتضمن هذه الصفحة، وتلقى الصفحة الأقل استخدما مؤخرا (أي تحذف من الـ TLB) خارجه. يدل حقل

العمر على السجلات الأقل استخدما مؤخرا. إن سبب استخدام الـ TLB هو مقارنة رقم المقطع مع صفحة جميع السجلات على التوازي.

4 8 ـتقنية التضعيف عند محاولة الكتابةـ

Copy On Write

كما نعلم..فإن كل إنشاء لعلمية ابن يتم فيه نسخ الفضاء الذاكري (وذلك على أسس الأنظمة المبنية على تقنيات Unix) الأب للابن وذل لتحقيق التعاون بين العملية الأب والابن، ولكن في معظم الأحيان، فإن الابن لا يحتاج من هذه الذاكرة إلى القراءة فقط، زكذلك فإنه لا يحتاج إلا لبعض المتحولات من الأب وليس كلها، مما يعني هدرا في الذاكرة المحجوزة. ولنتخيل في حال تم إنشاء عشرة عمليات ابن لعملية أب، فهذا يعني عشرة نسخ من هذا الفضاء. وبالتالي فإن هذا الهدر سيتضاعف.

لقد اقترح لحل هذه المشكلة أن يتحول فضاء الذاكرة في حال إنشاء الابن إلى مجال ذاكري قابل للقراءة فقط، وذلك دون مضاعفته، وأن يكون هذا المكان مشتركا بين الأب والابن، ولكن بمجرد محاولة الكتابة على هذا المجال، سواءا من الأب أم الابن، فسيتم مضاعة (نسخ) المكان الذاكري المراد الكتابة عليه، وإعطاء النسخة الجديدة من هذا المكان للعملية التي طلبت الكتابة، وبالتالي نكون قد انتهينا من مشكلة هدر الذاكرة، وتسمى هذه التقنية بـ Copy On Write (COW).

4 9 ـقسم التنفيذ

سيتم في هذا القسم دراسة المكاتب التي تقوم بإدارة الذاكرة، وفي البداية سنمر على الملفات التي تحوي المكاتب التي تقوم بتنفيذ العمليات التي تتعلق بإدارة الذاكرة:

جدول8

اسم الملف	العمل الذي يقوم به هذا الملف
paging.c & paging.h	تحوي هذه الملفات على التوابع التي تقوم بإدارة عملية الحجز للصفحات الذاكرية، وعمليات تحريرها، إضافة إلى عملية الربط بين عناوين الذاكرة الافتراضية، وعناوين الذاكرة الخطية، وتحوي بالإضافة إلى ذلك على برنامج خدمة مقاطعة الـ page fault.
kernel_map.h	يحوي هذا الملف على الخريطة الذاكرية التي تحدد مكان النواة في الذاكرة، ومكان الذاكرة الافتراضية،...الخ.
kmalloc.c &kmalloc.h	ومن خلال هذه المكتبة سيتم الحجز لمساحات معينة من الذاكرة وذلك للعمليات على مستوى النواة، ومن هنا تأتي البادئة k على أنها kernel على خلاف الحال في الـ umalloc كما سنرى لاحقا، وهذه المكاتب هي مكاتب أعلى من مكاتب الـ paging والتعامل معها أبسط ومستقل تماما عن طبقة الـ hardware، ويضمن التغليف لعملية الحجز والتحرير للذاكرة، مما يسهل الأمر على المستدعي في مستوى النواة.
umalloc.c &umalloc.h	أما هذه المكتبة فتقوم بعملية الحجز الذاكري المغلق على مستوى المستخدم، أي ليس على مستوى النواة، وهذه المكاتب ستؤمن ما تؤمنه مكاتب الـ kmalloc وذلك للطلبات على مستوى المستخدم.
cow.h & cow.c	وتقوم هذه المكاتب بتحقيق تقنية الـ cow التي تم التكلم عنها في القسم النظري من هذا الفصل، وكذلك برنامج تخديم الاستثناء لهذه التقنية في حال حدوث محاولة للقراءة من أحد العمليات.
kpagerd.c	والذي يعمل كالـ Garbage collectorكما في لغات البرمجة، حيث يمر على العليمات التي في حالة zombie وتحرير الموارد التي تحتلها.

4 9 1- المكتبة kernel_map.h

كما ذكرنا فيما مضى، فإن هذه المكتبة ترسم حدود كل جزء من أجزاء الذاكرة، وذلك حتى لا يتم التعامل مع أي نوع إلا ضمن الحدود المخصصة له، مما يسهل عملية إدارة الحماية لكل منطقة من هذه المناطق (وذلك بإعطاء السماحيات المناسبة لكل جزء، فعلى سبيل المثال يجب أن يكون المكان المخصص بشفرة النواة قبلا للقراءة فحسب، ولا يجوز الكتابة عليه أبدا من أي كان).

حيث يتم فيها تحديد المكان الخاص في النواة، والحيز الفيزيائي من الذاكرة، والحيز الذي يتم استخدمه من أجل تنفيذ التحويل من العنوان الخطي إلى العنوان الفيزيائي (الذاكرة الافتراضية)، والحيز الخاص بمنطقة الكومة.

4 9 2- مكاتب التعامل مع الصفحات Paging

يتم في هذه المكتبة كما ذكرنا سابقا العمليات المسئولة عن إدارة الصفحات على مستوى العتاد، ففي بداية الملف paging.h يتم تعريف بعض الـ macros التي تسهل التعامل مع الـ pages والتي تكون بحجم 4-KB والتعامل مع تقنيتي الـ TLB التي تكلمنا عنها في القسم النظري.وفيها أيضا طريقة التعامل مع الـ Low memory Management، والتي اخترنا فيها طريقة المكدس.

4 9 2 1- تهيئة النظام للتعامل مع الصفحات paging

يتم تهيئة نظام التعامل مع الصفحات عن طريق التابع:

```
void __INIT__ init_paging()
```

وكما تلاحظ فإن هذا التابع هو من النوع init أي أنه يتم استخدامه في أول الإقلاع ومن ثم يتم تحرير المكان المحجوز لهذا التابع بعد انتهاء عملية التهيئة.ويجب استدعاء هذا التابع في التابع main وذلك قبل التوابع الأخرى، خاص توابع التهيئة المتعلقة بالعمليات، حيث يقوم هذا التابع بالمهام التالية:

في البداية يقوم هذا التابع بتحديد الحجم الفعلي للذاكرة من خلال أخذها من نظام الـ multibooting.

بعد ذلك يتم تحديد مجال الذاكرة المخصصة للتعامل مع الـ DMA بشكل ديناميكي.

تقوم بعدها هذه الإجرائية باستدعاء التابع init_free_frames() والذي يقم بعملية تهيئة المكدس الذي سنتحدث عنه بعد قليل.

تقوم هذه الإجرائية بفك كل الروابط بين العناوين الفيزيائية في الـ 4-MB الأولى من الذاكرة وبين العناوين الخطية في خارطة الذاكرة الوهمية.

يقوم بعدها بتحويل القسم الذاكري الذي يحوي شفرة النواة إلى مكان ذاكري قابل لقراءة فقط (أي يتم حجب إمكانية التعديل على الشفرة المتعلقة بالنواة كأسلوب من أساليب الأمن).

بعد ذلك يقوم بجعل القسم الذاكري من الشفرة في النواة مرئيا بالنسبة للمستخدم حتى يتمكن من استدعاء إجرائيان الخروج من النظام.

يتم بعدها تهيئة الأقسام الديناميكية من الذاكرة حتى تكون قادرة على التعامل مع الذاكرة الافتراضية.

تقوم بعدها بتفعيل أو إلغاء تفعيل خاصة الـ Global Pages والتي تحدثنا عنها في القسم النظري وذلك للقسم المتعلق بالنواة لنظام التشغيل، وذلك حسب نوع المعالج.

تقوم بتفريغ ذاكرة الـ TLB حتى يتم بدء العمل.

4 9 2 2 -التعامل مع مكدس الصفحات الغير محجوزة

كما ذكرنا فإن هناك عدة طرق للتعامل مع إدارة الذاكرة المنخفضة، ومنها طريقة اللوائح المترابطة وطريقة مصفوفة من البتات، وأخيرا طريقة المكدس، وهذه الطريقة الأخيرة هي الطريقة التي سنعتمدها لمعرفة الصفحات الحرة من الصفحات المحجوزة.في البداية يتم تهيئة هذا المكدس بالصفحات الفارغة، وذلك من خلال التابع:

```
void __INIT__ init_free_frames()
```

حيث يقوم بتهيئة هذا المكدس بعناوين الصفحات ابتداء من العنوان 16-MB الأولى إلى آخر الذاكرة الفيزيائية، وذلك لأن القسم الذي يسبق هذا القسم محجوز للنواة وبرامج المقاطعة المتعلقة بالشريحة BIOS. بعدها يتم وضع العنصر NULL دالا على نهاية المكدس.ففي حال تم طلب جميع الصفحات الفارغة تعاد القيمة NULL إلى كل من يطلب صفحة جديدة مجددا، وذلك بدلا من إعادة رقم عشوائي.أما طلب هذه الصفحات فيتم من خلال التابع:

```
__INLINE__ size_t pop_frame()
```

والذي يعيد حالة NULL في حال فشل عملية أخذ صفحات فارغة، أو العنوان الذاكري في حال تم نجاح هذه العملية.وعندما ينتهي المستخدم من المكان الذاكري الذي قام بحجزه، يقوم التابع التالي بإعادة الصفحة إلى المكدس، وذلك من خلال التابع:

```
__INLINE__ void push_frame( size_t p_addr )
```

4 9 2 3 -آلية الربط بين العناوين الخطية، والعناوين الفيزيائية

ويتم هذا من خلال عدة توابع، والتي تقوم بإنشاء الـ PTE كل ما دعت الحاجة لذلك، إضافة إلى حجز الصفحات بشكل فيزيائي، وما إلى هنالك من أمور.فعندما يطلب المستخدم حجز جزء من الذاكرة، يقوم التابع kmalloc بحجز مكان في الذاكرة الافتراضية لهذا التابع، ومن ثم إذا استخدمت هذه الذاكرة للقراءة أو أبي عملية أخرى، فإن هذه المهمة ستتعرض لاستثناء الـ page fault والذي من خلاله يتم استدعاء التابع map_page الذي سندرسه في هذه الفقرة، وذلك حتى يتم عملية الحجز بشكل ديناميكي وعند الحاجة، وتسمى هذه العملية بالحجز الذاكري عند الحاجة (وليس عند الطلب)، memory on demande.

4 9 3 -التابع map_page

ويكون شكل هذا التابع على النحو التالي:

```
bool map_page( size_t vir_addr, size_t phys_addr, uint16_t attribs );
```

حيث يكون العنصر الأول والثاني هما العناوين الخطية والفيزيائية على الترتيب، بينما يكون الوسيط الثالث هو الخواص المتعلقة بالـ PTE المربوط بالعنوان الفيزيائي.إن ما يقوم به التابع map_page هو الربط بين العناوين الفيزيائية والأخرى الخطية، بحيث تكون مفهومة للمعالج Intel، وإذا تطلبت الحاجة لمساحات إضافية من أجل الـ PTE يتم حجز الصفحات من خلال التابع pop_frame والذي درسناه سابقا.أما التابع unmap_page فهو عبارة عن تابع يقوم بعكس المهمة التي يقوم بها التابع map_page. كذلك التابع del_page يقوم بنفس العمل الذي يقوم به التابع unmap_page ولكن يقوم بدوره بحذف المكان الفيزيائي للصفحة أيضا، وإعادته إلى المكدس.

4 9 4 -تابع معالج الاستثناءات Page falut handler

يتم معالجة حالة الاستثناء في التابع page_fault_handler والذي يقوم برنامج معالجة الأخطاء في الملف exception.c باستدعائه كل ما أصبح لدي page fault exception.يقوم هذا التابع بعملية الفحص التي من خلالها سيتم حجز مكان ذاكري للطلب الخاطئ الذي تم.إن ما يحصل حقيقة في هذا العمل، هو أن طلب حجز ذاكرة بحجم (س) على سبيل المثال سيتم من خلال التابع kmalloc والذي يقوم بحجز ذاكرة افتراضية بهذا الحجم، ومن ثم في حال تم استخدام هذه الذاكرة على أنها محجوزة، يقوم التابع page_fault_handler بحجز هذه الذاكرة بشكل فيزيائي.

حيث يستدعي التابع pop_frame من أجل حجز مكان فيزيائي فارغ في الصفحة، من ثم التابع map_page والذي يربط بين الذاكرة الافتراضية التي أصبح عندها الخطأ والذاكرة الفيزيائية التي حصلنا عليها من التابع pop_page.أما التابع page_fault_cow_handler فهو تابع لمعالجة الأخطاء التي تنتج من التعامل مع الصفحات المشتركة بين العليمات على أنها قابلة للكتابة، ومن هنا كان واجب هذا التابع أن يفك التضعيف من خلال إنشاء صفحة جديدة ونسخ المعلومات من النسخة المطلوبة إلى النسخة الجديدة، ثم تغيير عناوين الذاكرة الافتراضية، والتعامل مع هذه الصفحة الجديدة على أنها قابلة للكتابة.

4 9 5 -حجز الذاكرة على مستوى النواة kmalloc

يتم طلب حجز الذاكرة من توابع على مستوى النواة من خلال التابع:

```
void *kmalloc( size_t size, int mflag )
```

والذي يقوم بحجز الذاكرة لمن يطلبها وذلك على مستوى الذاكرة الافتراضية كما تم شرح ذلك سابقا.وتعتمد هذه المكتبة البنية MEM_BLOCK للتعبير عن المساحة الذاكرية، حيث تكون بنيتها على النحو التالي:

```
typedef struct MEM_BLOCK
{
dword magic,
        flags,
        size,
        owner;
} mem_block_t;
```

حيث يعبر الحقل owner عن المالك لهذه الحجرة الذاكرية، أي الذي قام بطلبها. أما الحقل flags فيبين فيما إذا كانت هذه الكتلة حرة أم لا. بينا يعبر size عن الحجم الذاكري لهذه البنية.وهناك تابع يتم استدعاءه في بداية عمل النظام، وذلك من أجل التهيئة، وهذا التابع هو:

```
void __INIT__ kmalloc_init( void )
```

حيث يقوم بحجز mem_block_t من الذاكرة حجمه بحجم 256-MB وهذا mem_block_t هو المساحة الحرة التي سيتم الحجز فيها، وذلك عن طريق أخذ مساحات من هذه الذاكرة، وتقسيم (تقليل حجم هذه الكتلة بالمقدار المطلوب، ومن ثم إنشاء كتلة جديدة بهذا الحجم للتعامل معها على أنها المكان الذاكري المحجوز) هذا الـ mem_block_t للحصول على الذاكرة المطلوبة.

أما التابع deallocate والذي يقوم بتحرير الذاكرة الفيزيائية (والتي تقابل الذاكرة الافتراضية في الكتلة) المحجوزة. بينما يقوم التابع kfree بتحرير الذاكرة الافتراضية، ومن ثم يستدعي التابع deallocate لتحرير الذاكرة الفيزيائية بعد ذلك، أي أن على المستخدمين للتابع kmalloc من اجل حجز مكان ذاكري فارغ استخدام التابع kfree عند الانتهاء من الحجرة الذاكرية التي تم استخدامها.

أما التابع:

```
void *kmemalign( size_t alignment, size_t size, int flags )
```

فهو يقوم بالحجز الذاكري على اعتبار أن الذاكرة التي نريد الحجز فيها هي ذاكرة مقسمة إلى أحجام من نوع alignment علما انه يتم حجز الذاكرة في أول القسم. ويستخدم هذا التابع من أجل حجز ذاكرة في أول الـ page من اجل تخزين الـ TSS حتى يتمكن من إجراء عملية الـ switching كما ذكرنا سابقا، عندها يكون الوسيط alignment مساويا لـ PAGE_SIZE.والتابع kcalloc هو نفسه التابع kmalloc إلا أنه يحجز عدد من الحجرات(الوسيط الأول) من حجم معين (الوسيط الثاني) على شكل مصفوفة، وذلك بدلا من أن يمرر الحجم الكلي المراد في وسيط واحد، وهو على هذا الشكل:

```
void *kcalloc( size_t num, size_t size, int flags );
```

6 9 4 -مكتبة الـ umalloc

يقوم هذا التابع بالتعامل مع ذاكرة الكومة المتعلقة بفضاء ذاكرة المستخدم، وذلك بدلا من التعامل مع المكدس في مستوى النواة. وهذه هي المكتبة التي سيتم استدعاؤها في حال تم طلب التعامل مع موارد الذاكرة في مستوى المستخدم.

تحوي هذه المكتبة تابع umalloc_init لتهيئة ذاكرة الكومة في الفضاء المخصص للمستخدم، وذلك في بداية إنشاء العملية، حيث علينا أن نحذر من الخطأ في استدعاء التابع من

قبل thread لأن الخيط لا يملك مكان كومة ذاكرية خاصة به، بل على العكس من ذلك فهو يقوم باستخدام والمشاركة على نفس المكان الذاكري في العملية، ويكون هذا التابع على الشكل التالي:

```
void umalloc_init(task_t *t, size_t heap_start, size_t heap_size)
```

حيث كما هو الحال في مكاتب الـ kmalloc فإن هذه المكتبة تستخدم بنية تدل على الكتلة المحجوزة، والتي تكون على الشكل التالي:

```
typedef struct UMEM_BLOCK
{
        uint32_t        magic,
                        flags,
                        size,
                        owner;
} umem_block_t;
```

أما التابع الذي do_umalloc الذي يقوم بحجز المكان الذاكري من هذه الكومة وذلك من خلال التمرير للحجم الذي يراد حجزه على الكومة:

```
void *do_umalloc( task_t *ut, size_t size )
```

ويراعي هذا التابع أن الوسيط ut الذي تم تمريره إلى التابع من الممكن أن يكون thread حيث يتم فحص طبيعة هذا التابع وتمرير العملية الأب له في حال كان مجرد thread، وذلك لأن هذه العميلة تحتاج إلى العملية التي سيتم الحجز في فضاءها الذاكري. حيث يتم استخدام الخوارزمية best free التي تكلمنا عنها في القسم النظري.

يقوم التابع deallocate بعملية التحرير الفيزيائي للذاكرة (كما هو الحال في التابع deallocate في المكتبة kmalloc.h). أما التابع do_ufree فيقوم بنفس التعليمات التي يقوم بها التابع kfree في المكتبة kmalloc.

لا يتم التعامل مع التابعين do_ufree والتابع do_umalloc مباشرة، وإنما يتم التعامل معهما من خلال تابعين آخرين يكفلان التكاملية في عملية الاستدعاء المتبعة، ألا وهما:

```
void ufree( void *ptr );
void *umalloc( size_t size );
```

4 9 7- مكتبة الـ COW

يتم في هذه المكتبة تنفيذ التقنية Copy On Write والتي من خلالها سيتم التعامل مع التضعيف وفك التضعيف في الصفحات المشتركة بين العليمات الأب والعمليات الابن. علما أننا تكلمنا عن معالجة الاستثناءات التي ترفع في حال حدوث استثناء لمحاولة الكتابة على هذه الصفحات في التابع page_fault_cow_handler وذلك في الملف paging.c.

يتم معرفة الصفحات التي تتعامل مع التقنية cow وذلك من خلال بنية أرتال يكون العنصر فيها على الشكل التالي:

```
typedef struct cow {
        size_t addr;
        int count;
} cow_t;
```

حيث يعبر الحقل count عن العدد من العمليات الذي من خلاله يتم الدخول إلى هذه الصفحة، ويزداد هذا العدد في كل عملية لإنشاء عملية ابن جديد، ويتم إنقاص هذا العدد في كل مرة تحاول احد العمليات الكتابة على هذه الصفحة، عندها سيتم نسخ هذه الصفحة إلى صفحة فارغة جديدة عن الصفحة التي تريد العلمية الكتابة عليها وتغيير العناوين الافتراضية بحيث تؤشر إلى هذا المكان الفيزيائي الجديد.

يقوم التابع set_shared_page بالبحث عن العنصر الممرر له كوسيط في الرتل في حال وجده يتم زيادة العداد count بالقيمة 1، فإن لم يجده سيضيف هذا العنصر إلى الرتل، ويكون عندها قيمة العداد مساوية للواحد.

```
int set_shared_page(size_t addr)
```

أما التابع unset_shared_page فيقوم بالبحث عن العنصر الممرر له كوسيط، وإنقاص العداد count لهذا العنصر بالقيمة واحد، فإذا كانت القيمة مساوية للصفر سيتم حذف هذا العنصر من العملية وذلك لأنه لم تعد هناك عمليات تشترك على هذه الصفحة، أي أن الصفحة لم تعد تخضع للعنوان cow.

4 9 8 -مكتبة الـ kpagerd

تعمل هذه المكتبة (كما ذكرنا سابقا) كما يعمل الـ Garbage collector في لغات البرمجة، حيث تكون هناك عملية بشكل مستمر طوال حياة نظام التشغيل تسمى الـ kpagerd، وتكون هذه العملية نائمة ما لم يتم إيقاظها من أحد العمليات. تقوم هذه العملية بالتجول في الرتل zombie_queue وذلك من أجل تحرير الموارد الذاكرية المتعلقة بهذه العملية، ثم إجراء القتل لهذه العملية وذلك بحذف البنى المتعلقة بوصف حالة العملية، تتجسد هذه العلمية بالتنفيذ التابع التالي:

```
void kpagerd(int argc, char **argv)
```

وفي حال لم تجد هذه العملية أي عملية أخرى في رتل الـ zombie_queue فإنها تقوم بوضع نفسها في الـ wait_queue حتى يتم إيقاظها من جديد مرة أخرى.

4 10 -المراجع :

السنة	الإصدار	المؤلف	اسم المرجع
October 2000 ISBN: 0-596-00002-2,	First Edition	Daniel P. Bovet Marco Cesati	Understanding the Linux Kernel / chapter 2&6&7
1992	second Edition	Andrew Tanenbaum	Modern Operating Systems / Chapter 4 – Memory Managment
1992	second Edition	Andrew Tanenbaum	Operating Systems Design and Implementation/ Chapter 4 – Memory Managment
2005	Order Numbers 253666 and 253667	Intel	The IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference.
2005	Order Number: 253668-016	Intel	IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide/ Chapter 2&3
2002		Tim Robinson	(Memory Management 1) http://www.osdever.net
2002		Tim Robinson	(Memory Management 2) http://www.osdever.net
2002		Mike Rieker	(Pagetables) http://www.osdever.net
1999	(Dynamic Memory Allocation)http://www.osdever.net -		
(Algorithms and Tips for Memory Management) http://www.mega-tokyo.com/osfaq2/index.php/Algorithms%20and%20Tips%20for%20Memory%20Management-			

5 -إدارة العمليات

Process Management

مُقَدِّمَة

تستطيع جميع الحواسب الجديدة بتنفيذ أكثر من مهمة في نفس الوقت، فمن الممكن للمستخدم أن يشغل برنامج ويتفاعل معه، بينما يقوم المعالج في نفس اللحظات بقراءة من القرص والكتابة عليه. وعمليا فإن المعالج لا يستطيع أن يقوم بأكثر من عملية واحدة في لحظة زمنية واحدة، إلا أن المعالج يقوم في الثانية الواحدة بتنفيذ أكثر من برنامج واحد والعمل عليه، وهذا ما يسمى في بعض المراجع بالتوازي الظاهري (Pseudo Parallelism) وذلك حتى يتم التمييز بينها وبين التوازي الحقيقي في الأنظمة المتعددة المعالجات Multiprocessors.

ومن الصعب على المستخدمين تطوير الأنظمة بحيث تعمل على التفرع، لذلك سعى مطوروها أنظمة التشغيل لتحقيق هذا العمل وذلك بالانتقال بأنظمة التشغيل من تنفيذ مهام تسلسلية معينة، إلى تنفيذها على التفرع، وذلك من خلال تجريد مفهوم العمل على التسلسل وفق مفهوم سمي بالعمليات التسلسلية أو ما يسمى الـ Process.

في هذا التجريد للعملية يتم الانتقال بشكل أسهل لمفهوم الـ Multiprocessing. حيث يتم تنفيذ عملية واحدة في كل لحظة زمنية، وهذه العملية ليست إلا تنفيذ تسلسلي لشفرات معينة، ثم بعد ذلك يتم الانتقال لتنفيذ عملية أخرى، وهذا يتم من خلال حفظ الحالة القديمة للعملية (من المكان الذي وصلت إليه في تنفيذ الشفرة، إلى تخزين حالات المسجلات...) والانتقال لتحميل عملية قديمة ثم بعد ذلك تنفيذها، ويتم ذلك من خلال عملية خاصة تسمى المجدول Scheduler.كما نلاحظ فإن هناك فرق بين البرنامج والعملية، وهذا الفرق بسيط ولكنه مهم، فالبرنامج هو الخطوات المحددة لإنجاز مهمة معينة، أما العملية فهي القيام بتنفيذ هذه الخطوات.

إن الفكرة الأساسية هي أن العملية عبارة عن نشاط أو فعالية من نوع معين، تملك هذه العملية برنامجا ودخلا وخرجا وحالة. ويمكن مشاركة معالج واحد لأكثر من عملية مع وجود خوارزميات جدولة (Scheduling Algorithm) تستخدم لتحديد زمن التوقف عن خدمة عملية ما والبدء بتنفيذ خدمة عملية أخرى.

5 1 -إنشاء عملية

تحتاج أنظمة التشغيل لطريقة ما للتأكد من أن كل العمليات التي تحتاج إليها ضرورية وموجودة. وفي الأنظمة البسيطة التي لا تستخدم غير برنامج وحيد، على سبيل المثال كما في الـ Microwave، فمن الممكن لهذه العمليات أن تنشأ بشكل مباشر أثناء أقلاع نظام التشغيل. أما في الأنظمة ذات الأغراض العامة فيجب تحديد آلية لإنشاء وإنهاء العملية وذلك حسب الحاجة.هناك أربعة أحداث أساسية تتم لإنشاء العملية إلا أنها تختلف من نظام إلى آخر من حيث التفاصيل [1]:

تهيئة النظام.
تنفيذ استدعاء للنظام لإنشاء عملية من قبل عملية قيد التنفيذ.
طلب من المستخدم لإنشاء عملية جديدة.
بدء عمل دفعي.

عند إقلاع نظام التشغيل يتم عادة إنشاء عدة عمليات، يمكن أن يكون بعضها عمليات أمامية، وهي عمليات يتم التفاعل معها من قبل المستخدم، وعمليات أخرى تتم في الخلف، وتسمى هذه العمليات بالعفاريت إن صح التعريب (Daemons) أو البرامج الخفية، فمثلا عندما أقوم بالقراءة من ملف على القرص أو عندما أستقبل التعليمات على لوحة المفاتيح فإن هذه المهمات تنفذ من خلال عمليات لا تظهر للمستخدم، ولذلك سميت بهذا الاسم، وهي تبدأ العمل في حال حدوث مقاطعة ما، أو طلب معين.

يتم اكتشاف هذه العمليات في نظام Unix والأنظمة الموافقة له من خلال التعليمة ps (وهذه التعليمة موجودة في نظامنا)والتي تقوم بطباعة كل العمليات الموجودة الآن على النظام، أما في Windows فإن هذه العمليات تظهر من خلال التعليمة Alt+Ctrl+Del.وفي كل نظام تشغيل هناك تعليمات يمكن طلبها من النظام حتى يتم توليد عملية جديدة، يوجد في نظام Unix والأنظمة الموافقة له تعليمة واحدة لهذه المهمة، وهذه التعليمة المعروفة جدا بين مبرمجي Unix هي التعليمة fork، حيث ينشئ هذا الاستدعاء عملية طبق الأصل عن العملية التي قامت باستدعائها، تملك في هذه الحالة العملية الجديدة والعملية المستدعية نفس الصورة على الذاكرة، وكذلك نفس متحولات البيئة ونفس الملفات المفتوحة، ثم بعد ذلك تقوم هذه العملية تنفيذ برنامج معين حسب المطلوب عن طريق الأمر execve.

أما في نظام التشغيل Windows فيوجد استدعاء تابع Win32 وحيد واسمه CreateProcess حيث تعالج كلم ن إنشاء العملية وتنفيذ البرنامج الصحيح إلى العملية الجديدة، لهذا الاستدعاء عشرة وسائط تتضمن مكان تنفيذ البرنامج، وبتات تحدد الشفرة المطلوبة من أجل تحديد الملفات المفتوحة، وكذلك مستوى الأمن المطلوب.

وبعد إنشاء العملية الابن في كل من الأنظمة السابقة يكون لكل من الأب والابن المكان الذاكري الخاص به، فإذا تم التغيير في مكان ذاكري خاص بالأب مثلا فهذا لا يؤثر على المكان الذاكري الخاص بالابن. وفي نظام الـ Unix يكون فضاء العناوين الخاص بالابن هو نسخة طبق الأصل عنه في الأب، ولا يوجد ذاكرة مشتركة بينهما قابلة للكتابة بشكل مشترك (حيث في بعض الأنظمة Unix لا يتم نسخ المكان الذاكري المتعلق في الشفرة التنفيذية للبرنامج).أما في Windowsفيكون هناك فرق واضح في فضاء الذاكرة بين الأب والابن، فهما مختلفان تماما.

2 5 -إنهاء العملية

بعد إنشاء عملية، تبدأ العملية بتنفيذ العمل المطلوب منها، لكن لا شيء يدوم للأبد حتى العمليات، ويكون ذلك عادة لأحد الأسباب التالية[1]:

- خروج طبيعي (طوعي).
- خروج بخطأ، (طوعي).
- خطأ قاتل (قسري).

إنهاء من قبل عملية أخرى (قسري).

تنتهي معظم العمليات لأنها تكون قد أنجزت عملها. ويعبر عن هذا العمل (انتهاء العملية) باستدعاء التابع exit في الأنظمة المتوافقة مع الـ Unix، والاستدعاء ExitProcess في Windows أما السبب الثاني لإنهاء عملية هو اكتشاف عملية لخطا ما أثناء التنفيذ، مثلا خطأ في المتحولات الممررة في بداية تشغيل البرنامج.

السبب الثالث لإنهاء العملية هو حدوث خطأ من قبل العملية، ويكون ذلك غالبا بسبب خطأ في البرنامج. من الأمثلة على ذلك تنفيذ تعليمة غير شرعية، أو من خلال طلب لعنوان ذاكري غير موجود، مما يولد خطأ Page Fault، أ التقسيم على صفر. وتستطيع العمليات في بعض الأنظمة (مثل الـ Unix) أنها ستقوم بمعالجة بعض الأخطاء المعينة بنفسها. وبالتالي فإنه يتم إخبار العملية عن هذا الخطأ الحاصل (عن طريق طلب مقاطعة) بدلا من إنهائها عند حدوث أحد هذه الأخطاء.

السبب الرابع، هو تنفيذ استدعاء من عملية أخرى يطلب من نظام التشغيل إنهاء عملية معينة.وكما هو معرف فإن هذا الاستدعاء معروف في Unix على أنه التابع kill. أما في Windows وفي استدعاءات الـ Win32 فهناك تابع يسمى الـ Terminate Process. ومن الطبيعي أن يكون للعملية التي تقوم بتنفيذ هذا الاستدعاء الصلاحيات اللازمة للقيام بهذا العمل.من المنطقي القول أن قتل أحد هذه العمليات سيؤدي إلى قتل جميع الأولاد، ولكن.كلا النظامين Windows و Unix لا يعملان بهذه الطريقة.

5 3 ـ هرمية العمليات

عندما تقوم عملية ما باستدعاء إنشاء عملية، فإن معظم الأنظمة تبقي على علاقة بين الأب والابن، وهذه العلاقة تكون مختلفة من نظام لآخر.وتستطيع العملية الابن أن تنشئ عملية ابن أخرى لها، مما يلاحظ أن البيئة المشكلة هي بنية هرمية.

وكمثال عن الهرمية في Unix والأنظمة المتوافقة معه، يقوم النظام أولا بالإقلاع، تكون هناك عملية خاصة مبنية في بداية الإقلاع، وتسمى هذه العملية بالـ init Process وهذه العملية هي العملية الأولى فعليا في النظام، حيث تقرأ هذه العملية عدد ألل البرامج الفرعية المحيطة بالنظام وذلك من خلال ملف خاص، ثم بعد ذلك تقوم بإنشاء عملية خاصة لكل جهاز طرفي محيط بالحاسب. وتنتظر هذه العملية لتسجيل أحد المستخدمين، وفي حال تم التسجيل بشكل سليم، تقوم هذه العملية بإنشاء عملية من أجل تنفيذ أوامر الاستقبال من محث الأوامر. وهذه العملية المولودة أخيرا قد تقوم هي بدورها بإنشاء عمليات أخرى، وهكذا. أي أن العمليات في نظام الـ Unix هي عبارة عن شجرة تنتهي إلى الأب المشترك لهذه العمليات، وهو العملية init.

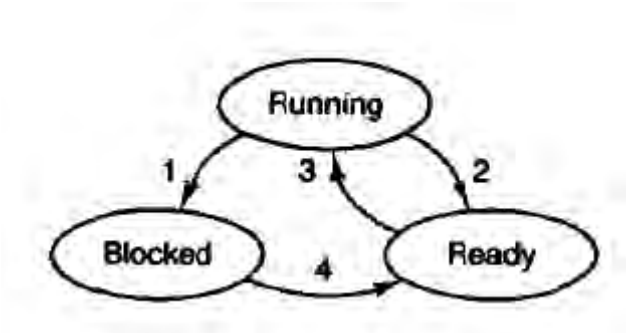
على العكس من ذلك لا يملك Windows مفهوم الهرمية في العمليات، فجميع العمليات عنده متساوية. والمكان الوحيد الذي يكون فيه شيء يشبه الهرمية في العمليات هو عندما تنشئ عملية، حيث يعطي الأب قيمة خاصة (تسمى المقبض Handle) يستطيع من خلالها التحم بالابن. إلا أن الأب يستطيع ببساطة إعطاء هذه القيمة إلى عملية أخرى مما يؤدي إلى إلغاء البنية الهرمية. أما العمليات في Unix فهي لا تستطيع إنكار أبنائها.

5 4 ـ حالات العملية

كما هو معروف، فإن هناك حالات عدة للعمليات تمر بها أثناء تنفيذها برنامج معين، وهذه الحالات بشكل عام هي كالتالي:

- فعالة (تستخدم المعالج في هذه اللحظة بشكل فعلي أي أنها العملية التي يقوم المعالج بتنفيذها).
- جاهزة (أي أنها قابلة للعمل، لكنها متوقفة مؤقتا للسماح بتشغيل مهمة أخرى).

متوقفة (غير قادرة على العمل حتى حدوث حدث خارجي).



الشكل29

أما في الأنظمة المتوافقة مع الـ Unix فمن الممكن أن نضيف حالة أخرى وهي حالة العملية الميتة، أو ما يسمى Zombie Process، حيث أن العمليات التي تم إيقافها بشكل مشروع

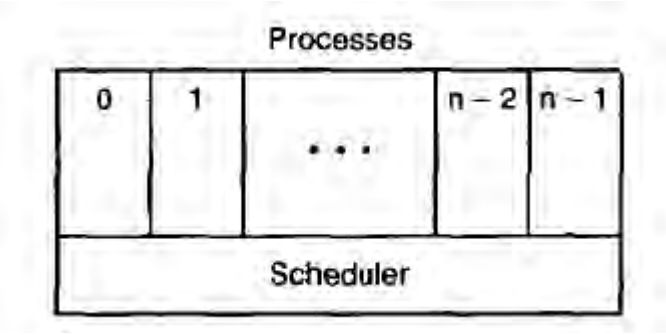
تبقى كما هي حتى تتم قراءة النتائج التي حصلت عليها من العملية الأب، ومن ثم يتم تدمير الموارد الخاصة التي فتحت لهذه العملية. وتتم هذه العملية المؤقتة من التخزين وذلك للحصول على المعلومات التي توصلت إليها هذه العملية من العمليات الأخرى، وذلك تجنبا من موتها بشكل مباشر وضياع النتائج التي توصلت إليها.

وكما نلاحظ في حالتي الفعالية والجاهزة، فإن هذه الحالتين متماثلتان تقريبا، حيث كلا العمليتين جاهزتين للعمل ولكن الأمر في الحالة الثانية هو عدم وجود معالج لتنفيذ هذه العملية على الرغم من أنها قابلة للعمل. الحالة الثالثة مختلفة عنهما في أن العملية غير قابلة للعمل، حتى ولو كان المعالج لا يقوم بتنفيذ أي شيء.

يحدث الانتقال 1 عندما تكتشف هذه العملية أنها لا تستطيع المتابعة، أما الانتقالات 2 & 3 فهما انتقالان يحدثان من خلال مجدول المهام، وهو جزء من نظام التشغيل دون أن تدري العمليات بوجوده، ودون أن تدري بحدوث هذه الانتقالات، إذ يجب أن تكون هذه الانتقالات مخفية عنها.

يحصل الانتقال 3 عندما تنتهي حصة العملية التي تنفذ الآن، ويحين الوقت لتنفيذ عملية جديدة حيث يقوم مجدول المهام من نزع العملية القديمة من المعالج وتسليمه لمهمة أخرى. وتعتبر عملية جدولة المهام أي تحديد العملية التي يجب أن تعمل ومتى ولكم من الوقت أمور هامة جدا في مسألة تصميم نظام التشغيل والعمل الذي سيخصص له هذا النظام.

ويحدث الانتقال 4 عندما يحصل الحدث الخارجي الذي كانت تنتظره العليمة. وإذا لم يكن هناك عملية فعالة في الوقت الحالي على المعالج، فإن الانتقال 3 سيتم، وإلا فإن العملية ستنتظر في حالة الجاهزية فترة من الزمن ريثما يتوفر المعالج ويأتي دورها.



الشكل30

سيكون شكل النظام كما هو مبين في الشكل الآن، حيث يكون المستوى الأدنى لنظام التشغيل هنا هو المجدول Scheduler، مع وجود عمليات متنوعة فوقه، علما أن هذا المجدول يجب أن لا يحوي الكثير من الشفرة، ويجب أن يكون تنفيذه سريع نسبياً.

5 5 ـ تحقيق العملية

يجب أن تحوي العملية على عدة متحولات تعبر عن حالة العملية والمكان الذي وصلت له في الذاكرة، إضافة إلى الموارد التي حجزتها لها، وتوضع هذه المتحولات فيما يسمى بجدول العملية Process Table، وتحفظ هذه المتحولات في مكان ما معرف بالنسبة للمجدول، وذلك لتخزين هذه المتحولات قبل انتزاع العملية من المعالج، وإعادة هذه المتحولات من قبل المجدول بعد إعادة تسليم المعالج لهذه العملية.

يبين الحقل التالي بعض الحقول الأساسية في نظام نموذجي. تتعلق الحقول في العمود الأول بإدارة العليمات، ويتعلق الحقلان الثاني والثالث بإدراة موارد الذاكرة والملفات لهذه العملية. ومن الملاحظ أن الحقول الموضوعة في إدارة العليمات لها علاقة كبيرة في النظام المدروس وطبيعة عمله، ولكن هذا الحقل يعطي تصورا عاما عن المعلومات المهمة في إدارة العلميات.وفي القسم التنفيذ لهذا الفصل سنقوم بدراسة الحقول المتعلقة بالعملية بنوع من التفصيل، وبإسقاط مباشر على نظامنا.

Process management	Memory management	File management
Registers	Pointer to text segment	UMASK mask
Program counter	Pointer to data segment	Root directory
Program status word	Pointer to bss segment	Working directory
Stack pointer	Exit status	File descriptors
Process state	Signal status	Effective uid
Time when process started	Process id	Effective gid
CPU time used	Parent process	System call parameters
Children's CPU time	Process group	Various flag bits
Time of next alarm	Real uid	
Message queue pointers	Effective	
Pending signal bits	Real gid	
Process id	Effective gid	
Various flag bits	Bit maps for signals	
	Various flag bits	

الشكل 31

5 6 -خيوط التنفيذ (Threads)

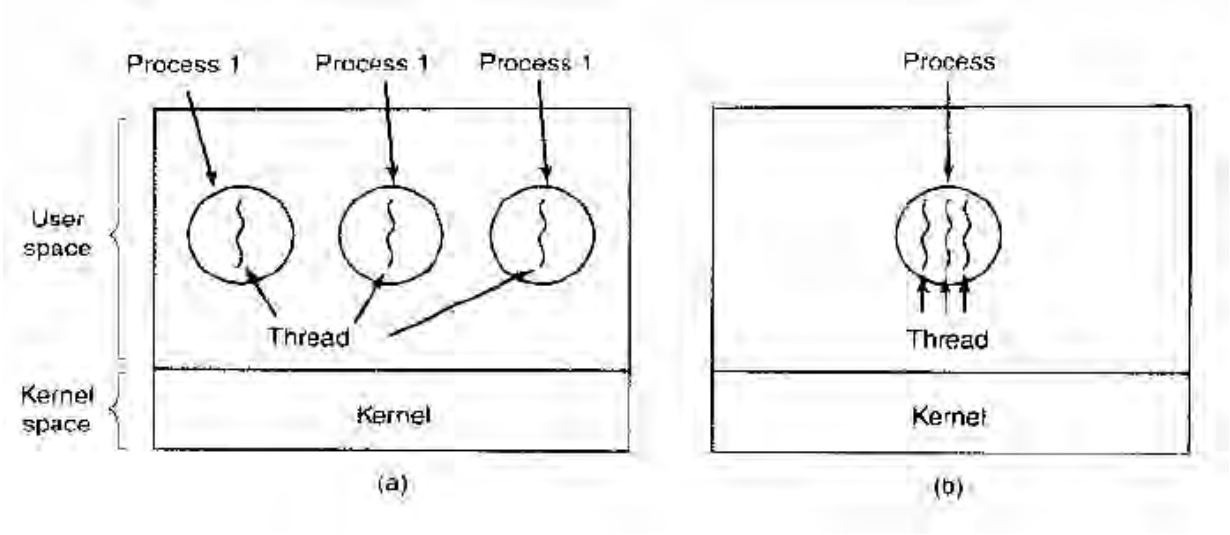
في أنظمة التشغيل التقليدية، كل عملية هلا خيط تنفيذ وفضاء عناوين وحيدين، وفي الواقع هذا هو تعريف العملية. ولكن الواقع أثبت انه من الأفضل أن تعمل أكثر من عملية على فضاء واحد من الذاكرة، وتنفذ هذه العمليات على التوازي وكأنها عمليات منفصلة إلا أنها تعمل في مجال ذاكري مشترك. سندرس في هذا القسم ميزات الخيوط التنفيذية، وأنواعها، وإيجابيات وسلبيات كل منها.

1 6 5 -توصيف خيوط التنفيذ

يشبه الـ Thread أو خيط التنفيذ إلى حد كبير العملية Process، فل Thread عداد برنامج سيجدد التعليمة التي ستنفذ لاحقا، وكذلك يملك مسجلات تحوي على متحولات العمل الحالي. وله مكدس يحوي خطوات التنفيذ السابقة، حيث يحوي إطارا واحدا لكل إجراء تم استدعاءه ولم يعد بعد. ولكن على الرغم من أن الخيط يجب أن ينفذ في عملية إلا أن العملية والخيط مفهومان مختلفان تماما ويمكن التعامل معهما بشكل منفصل. حيث تستخدم العملية من أجل تجميع الموارد (المخصصة لبرنامج معين على سبيل المثال)، بينما تعرف الخيوط على أنها تلك التي تجدول من أجل التنفيذ داخل المعالج[1].

إن ما تضيفه الخيوط هي السماح بتنفيذ إجراءات متعددة ضمن نفس بيئة العملية، وبدرجة كبيرة من الاستقلالية عن بعضها وتعمل على التوازي في جهاز واحد. إن وجود عدة خيوط تعمل بشكل مستقل ضمن العملية، يشبه عدة عمليات تقوم بالعمل بشكل مستقل عن بعضها في حاسب واحد. في الحالة الأولى تتشارك الخيوط على المجالات الذاكرية والملفات المفتوحة والموارد الأخرى، في حين تتشارك العمليات على الذاكرة الفيزيائية والأقراص والطابعات وموارد أخرى، وبما أن الخيوط تملك الكثير من خواص العمليات، لذلك يطلق عليها في بعض الأحيان العمليات الخفيفة (Lightweight Process)، علما أن بعض المراجع تميز بين العملية الخفيفة، وخيط التنفيذ. وكذلك يستخدم مفهوم الخيوط المتعددة(Multithreading)، حتى يتم الدلالة على أن هذا النظام يمكّن من عمل عدة خيوط على عدة عمليات.

نرى في الشكل اللاحق ثلاثة عليمات تقليدية تعمل بشكل مستقل على حاسب واحد وذلك في القسم (a)، بينما في القسم (b) هناك ثلاثة خيوط تعمل بشكل مستقل على عملية واحدة، إن الفرق بين الأولى والثانية، هي أن الخيوط تتشارك على نفس موارد العملية، وهذا لا يتم على مستوى العلميات.



الشكل 32

عندما تعمل عدة خيوط في عملية واحدة فإنها تعمل على التناوب، أي كما هو الحال عندما تعمل العمليات المتعددة على حاسب واحد، حيث يعطي النظام انطباعا على أن هناك أكثر من عملية تعمل على التسلسل وبشكل متواز من خلال التبديل جيئة وذهابا بين العمليات المتعددة. وكذلك فإن الخيوط تعمل بنفس الطرقة التي تعمل بها العمليات المتعددة على حاسب واحد.حيث يتم التبادل بين الخيوط على العملية الواحدة، وبالتالي تبدو وكأنها تعمل على التوازي، ولكن تبدو وكأنها تعمل على معالج أبطأ مما هو عليه حقيقة، فإذا كان لدينا على سبيل المثال ثلاثة خيوط على عملية واحدة، فإن هذه الخيوط تبدو وكأنها تعمل على معالج أبطأ إلى درجة الثلث مما هو عليه في الحقيقة.

لا تكون الخيوط مستقلة تماما في العمل كما هو عليه الحال في العلميات، فجميع الخيوط لها الفضاء من العناوين نفسه، وهذا يعني أنها تتشارك أيضا بالمتحولات العامة أي أنه لا

توجد أي حماية بين هذه الخيوط لأن ذلك مستحيل أولا ويحب ألا يكون ضروريا ثانيا. فعلى العكس من العمليات التي تأتي من مستخدمين مختلفين وتكون في الغالب معادية لبعضها البعض، فإن الخيوط تنشأ لعملية واحدة لها نفس المستخدم والي قام بإنشائها للتعاون وليس من أجل أن تتقاتل.

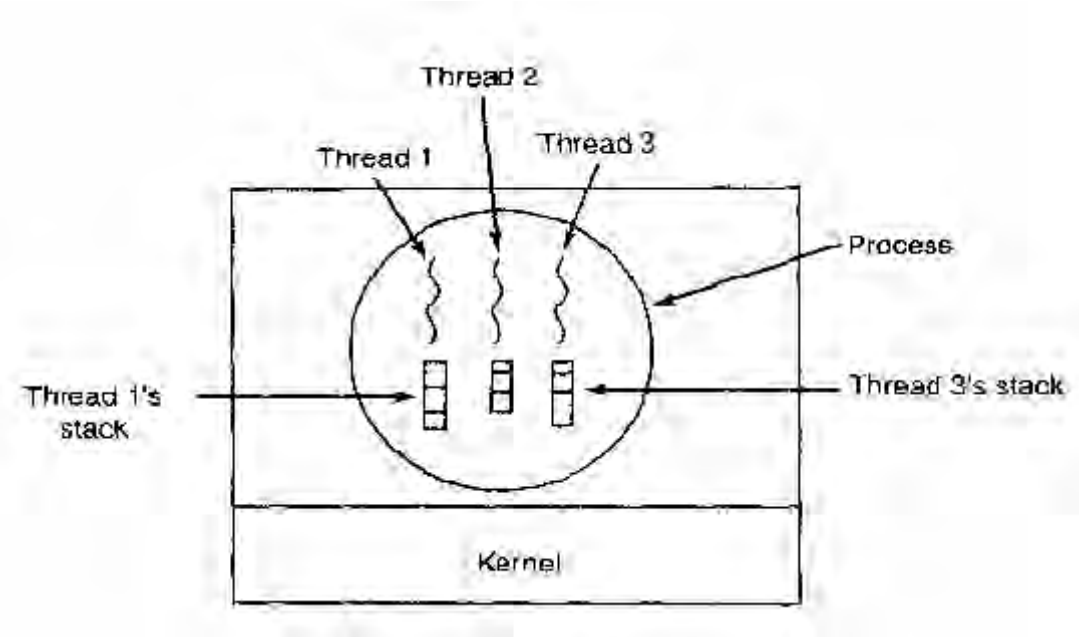
وهذا موضح في الشكل والذي يبين الموارد المشتركة بين الخيوط، فعناصر العمود الأول هي خوص العمليات وليست خواص الخيوط، فمثلا إذ قام أحد الخيوط بفتح ملف فإن هذا الملف سيكون مرئيا من قبل كل الخيوط الأخرى على مستوى العملية، وهذا منطقي لأن العملية هي المسئولة عن إدارة الموارد وليس الخيط. فإذا أصبح لكل خيط مجال عناوين خاص به وموارد مستقلة عن الخيوط الأخرى، حينها يصبح هذا الخيط عملية مستقلة بحد ذاتها، إن الهدف من إنشاء خيوط متعددة في عملية واحدة هو إقامة التعاون بين هذه الخيوط لإنجاز مهمة معينة.

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

الشكل 33

وكما هو الحال في حالات العمليات، فإن للخيوط أيضا عدة حالات، الفعالة، والجاهزة، والمنتظرة، إضافة إلى الخيوط المنتهية من عملها. فالخيط الفعال هو الخيط الذي يملك المعالج الآن، أما الخيط المنتظر فهو الذي ينتظر حدوث حدث معين، ومن الممكن أن يقوم خيط آخر في نفس العملية بإحداث هذا الحدث.

ينبغي أن يكون لكل خيط مكدس خاص به، كما هو موضح في الشكل. حيث يحوي مكدس كل خيط حجرة ذاكرية تحوي عنوان كل إجراء مستدعى لم يتم الانتهاء منه بعد، يحوي هذا المكدس أيضا المتحولات المحلية للإجراء وعنوان العودة الذي سيستخدم عند انتهاء الإجراء. ويمكن أن يبرر وجود المكدس المستقل لكل خيط وذلك لأن لكل خيط تسلسل من السير والاستدعاءات المختلفة عن الخيط الآخر، مما يعني ضرورة حفظ هذا المسار في بنية مستقلة أيضا، وهذه البنية تتمثل بالمكدس المستقل.



الشكل34

عند وجود إمكانية تعدد الخيوط في النظام، فإن العملية عندما تبدأ العمل تكون بخيط واحد، ومن ثم يقوم هذا الخيط بإنشاء الخيوط الأخرى عن طريق استدعاء ما، وهذا الاستدعاء سيحدد بوسائط من قبل الخيط المستدعي، وليس من الضروري تحديد الفضاء الخاص بهذا الخيط، لأنه يمتلك الفضاء المتعلق بالعملية بشكل كامل، وتكون هذه العلاقة بين الخيط الذي قام بتنفيذ استدعاء إنشاء الخيط والخيط المنشأ هي علاقة تساوي (أي أن العلاقة بين هذه الخيوط ليست علاقة هرمية كما هي عليه الحال في العمليات).

والآن سنتحدث عن أنواع الخيوط وفيها سنتطرق بنوع من التفاصيل على النوعين الأساسيين (الخيط على مستوى النواة، والخيط على مستوى المستخدم)، أما الخيوط الأخرى فهي محاولات للدمج بين هذين النوعين من الخيوط، ولكننا لن نتكلم عنه في أطروحتنا هذه.

5 6 2 -أنواع الخيوط

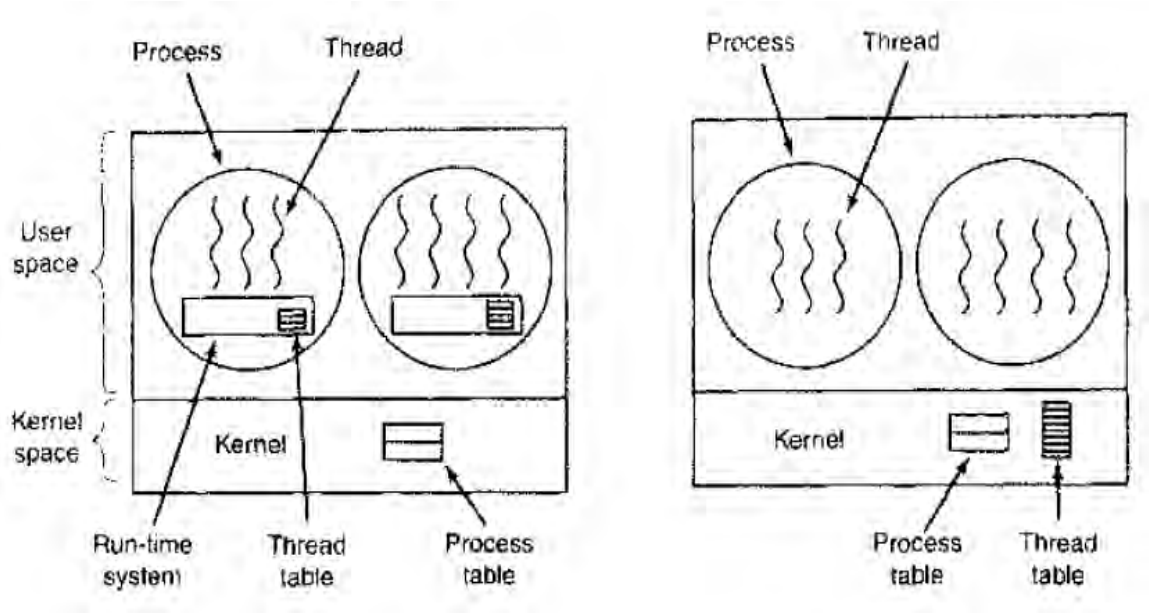
هناك طريقتين لتطبيق الخيوط، الأولى على مستوى المستخدم Threads in user space، والثانية على مستوى النواة Threads in kernel space. وذلك فهناك نوع من الخيوط الهجينة بين هذين المستويين.وسندرس الآن هذين النوعين.

5 6 3 -الخيوط على مستوى المستخدم

يعني هذا النوع أن تنفيذ الخيط يكون على مستوى المستخدم، ولا تعلم النواة عنه شيء. وتكون هذه الحالة مفيدة عندما نريد إنشاء تطبيق متعدد الخيوط في نواة نظام تشغيل قديم لا تدعم الخيوط (وهذا حاصل في كل الأنظمة القديمة، إضافة إلى أن بعض الأنظمة الحديثة لا تدعم هذه التقنية).

عندما تدار الخيوط على هذا المستوى فإننا بحاجة إلى منفذ لعملية الجدولة هذه، كذلك نحتاج إلى جدول لحفظ حالة الخيط على مستوى المستخدم(Thread Table)، تماما كما كنا

بحاجة إلى جدول لحفظ حالة العملية، ويحتفظ هذا الجدول بخصائص عن حالة الخيط، ومكان التعليمة التالية التي سيقوم بتنفيذها، ومؤشر المكدس، وقيمة المتحولات. ويقوم المدير الخيوط الذي يكون على مستوى المستخدم بتوزيع الوقت بين الخيوط كما كانت النواة توزع المعالج بين العمليات بناء على عامل الوقت.



الشكل35

فعندما ينتظر خيطا ما خيطا آخر ليكمل عملا ما، سيطلب إجراء من نظام زمن التنفيذ، يفحص هذا الإجراء وجود نقل هذا الخيط إلى حالة التوقف، وعندها يجب حفظ حالة المسجلات، وحالة المكدس والأمور الأخرى التي تتعلق بالخيط الحالي، ويقوم بعد ذلك بإيقاف هذا الخيط، ونقل خيط آخر إلى التنفيذ (حالة الفعالية) وذلك عن طريق نقل حالة المسجلات المحفوظة لهذا الخيط (وذلك في جدول الخيط Thread table) إلى مسجلات المعالج وتنفيذ التعليمات التالية التي وقف عندها هذا الخيط (في حال تم استبعاده من قبل).

إن إنجاز تبديل الخيوط بهذه السرعة عندما يكون الخيط على مستوى المستخدم يعتبر من أهم الميزات التي تفوق الخيوط على مستوى النواة الذي يتطلب تعليمات أكبر في حال التبديل بين الخيوط.عند انتهاء خيط من عمل ما، يتم استدعاءه لمجدول الخيوط ليقوم هذا الأخير بالتبديل بين الخيوط، وذلك دون الحاجة إلى تفريغ الذاكرة المخبئة (Cache Memory) أو النزول إلى مستوى النواة للقيام بهذا التبديل، وكل هذا ينصب لمصلحة عامل السرعة الذي يميز هذه الخيوط كما أشرنا سابقا.

كما أن هذه الخيوط تتميز في أنها تمنح كل عملية إمكانية اختيارها لأسلوب الجدولة الخاص بها. وهذا الأمر غير متوفر في الخيوط على مستوى النواة، فمثلا تقوم بعض العمليات بفرز خيط خاص بإجراء عمليات تحرير الموارد المحجوزة والغير مستخدمة.وعلى الرغم من أن هذه الخيوط تتميز بالسرعة العالية إلا أنها تعاني من المشاكل الكبيرة، ومن أهمها مشكلة استدعاء النظام المعيق من النواة، فعندما يطلب خيط أمر قراءة من القرص، فإن النواة عندما تستقبل هذا الطلب، ترى أن هذا الطلب موجه لها من العملية، وليس من خيط ضمن هذه العملية، وذلك لأنها لا ترى الخيوط على مستو النواة، عندها وبدلا من أن تقوم بإعاقة الخيط فقط من هذه العملية، تقوم بإعاقة العملية كلها، على الرغم من وجود خيوط أخرى في حالة الجاهزية للعمل.

من الممكن حل هذه المشكلة بأسلوب نظري من خلال تنفيذ العمليات التي تحتاج إلى إعاقة من خلال أوامر مشروطة، فمثلا لا تقوم عملية طلب القراءة بإعاقة العملية، بل تقوم بإعادة النتيجة في حال كانت المعلومات المطلوبة موجودة مسبقا، وتعيد القيمة صفر في حال عدم تواجد هذه المعلومات على الذاكرة، وذلك دون الاضطرار لإيقاف هذه العملية(استدعاءات غير معيقة)، وتتابع العملية في عمل آخر، بينما يتنبه المعالج - المتخصص بنقل المعلومات من القرص الصلب إلى الذاكرة - إلى أهمية جلب المعلومات التي طلبت ولم تكن جاهزة بعد. ولكن هذا سيغير من طبيعة عمل استدعاءات القراءة من القرص، وبالتالي تغيير بنية النواة، وهذا أمر غير مستحب. علما أن من أهم الميزات التي تتميز بها الخيوط على مستوى المستخدم أنها تعمل على نظام التشغيل دون الحاجة إلى أي تعديلات على مستوى النواة.

هناك مشكلة أخرى في الخيوط على مستوى المستخدم، وهي أن الخيط يستمر بالعمل على المعالج ما لم يتخلى طوعا عنه، مع العلم أن هناك خيوط أخرى مستعدة للعمل وفي حالة الجاهزية، وذلك لعدم وجود مقاطعات زمنية على هذا المستوى، وتكون عملية الجدولة مستحيلة ما لم يتم استدعاء هذا الخيط لمنفذ الجدولة من فترة إلى أخرى، وهذا يجعل من الصعب تطبيق عملية الجدولة على مستوى المستخدم.

وقد تم اقتراح الكثير من الحلول لهذه المشكلة، إلا أنها تبقى مشاكل صعبة التنفيذ وتبطئ من عمل النظام، وإن لم يكن ذلك، فسيكون لها اثر على بنية البرنامج وثباته.

5 6 4 -الخيط على مستوى النواة

في هذه الحالة تكون النواة على علم بالخيوط الموجودة في العملية، وفي هذه الحالة فإننا لم نعد بحاجة إلى منفذ الجدولة حيث يكون هناك جدول في مستوى النواة يحوي معلومات عن كل الخيوط الموجودة في النظام. وعندما يريد خيطا ما طلب إنشاء خيط آخر فإنه يقوم بطلب هذا الأمر من النظام الذي يقوم هو بدوره بهذه العملية وتعديل الجدول الذي لديه والمتعلق بالخيوط، كذلك الأمر بالنسبة للتدمير.

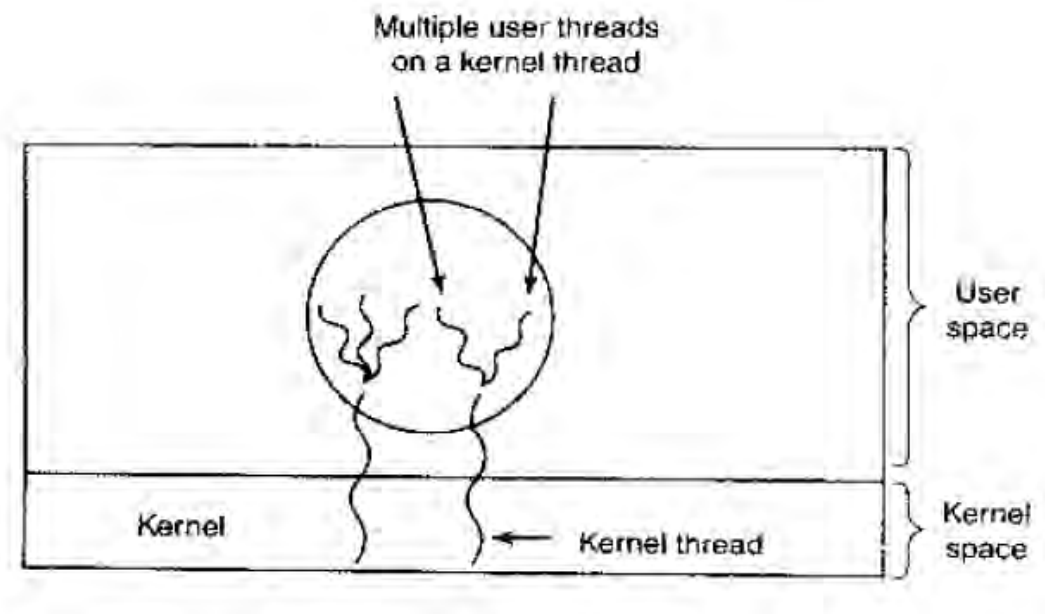
يحوي جدول الخيوط معلومات عن كل خيط تتعلق بحالات المسجلات، ومؤشر إلى التعليمة التالية، وتكون هذه المعلومات محفوظة على مستوى النواة بدلا من منفذ الجدولة على مستوى المستخدم، وتكون هذه المعلومات عبارة عن معلومات فرعية من المعلومات التي يحفظها النظام عن العملية في الأنظمة التقليدية. وبالإضافة إلى حفظ جدول الخيوط على مستوى النواة، تقوم هذه الأخيرة بحفظ حالة العمليات في جدول العمليات وفي نفس المستوى كما هو معرف سابقا.

تنفذ جميع الاستدعاءات المتعلقة بإدارة الخيوط على مستوى النواة، أي أنها تستغرق وقت وكلفة اكبر منها في الخيوط على مستوى المستخدم، وعندما يتم إيقاف أحد الخيوط، تكون النواة حرة في تنفيذ أحد الخيوط من نفس العملية أو الانتقال لتنفيذ خيط آخر من عملية أخرى في حال انتهت المدة المخصصة لهذه العملية، بينما كان نظام زمن التنفيذ في الخيوط على مستوى المستخدم قادرا على التبديل فقط بين الخيوط للعملية الواحدة حتى يقوم النظام بنزع المعالج من هذه العملية عند انتهاء الفترة المخصصة لها.

نظرا لكلفة استدعاء إنشاء خيط على مستوى النواة أو هدمه، تستخدم بعض الخوارزميات لتبسيط هذه العملية، ومن بينها عملية تكرير الخيوط. فمثلا عندما يطلب من النواة

أن تهدم خيط ما، توحي النواة للعملية أن الخيط المتعلق بها قد تم تدميره، بينما يكون قد أوقف هذا الخيط عن العمل، ولم يتم هدمه على مستوى النواة. وفي حال تلقي طلب من أحد العمليات بإنشاء خيط، تقوم النواة بتفعيل الخيط القديم، وربطه بالبرنامج الجديد الذي يريد المستخدم الحصول عليه، مما يوفر في عملية الإنشاء والهدم. ويمكن تطبيق هذه الخوارزمية على الخيوط في مستوى المستخدم، ولكن لصغر التكلفة في إنشاء وهدم الخيوط على هذا المستوى، لا تطبق هذه الخوارزمية عادة.

لا تحتاج الخيوط على مستوى النواة لأي استدعاءات غير معيقة، فمن السهل على النواة التمييز بين العملية والخيط الذي ينتمي لهذه العملية والذي طلب أحد الاستدعاءات المعيقة. السيئة الكبيرة لهذه الخيوط هي أن كلفة استدعاء النظام كبيرة، لذلك في حال وجود استدعاءات إنشاء وهدم كثيرة، فإن هذا سيكلف النظام أعباء كبيرة. و تجدر الملاحظة:يسعى مصممو نظم التشغيل الحديثة لدمج هذه الخيوط مع بعضها للحصول على خيوط هجين تدمج بين محاسن كل من هذين النموذجين السابقين كما هو مبين في الشكل أسفلا.



الشكل36

7 5 -تقنية النسخ عند الكتابة- Copy On Write:

كما نعلم..فإن كل إنشاء لعملية ابن يتم فيه نسخ الفضاء الذاكري (وذلك على أسس الأنظمة المبنية على تقنيات Unix) الأب للابن وذل لتحقيق التعاون بين العملية الأب والابن، ولكن في معظم الأحيان، فإن الابن لا يحتاج من هذه الذاكرة إلى القراءة فقط، وكذلك فإنه لا يحتاج إلا لبعض المتحولات من الأب وليس كلها، مما يعني هدرا في الذاكرة المحجوزة. ولنتخيل في حال تم إنشاء عشرة عمليات ابن لعملية أب، فهذا يعني عشرة نسخ من هذا الفضاء. وبالتالي فإن هذا الترويسة ستتضاعف.

لقد اقترح لحل هذه المشكلة أن يتحول فضاء الذاكرة في حال إنشاء الابن إلى مجال ذاكري قابل للقراءة فقط، وذلك دون مضاعفته، وأن يكون هذا المكان مشتركا بين الأب والابن، ولكن بمجرد محاولة الكتابة على هذا المجال، سواء من الأب أم الابن، فسيتم مضاعفة (نسخ)

المكان الذاكري المراد الكتابة عليه، وإعطاء النسخة الجديدة من هذا المكان للعملية التي طلبت الكتابة، وبالتالي نكون قد انتهينا من مشكلة هدر الذاكرة، وتسمى هذه التقنية بـ Copy On Write (COW)

5 8 -القسم العملي

في هذا القسم سيتم إن شاء الله توصيف بنية الشفرة التي تمثل إدارة العمليات، حيث سيتم في البداية شرح بينة الملفات التي تتعلق بهذا القسم، ثم بعد ذلك سنقوم بشرح عمل بعض التوابع الأساسية والخوارزميات التي تستخدم فيها.

5 8 1- بنية الملفات في قسم إدارة العلميات

جدول9

اسم الملف	مهمة الملف
queue.h	عبارة عن مكتبة للتعامل مع الأرتال (علما أن الأرتال تستخدم بكثرة في هذا القسم).
atomic.h	وهو عبارة عن عمليات تقوم ببناء بعض التعليمات التي تنفذ مرة واحدة دون أن يتخللها أي مقاطعة، وتعتمد هذه المكتبة على بنية المعالج Intel ولذلك فهي موجودة في ضمن المجلد arch.
spinlock.h	تقوم هذه المكتبة بتعريف بنية عمل spinlock والذي يعتبر أحد أنواع الـ Semaphores، وكذلك فإن هذه المكتبة تعتمد على بنية المعالج Intel ولذلك فهي موجودة في ضمن المجلد arch.
Semaphore.h & semaphore	وهذا الملفان هما تنفيذ للقفل Semaphore والذي يقوم بحماية الموارد المشتركة بين العمليات. تعتمد هذه البنية على الملف spinlock.h، بحيث يكون الأخير متعلق ببنية الـ Hardware في حين يكون semaphore.c مستقلا قدر الإمكان عن بنية العتاد.
task.h & task.c	وهي عبارة عن تمثيل للعملية وهي مسماة كما في نظام الـ Linux على أنها task. وفيها البنى المتعلقة بالعمليات، إضافة إلى التوابع التي تقوم بإنشاء وهدم هذه العليمات. إضافة لذلك يتم تنفيذ الخيوط Threads في هذه الملفات.
sched.h & sched.c	وهذه الملفات تقوم بعملية إدارة الجدولة بين العمليات، حيث يتم استدعاء تابع الجدولة كل مرة يتم فيها تنفيذ برنامج خدمة المقاطعة في الـ timer.

عناصر المكتبة semaphore

يتم في هذه المكتبة تنفيذ القفل Semaphore وذلك من أجل القدرة على التعامل مع الموارد المشتركة بين العمليات المختلفة، والتي لا تقبل التعامل مع أكثر من عدد معين من العمليات (علما أن هذه المكتبة مخصصة للتعامل مع الأقفال التي تحمي موردا واحدا فقط- العداد له القيمة البدائية 1-، ولكن من الممكن تعميمها لكي تشمل أكثر من مورد واحد) في وقت واحد.يكون شكل البنية التي تمثل الـ semaphore على الشكل التالي:

```
typedef struct semaphore
{
        atomic_t count;
        int sleepers;
        queue_t *waitq;
} semaphore_t;
```

حيث يعبر المتحول count عن قيمة الموارد الحرة التي لدى الـ semaphore وفي حال كانت هذه القيمة صفر أثناء الطلب، فهذا يعني أن على العملية التي قامت بطلب هذا المورد الانتظار في رتل الانتظار waitq حيث يتم إضافتها إليه.ويتم التصريح عن هذه البنية من خلال هذا الـ macro:

```
#define DECLARE_MUTEX( name ) \
        semaphore_t name = { atomic_init(1), 0, NULL }
```

ويقوم هذا الـ macro بإنشاء هذا القفل وتهيئته على أنه يملك مورد واحدا حرا.

أما الـ macro الثاني:

```
#define DECLARE_MUTEX_LOCKED( name ) \
        semaphore_t name = { atomic_init(0), 0, NULL }
```

فيقوم بإنشاء وتهيئة هذا القفل بدون موارد حرة.ويقوم التابعين:

```
static __INLINE__ void init_MUTEX( semaphore_t *sem );
static __INLINE__ void init_MUTEX_LOCKED( semaphore_t *sem );
```

بتهيئة هذه المتحولات (على أنها في حالة فتح، أو قفل على الترتيب) وذلك من دون إنشاء لهذه المتحولات (أي أنها يجب أن تكون منشأة مسبقا).وهناك تابع آخر يقوم بفحص حالة الـ Semaphore في حال كانت مفتوحة ام مغلقة وذلك دون إعاقة، وهذا التابع هو:

```
#define sem_is_locked( s ) \
                ( atomic_read((&((s)->count)))!=1 )
```

وكما تلاحظ.. فهو مجرد macro من أجل التعامل مع الـ Semaphors التي لا تملك أكثر من مورد واحد.

5 8 1 1- حجز مورد

سنتكلم الآن عن خوارزمية طلب حجز مورد معين مربوط بالقفل Semaphore وهذا الأمر يتم من خلال طلب التابع:

```
static __INLINE__ void DOWN( semaphore_t *sem );
```

حيث تتم الخطوات في الحجز على النحو التالي:

في البداية يتم إنقاص المتحول count بالقيمة 1 ون ثم فحص إن كان المتحول أكبر أو تساوي الصفر فإن حجز المورد قد تم بنجاح، وعندها يتم الخروج من التابع بشكل مباشر. أما في حال كان الشرط غير محقق سيتم في البداية طلب التابع الداخلي __down_failed(sem) الذي سيقوم بالخطوة التالية.
يضيف هذا التابع العملية الحالية (التي طلبت المورد) إلى الرتل waitq ومن ثم يقوم بزيادة العداد sleepers بالقيمة 1.
بعد ذلك تدخل المعالجة في حلقة لا نهائية، والتي تكون الخطوات فيها على النحو التالي:
أولا يتم إدخال المعالج في حالة حجب المقاطعات.
بعد ذلك يتم فحص في حال أصبح المورد محررا أم لا، ففي حال تحرر المورد من العملية التي كانت قد حجزته قبل هذه العملية عندها يتم الخروج من الحلقة، إلا تتابع في تنفيذ التعليمة التالية.
تقوم بالخروج من حالة حجب المقاطعات.
تقوم بتحويل هذه العملية إلى حالة الانتظار (sleeping process).
وفي حال استيقاظها ستتابع في تنفيذ التعليمة الأولى من الحلقة اللانهائية (حجب المقاطعات)، حيث يتم إيقاظها من العملية التي كانت قد حجزت المورد، أو من العملية التي كانت قبلها في رتل الانتظار.
بعد الخروج من حالة النوم يتم نزع هذه العملية من رتل الانتظار waitq ومن ثم تحاول إيقاظ العملية التي تليها.
بعد ذلك تقوم بإلغاء حالة حجب المقاطعات التي كانت فيها.

5 8 1 2 -تحرير المورد

يتم هذا الأمر من خلال التابع المعاكس للتابع السابق في العمل وهو التابع:

```
static __INLINE__ void UP( semaphore_t *sem );
```

وخطوات هذا التابع على النحو التالي:

ويقوم هذا التابع بزيادة القيمة للعداد count ومن ثم يستدعي التابع __up_wakeup(sem) في حال كانت قيمة المتحول العداد أصغر أو تسوي الصفر (أي أن هناك عملية طلبت المورد بعد أن حجزته العملية الحالية).

بعدها يقوم هذا التابع الأخير بالدخول في حالة حجب المقاطعات، ويقوم بإيقاظ أول عنصر(عملية) في الرتل waitq والذي يقوم بالمتابعة في الحلقة اللانهائية التي تم شرحها سابقا ضمن التابع __down_failed(sem)

بعد ذلك يتم إلغاء حالة الحجب للمقاطعات، والخروج من التابع.

5 8 2 - مكتبة الـ Tasks

سنتكلم في هذه المكتبة عن التمثيل الفعلي للعمليات، والبنى الضرورية التي تعبر عن العمليات، وكذلك التوابع المتعلقة بإنشاء العمليات وهدمها وتوابع أخرى متعلقة بها. كذلك سنتكلم في هذه المكتبة عن التوابع المتعلقة بإنشاء الخيوط threads والتوابع الأخرى المتعلقة بها، وسنتطرق بعدها إلى آلية عمل خوارزمية الـ fork المعروفة من قبل المبرمجين، وخوارزمية تنفيذ التطبيق (علما أن هذه الخوارزمية تستطيع تنفيذ البرامج المحفوظة على القرص بصيغة الملفات elf فقط).

5 8 2 1- العمليات

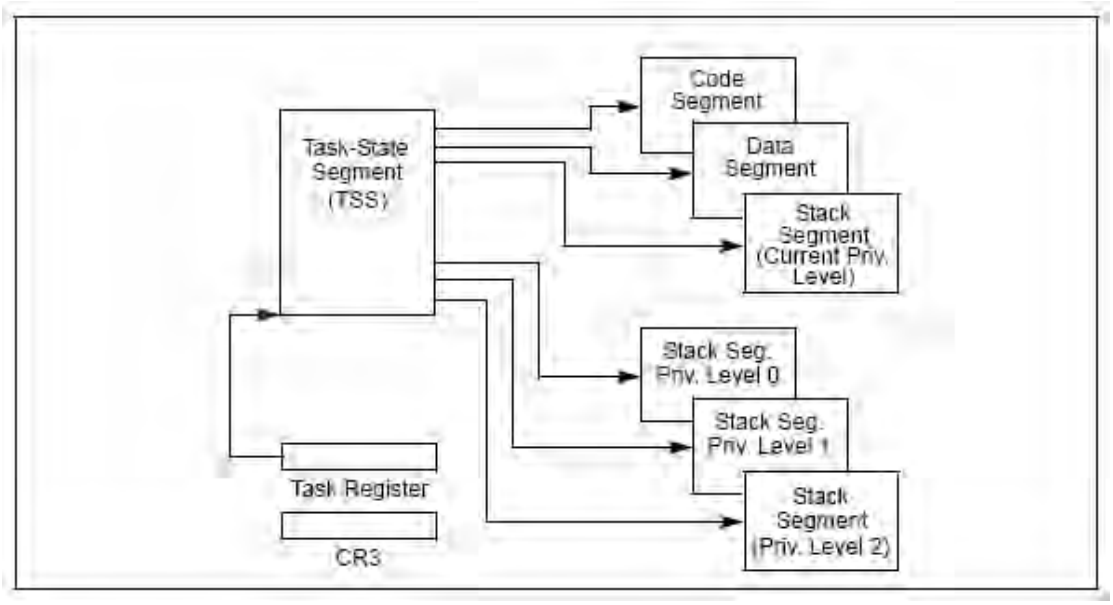
سنتلكم الآن عن العمليات والتوابع المتعلقة بها، حيث تمثل العملية في البنية task ضمن الملف task.h على النحو التالي:

```
typedef struct task
{
        tss_IO_t        tss;
        uint16_t        tss_sel;

        uint32_t        *pdbr;
        unsigned long       pdbr_update_counter;

        pid_t           pid;
        uid_t           uid, euid, suid, fsuid;
        gid_t           gid, egid, sgid, fsgid;

        size_t pl0_stack;

size_t stack;

size_t heap_start;

size_t heap_size;

semaphore_t heap_sem;
        byte            state;

struct task     *father;
int             console;

        task_flags_t  flags;
        int             privilege;
        int             priority;
        int             counter;
        char            name[ 256 ];
} task_t;
```

كما نلاحظ، فإن الحقل الأول من نوع tss_IO_t والذي سندرسه لاحقا، وبالتفصيل، وهو عبارة عن ممثل للتعليمات يفهمه المعالج Intel، وكذلك فإن الحقل الثاني هو الناخب للواصف الذي يكون من النوع TSS Descriptor، والموجود في مصفوفة الـ GDT (راجع

فصل النظام المحمي Protected Mode).كما يمثل الحقل الثالث pdbr فهو مؤشر على الفضاء الذاكري المتعلق بالعملية (يمكنك مراجعة فصل إدارة الذاكرة).يعتبر الحقل pl0_stack مؤشرا إلى ذاكرة المكدس(بالإضافة إلى ذلك فإن هناك ثلاث مؤشرات إلى ثلاثة مكدسات في النظام، كل منها يؤشر إلى مكدس في مستوى من مستويات الصلاحية).

وهنا لدينا الشكل الذي يمثل البنية stack، وهذا الشكل مأخوذ من المرجع IA-32 Intel(R) Architecture Software Developer's Manual Volume3:



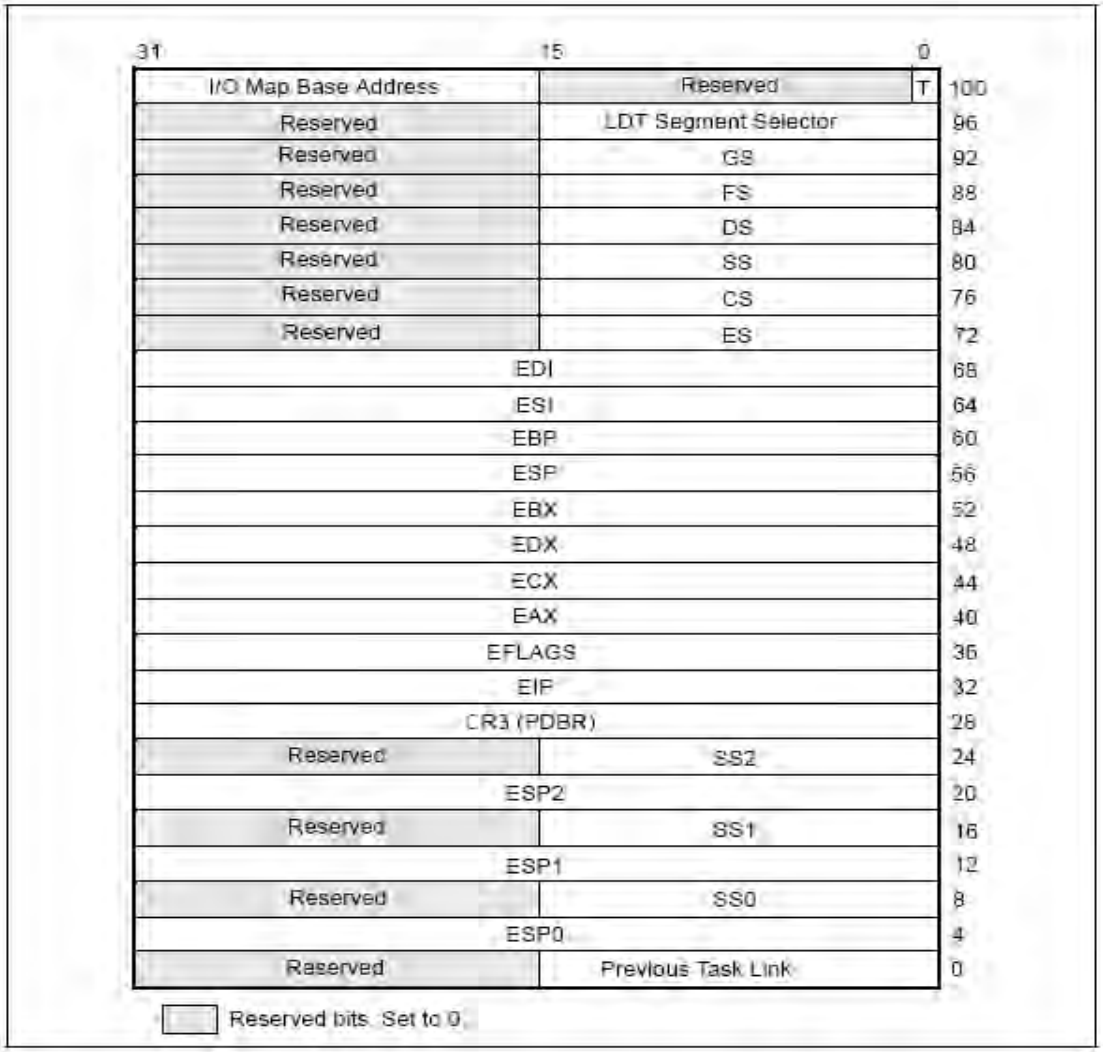
الشكل 37

حيث كما في الشكل السابق نلاحظ ثلاث مكدسات للمستويات الثلاثة بناء على مستوى الولوج المتبع في العملية.نلاحظ كذلك أن هناك حقلين هما heap_start والـ heap_size، واللذين يحددان حجم وبداية منطقة الكومة لهذه العملية.يحدد الحقل father الأب لهذه العملية، أما العملية init في هذا النظام فليس لها أب(أي أن قيمة هذا المؤشر تساوي إلى NULL).

أما الـ console فهو عبارة عن الواجهة التي ستكتب عليها العملية، وكذلك الدخل من خلال المستخدم لهذه العملية.وكما هو واضح من التسمية للحقل name فإن هذا الحقل يعبر عن اسم العملية.هكذا نكون قد وصفنا حقول العملية task بنوع من الإيجاز والشمولية قدر الإمكان.

Task-State Segment (TSS)- 2 2 8 5

يمكننا القول أن هذا التابع هو البنية تمثل العملية والتي يفهمها المعالج Intel، وهذه المقطع من الذاكرة يجب أن يحوي الحقول كما هي على الشكل التالي:



الشكل38

أما التمثيل لهذا التابع في الشفرة فيتم عن طريق السجل tss_IO:

```
typedef struct tss_IO
{
	uint32_t	link;
	uint32_t	esp0;
	uint16_t	ss0, __ss0h;
	uint32_t	esp1;
	uint16_t	ss1, __ss1h;
	uint32_t	esp2;
	uint16_t	ss2, __ss2h;
	uint32_t	cr3;
	uint32_t	eip;
	uint32_t	eflags;
	uint32_t	eax, ecx, edx, ebx;
	uint32_t	esp;
	uint32_t	ebp;
	uint32_t	esi;
	uint32_t	edi;
	uint16_t	es, __esh;
	uint16_t	cs, __csh;
	uint16_t	ss, __ssh;
	uint16_t	ds, __dsh;
	uint16_t	fs, __fsh;
	uint16_t	gs, __gsh;
	uint16_t	ldtr, __ldtrh;
	uint16_t	trace, io_map_addr;
	uint32_t	io_map[IO_MAP_SIZE];
} tss_IO_t;
```

ويمثل الحقل الأول مؤشر إلى العملية التي كانت تملك المعالج قبل هذه العملية الحالية.كذلك فإن هناك مؤشرات إلى المقاطع المتعلقة بهذه العملية،وهذه المؤشرات هي ES, CS, SS, DS, FS وكذلك المؤشر GS.وهناك متحولات تحفظ فيها حالة المسجلات للمعالج أثناء مغادرة العملية للمعالج من قبل المجدول، وكذلك حفظ هذه العمليات أثناء انتزاع المعالج من هذه العملية، وهذه المسجلات هي EAX, ECX, EDX, EBX, ESP, EBP, ESI والمتحول EDI.ويحوي الحقل IDT الذي يحدد شعاع المقاطعات المتعلق بهذه العملية.ويحوي الحقل cr3 (وهو اسم مسجل تحكم) على العنوان الفيزيائي لبداية الـ PDE (راجع فصل إدارة الذاكرة) والذي يحوي خريطة العناوين الذاكرية المتعلقة بهذه العملية.

يمكن مراجعة هذه الحقول وما تدل عليه وذلك في المرجع الذي أشير إليه سابقا من شركة Intel.و تجدر الملاحظة أنه في حالة استخدام الـ Paging في عملية إدارة الذاكرة، فكما هو موضح في الوثائق المرفقة بمعالج Intel، يجب ألا يكون بداية هذا المقطع في منتصف الصفحة، وإنما في أولها، وهذا بدوره يعني حجز صفحة جديدة لكل tss خاص بالعملية، ووضع هذا الـ tss في أول الصفحة، كما سنرى لاحقا في هذا الفصل.

3 8 5 ـحالات العمليات والأرتال المرافقة لها

كما درسنا في القسم النظري فكل عملية تمر بحالات تم دراستها سابقا، وهذه الحالة تخزن في متحول state الموجود في السجل task ويأخذ المتحول أحد هذه القيم التالية:

```
#define TASK_NEW        0
#define TASK_READY          1
#define TASK_WAIT       2
#define TASK_ZOMBIE     4
```

وكل حالة من هذه الحالات عدا الحالة الأولى تحوي رتل مقابل لها بحيث يتم إضافة هذه العملية إلى الرتل الموافق للحالة التي تحولت لها، وهذه العملية تتم من خلال macros متخصصة تقوم بتغيير الحالة للعملية واستدعاء التابع sched_refresh_queues والذي سندرسه لاحقا في هذا الفصل. حيث يقوم هذا التابع بالمرور على الأرتال الثلاثة كلها، ومن ثم النظر إلى حالة العمليات في هذه الأرتال، فإن كانت الحالة غير موافقة للرتل يتم نقل العلمية إلى الرتل المناسب، و تجدر الملاحظة إلى أنه لا يتم إنشاء رتل للحالة new لأن الحالة لا تلبث في هذه الحالة إلى فترة قليلة أثناء وجودها في التابع creat_task ومن ثم لا تعود إلى هذه الحالة أبدا، علما أيضا أنه لا يوجد إلى عملية واحدة في نفس اللحظة في الحالة new.

ملاحظة:

لا يتم إنشاء رتل للحالة new لأن العملية لا تلبث في هذه الحالة إلى فترة قليلة أثناء وجودها في التابع creat_task ومن ثم لا تعود إلى هذه الحالة أبدا، علما أيضا أنه لا يوجد إلى عملية واحدة في نفس اللحظة في الحالة new.

5 8 4 -إنشاء العلمية:

سندرس الآن التابع create_process والذي يقوم بإنشاء العملية. ويسير هذا التابع وفق المراحل التالية:

1. ينشئ هذا التابع مكان ذاكري في أول الصفحة بحيث من اجل المقطع tss الذي تكلمنا عنه سابقا.
2. يتم وضع هذه العملية في الرتل zombie وذلك لأن هذه العملية لم تسند على أنها إبن لأي أب إلى الآن، وطبعا تكون الحالة لها في البداية على أنها new.
3. يتم بعدها حجز مكان للمكدس في ذاكرة الكومة، ويكون هذا المكدس فقط للتعاملات على مستوى النواة، ومن ثم يتم إسناد المؤشر pl0- stack إلى هذا المكان، ويتم تهيئة هذا المكدس بالقيمة 0 على كامل حجمه.
4. يتم بعد ذلك إضافة واصف المقطع tss الذي تم إنشاءه مؤخرا إلى مصفوفة الـ GDT.
5. ثم بعد ذلك يتم إنشاء فضاء عناوين لهذه العملية.
6. يتم بعدها إنشاء مكدس للتعامل إما على مستوى النواة أو على مستوى المستخدم.
7. ثم بعد ذلك يتم تهيئة الكومة من أجل العملية.

8. بعد ذلك تهيئ بعض المتحولات في النظام. ومن ثم يتم الربط بين الأب والابن، ومن ثم تنقل حالة العملية من الحالة zombie إلى الحالة ready، كذلك ويتم الإشارة إلى الشيفرة التي يراد تنفيذها من قبل العملية(حيث قد مرر هذا المؤشر كوسيط للتابع).
9. يتم الخروج من حالة حجب المقاطعات، حيث يتم حجبها منذ بداية الدخول في هذا التابع، وبما أن العمليات التي تم تنفيذها في هذا التابع تستغرق وقتا طويلا، فمن الأفضل استدعاء تابع الجدولة من اجل التحقق من عدم انتظار مهمة ذات ألوية عالية على الرتل أثناء تنفيذ هذا التابع.

5 8 5 -إنشاء خيط thread:

5 8 5 1- كلمة أولية:

كما عرفنا فإن هناك مستويين من الخيوط، الأول على مستوى المستخدم، والثاني على مستوى النواة، وفي النظام الذي بين أيدينا نلاحظ أن التطبيق لهذه الخيط يكون على مستوى النواة، فالنواة تكون على علم بوجود الخيوط، وهي المسؤولة عن إدارتها وجدولتها.

5 8 5 2- الشرح:

يتم في إنشاء الخيط نفس العمليات التي تمت في إنشاء العملية، إلا أن هناك من الأمور على مستوى العملية لا يتم حجزها على مستوى الخيط، سندرس الآن عملية إنشاء الخيط:

1. ينشئ هذا التابع مكان ذاكري في أول الصفحة بحيث من اجل المقطع tss الذي تكلمنا عنه سابقا.
2. يتم وضع هذه العملية في الرتل zombie وذلك لأن هذه العملية لم تسند على أنها إبن لأي أب إلى الآن، وطبعا تكون الحالة لها في البداية على أنها new.
3. يتم بعدها حجز مكان للمكدس في ذاكرة الكومة، ويكون هذا المكدس فقط للتعاملات على مستوى النواة، ومن ثم يتم إسناد المؤشر pl0- stack إلى هذا المكان، ويتم تهيئة هذا المكدس بالقيمة 0 على كامل حجمه.
4. يتم بعد ذلك إضافة واصف المقطع tss الذي تم إنشاءه مؤخرا إلى مصفوفة الـ GDT، وذلك على انه واصف لخيط وليس لعملية.
5. يتم بعدها إنشاء مكدس للتعامل على مستوى النواة ، او سيكون على المكدس على الكومة في حال لم يكن الاستدعاء من قبل النواة.
6. يتم الحصول على العملية الأب والابن، ومن ثم يتم ربط الكومة والمكدس وكل الموارد وفضاء العاوين لهذا الأب مع الابن.

7. بعد ذلك تهيئ بعض المتحولات في النظام. ومن ثم يتم الربط بين الأب والابن(يمكن أن يكون الأب هنا هو thread وليس عملية)، ومن ثم تنقل حالة العملية من الحالة zombie إلى الحالة ready، كذلك ويتم الإشارة إلى الشيفرة التي يراد تنفيذها من قبل الخيط.
8. يتم الخروج من حالة حجب المقاطعات، حيث يتم حجبها منذ بداية الدخول في هذا التابع، وبما أن العمليات التي تم تنفيذها في هذا التابع تستغرق وقتا طويلا، فمن الأفضل استدعاء تابع الجدولة من أجل التحقق من عدم انتظار مهمة ذات ألوية عالية على الرتل أثناء تنفيذ هذا التابع.

5 8 6 -التابع do_fork:

معروف عند كثير من مبرمجي الـ Unix التابع fork الذي يقوم بإنشاء عملية ابن عند استدعاء التابع وإنشاء مكان ذاكري جديد للابن (وذلك قبل استخدام تقنية الـ COW) وهو عبارة عن نسخة طبق الأصل عن العملية الأب، وكذلك قيم المتحولات، إلا أنه يعيد إلى العملية الابن القيمة (0)، بينما يعيد إلى العملية الأب الـ PID للابن الذي تم إنشاؤه، حيث يتم التواصل مع هذا الابن عن طريق هذا الرقم.

تشبه الخوارزمية المستخدمة في التابع do_fork الخوارزمية المستخدمة في تابع إنشاء عملية، إلا أنه يأخذ قيم المسجلات الحالية من المعالج (مسجلات العملية التي استدعت التابع fork) في بداية الإجرائية ويسجلها عنده، وذلك حتى يكون نسخة طبق الأصل عن هذه العملية التي أنشأته (حيث تكون هي العملية الحالية).

كما أن هذا التابع يعيد للأب الـ PID للعملية التي أنشأها مؤخرا، في حين لا يعيد شيئا للابن وهذا يعني أن القيمة المعادة له هي القيمة (0).

ويجدر الذكر أن هذا التابع الذي نعرفه الآن do_fork، يتم تغليفه من قبل التابع fork في استدعاء النظام System Call.

5 8 7 -تنفيذ ملف من خلال التابع exec_file:

يقوم هذا التابع باستدعاء توابع التحميل من القرص الصلب بعد حجز المكان الذاكري المناسب، وإنشاء عملية خاصة له، وربط هذه العملية بشيفرة هذا البرنامج. تكون الخطوات بالتفصيل على النحو التالي (حيث لا يقوم هذا التابع إلا بتحميل الملفات من elf):

1. يتم في البداية معرفة حجم هذا الملف على القرص ن خلال التابع `fat12_file_size`.
2. يتم حجز المكان الذاكري المناسب لهذا الملف، وذلك حتى يتم تحميله كاملا على الذاكرة.
3. يتم استدعاء التابع elf_load_file والذي يقوم بتحمييل هذا الملف، حيث يتم في هذا التابع التأكد أن صيغة هذا الملف من نوع elf

4. يتم بعدها إجراء الخطوات نفسها المتبعة لإنشاء عملية.

وسيتم تفصيل بنية الـ elf في فصل مستقل.

5 8 8 -قتل عملية:

يتم قتل هذه العلمية من خلال التابع التالي:

```
bool kill( pid_t pid );
```

حيث يقم هذا التابع بتحويل العملية من الحالة ready أو الحالة wait إلى الحالة zombie، حيث يتم حذف الموارد المخصصة لهذه العملية من خلال العملية kpagered التي سنتكلم عنها في قسم إدارة العليمات.

وعند تحويل هذه العملية إلى zombie فعلينا أن نحدث حالة العمليات الابن لهذه العملية وذلك من خلال المرور على العمليات كلها، وفحص العمليات الابن للعملية التي نريد قتلها، أما الخيوط التي تنتمي إلى هذه العملية فيجب أن تتحول بدورها إلى الحالة zombie.

أما التابع wait_pid فيقوم بجعل العملية المستدعاة تنتظر حتى موت العملية البن، والتي يكون الـ pid الذي لها ممررا كوسيط، اما إذا كان الوسيط pid ذو قيمة 1- فهذا يعني أن العملية الأب ستنتظر حتى موت عملية عشوائية من العمليات الابن لديها. ويكون شكل التابع:

```
pid_t waitpid(pid_t pid, int *status, int options);
```

5 8 9 -المجدول – مكتبة الـ sched:

سنتكلم الآن عن آلية عمل المجدول في نقل السياق بين العمليات طبعا أن يكون قد تم حفظ حالة العملية التي ستنتقل إلى حالة الجاهزية، وتحميل حالة العملية التي ستنتقل إلى حالة العمل (والتي سميناها سابقا حالة الفعالية).

هناك عدة توابع من أجل التعامل مع حالة المجدول (موقف أم في حالة عمل) وذلك من خلال التابع:

```
__INLINE__ int sched_is_disabled( void );
```

والذي يعيد الحالة صفر عندما يكون المجدول في حالة إيقاف، ويعيد رقم خلاف ذلك في حالة العمل.

أما تغيير حالة المجدول فهي عديدة، وكمثال عليها:

```
__INLINE__ void sched_enter_critical_region( void );
__INLINE__ void sched_leave_critical_region( void );
```

أما التابع :

```
static __INLINE__ void do_refresh_queues();
```

والذي يقوم بالمرور على الأرتال وفحص حالة العمليات التي فيها، فإن كانت حالة الرتل لا تتوافق مع حالة العملية، يقوم هذا التابع بنقل العملية إلى الرتل المناسب.

علما أن هناك العملية kpagd فهي تشبه الـ garbage collector في لغات البرمجة والتي تقوم بحذف الموارد التي تحجز من العمليات التي هي في حالة zombie وحذفها، وذلك في كل مرة يتم إيقاظه فيها، حيث تكون هذه العملية نائمة في معظم الأحيان.

5 9 -المجدول:

والآن سنتكلم عن المجدول والآلية التي يعمل بها، ويكون تصريح هذا التابع هو:

```
void schedule( void );
```

حيث سيقوم هذا التابع بالخطوات لتالية:

1. فحص حالة المجدول، فإن كان المجول يجب أن يكون في حالة إيقاف عمل، يتم انتهاء عمل هذا التابع فورا.
2. ثم بعد ذلك يقوم التابع بحجب المقاطعات من اجل الدخول في عملية الجدولة دون أي مقاطعة.
3. يستدعي هذا التابع في التابع `do_refresh_queues` إن كان هناك حاجة لذلك (يتم معرفة هذه الحاجة من خلال متحول من نوع atomic والذي يغير قيمته في كل مرة تتم فيها تعديل حالة إحدى العمليات).
4. بعد ذلك يتم المرور على رتل الجاهزية، و البحث عن العملية التي تملك المتحول counter الأصغر في الرتل(حيث يهيء هذا المتحول في بداية عمل العملية على أولوية المتحول)، وأثناء المرور على كل علمية فإنه يتم إنقاص هذا المتحول بالقيمة واحد.
5. يتم اختيار هذه العملية حتى تكون البديل عن العملية التي تعمل حاليا، وذلك من رخلال تنفيذ التابع `switch_to` والذي يقوم بالنقل بينن العمليات.
6. الخروج من حالة حجب المقاطعات.

أما التابع switch_to فهو يقوم باستدعاءات بعض أوامر المعالج Intel والذي يقوم بالنقل بين العمليات، ومن ثم يقوم هو بدوره بالانتقال إلى فضاء الذاكري للعملية الجديدة.

ويكون تصريحه على الشكل التالي:

```
static __INLINE__ void switch_to( task_t *prev, task_t *next );
```

5.9.1 - مكتبة الـ queue.h

يحوي هذا الملف على التوابع التي تسهل التعامل مع الأرتال، فهي تقوم بتغليفها بتوابع وmacros تسهل على المبرمج التعامل معها، وتكون مكتوبة بطريقة تسرع من البحث خلال هذه الأرتال.

5.9.1.1 - بنية الرتل

فبنية الأرتال تكون كما هو موضح بالشكل:

```
typedef struct queue
{
	void *value;
	struct queue *next;
	struct queue *prev;
} queue_t;
```

وكما هو ملاحظ فإن هذه الأرتال مضاعفة (أي أن كل عنصر يشير إلى العنصر السابق والعنصر اللاحق عن طريق المؤشرات prev والمؤشر next على الترتيب).

5.9.1.2 - إضافة عنصر

هناك تابعان لإضافة عنصر جديد إلى الرتل وهما:

```
static __INLINE__ int add_queue( queue_t **q, void *v );
static __INLINE__ int add_queue_head( queue_t **q, void *v );
```

حيث يضيف التابع الأول العنصر إلى آخر الرتل، في حين يضيف التابع الثاني العنصر إلى أول الرتل (أي بعد رأس الرتل head). علما أن القيمة value هي مؤشر إلى متحول القيمة الذي يراد أن يسند إلى العنصر الجديد المراد إنشاءه.

5.9.1.3 - حذف عنصر

بنفس الطريقة التي تم بها إضافة عنصر يتم حذف العنصر:

```
static __INLINE__ int queue_del_head( queue_t **q );
static __INLINE__ void __queue_del_entry( queue_t **q, queue_t *entry);
```

كذلك فإن التابع الأول يتم فيه حذف العنصر من الرتل، والثاني يتم فيه حذف العنصر الذي يلي رأس الرتل. حيث تعبر القيمة value الممررة في التابع الثاني عن قيمة العنصر المراد

حذفه، وذلك من أجل البحث عن هذا العنصر ومن ثم حذفه، في حين أن الحذف في التابع الأول يكون مباشرة للعنصر الذي يلي رأس الرتل.

5.9.4.1- الحصول على قيمة أحد العناصر

يقوم هذا التابع باستحصال القيمة value من العنصر وذلك من خلال هذا التابع.

```
static __INLINE__ void *queue_get_entry( queue_t *entry );
```

5.9.5.1- البحث عن قيمة متحول

يقوم هذا التابع بالبحث عن قيمة معينة لأحد العناصر الموجود في الرتل:

```
static __INLINE__ void *find_queue(queue_t **q, void *v);
```

5.9.6.1- هدم الرتل

بعد الانتهاء من استخدام هذا الرتل فإنه لابد من إيجاد بتابع يقوم بهدم الموارد الذاكرية التي قام هذا الرتل بحجزها، حيث يقوم هذا التابع بهذه المهمة:

```
static __INLINE__ int clear_queue( queue_t **q );
```

وهناك توابع أخرى في هذه العملية لن نذكرها هنا، وذلك تجنبا للدخول في تفاصيل المكتبة.

5.9.2- عناصر المكتبة atomic.h

تستخدم هذه المكتبة للتعامل مع التعليمات التي تتم دفعة واحدة ودون إجراء أي مقاطعة (تكون هذه التعليمات أصغر تعليمة مسموح بها في المعالج)، يتم في هذه المكتبة استخدام ما يسمى بالـ lock prefix في معالج Intel والتي تقوم بحجز ممر المعطيات للمعالج الذي طلب عملية القفل، وهذه العملية مهمة لتحقيق التعامل مع الذاكرة المشتركة في حال كان النظام يعمل على Multiprocessor.

يكون شكل بنية المتحول الذري على النحو التالي:

```
typedef struct atomic
{
        volatile int counter;
} atomic_t;
```

أما الهدف من التوابع التي تتعامل مع هذه البنية هو القيام بعمليات لا تقبل المقاطعة أثناء تنفيذ هذه التوابع.إن هدف هذه المكتبة هو إيجاد بنية تحتية تقوم عليها الأقفال على الموارد المشتركة مثل الـ Semaphore.

5.9.3- عناصر المكتبة spinlock.h

تعتبر المكتبة spinlock.h عبارة عن قفل من أجل إيقاف المقاطعات.

5 10 -المراجع

السنة	الإصدار	المؤلف	اسم المرجع
October 2000 ISBN: 0-596-00002-2,	First Edition	Daniel P. Bovet Marco Cesati	**Understanding the Linux Kernel** / chapter 2&6&7
1992	**second Edition**	**Andrew Tanenbaum**	**Modern Operating Systems / Chapter 4 – Memory Managment**
1992	**second Edition**	**Andrew Tanenbaum**	**Operating Systems Design and Implementation/ Chapter 4 – Memory Managment**
June 2005	Order Numbers 253666 and 253667	**Intel**	The IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference.
June 2005	**Order Number: 253668-016**	**Intel**	IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide/ Chapter 2&3
01/01/02		**Tim Robinson**	(Memory Management 1)http://www.osdever.net
01/01/02		**Tim Robinson**	(Memory Management 2)http://www.osdever.net
01/01/02		**Mike Rieker**	(Pagetables)http://www.osdever.net
November 18, 1999		-	(Dynamic Memory Allocation)http://www.osdever.net
		-	(Algorithms and Tips for Memory Management) http://www.mega-tokyo.com/osfaq2/index.php/Algorithms%20and%20Tips%20for%20Memory%20Management

6 -أنظمة الملفات

مُقَدِّمَة

تحتاج جميع تطبيقات الكمبيوتر إلى تخزين واستعادة المعلومات. وعندما تكون العملية في حالة عمل، فبإمكانها تخزين كمية قليلة من المعلومات ضمن حيزها الذاكري. ولكن معظم التطبيقات تحتاج لتخزين كمية كبيرة من المعلومات. وبالتالي هناك مشكلة أخرى في عملية حفظ المعلومات في الحيز الذاكري للعملية وهي أنه بانتهاء عمل العملية ستضيع هذه المعلومات.المشكلة الثالثة أنه أحياناً تحتاج عدة عمليات للوصول إلى المعلومات (أو جزء منها) في نفس الوقت. إذا كان لدينا دليل هواتف مخزن في الحيز الذاكري لعملية ما فإن هذه العملية هي فقط التي تستطيع الوصول إليه. تتمثل طريقة حل هذه المشكلة في جعل المعلومات نفسها مستقلة عن أي عملية معينة.

وبالتالي لدينا ثلاث متطلبات أساسية لتخزين المعلومات طويل المدى:

يجب أن يسمح بتخزين كمية كبيرة جدا من المعلومات.
يجب أن تبقى المعلومات بعد انتهاء العملية.
يجب أن تتمكن عدة عمليات من الوصول إلى المعلومات في نفس الوقت.

إن الحل الاعتيادي لجميع هذه المشاكل هو بتخزين المعلومات على الأقراص وغيرها من الوسائط الخارجية ضمن وحدات تسمى الملفات. تستطيع العمليات عند ذلك قراءة هذه الملفات وكتابة ملفات جديدة عند الحاجة. يجب أن تكون المعلومات المخزنة في الملفات محفوظة، أي أنها لا تتأثر بإنشاء أو انتهاء العمليات. يجب أن لا يختفي الملف إلا عندما يقوم مالكه بإزالته صراحة.

تدار الملفات من قبل نظام التشغيل. تعتبر بنائها وتسميتها والوصول إليها واستخدامها وحمايتها وتحقيقها من أهم المواضيع في تصميم أنظمة التشغيل. يسمى جزء نظام التشغيل الذي يتعامل مع الملفات بأكمله باسم نظام الملفات (File System)

6 .1 -الملفات

سنتكلم في الصفحات التالية عن الملفات كيف تستخدم و ما هي خصائصها، ووجهة نظر أنظمة التشغيل المشهورة فيها.

6 .1 .1 -تسمية الملفات

عندما تنشئ عملية ما ملفاً فإنها تعطي له اسماً. وعندما تنتهي العملية يبقى الملف موجودا ويمكن الوصول إليه من قبل عملية أخرى باستخدام اسمه.تختلف تفاصيل قواعد تسمية

الملفات من نظام إلى آخر, ولكن جميع أنظمة التشغيل الحالية تسمح بتسمية الملفات بسلاسل من حرف إلى ثمانية حروف. وتدعم العديد من أنظمة الملفات الأسماء الطويلة حتى 255 حرف.

تفرق بعض أنظمة الملفات بين الأحرف الكبيرة والصغيرة, بينما لا تميز بينهما أخرى. يصنف UNIX في المجموعة الأولى بينما يصنف MS-DOS في الثانية لذلك, يمكن في UNIXوجود ثلاثة ملفات بالأسماء SAMMER و Sammer و sammer. أما في MS-DOS فتشير جميع هذه الأسماء إلى نفس الملف.

يستخدم كل من Windows95, Windows98 نظام ملفات MS-DOS ويرثان نتيجة ذلك العديد من خصائصه مثل كيفية تشكيل أسماء الملفات. بالإضافة إلى ذلك يدعم كل من Windows 2000 و Widows NT و كذلكWindows XP نظام ملفات MS-DOS ويرثون أيضاً العديد من خصائصه. إلا أن الأنظمة الثلاثة الأخيرة لديها أيضاً نظام ملفات خاصة بها هو (NTFS) الذي يختص بخصائص مختلفة (مثل تسمية الملفات بشفرة Unicode) عندما نشير إلى نظام ملفات Windows في هذا الفصل فإننا نقصد نظام ملفات MS-DOSوهو نظام الملفات الوحيد الذي تدعمه جميع الإصدارات .Windows.

تدعم العديد من أنظمة التشغيل أسماء ملفات ذات جزأين حيث تفصل ما بين الجزأين نقطة كما في prog.c يسمى الجزء الذي يلي النقطة بامتداد الملف File Extension وعادة ما يدل على خاصية معينة للملف. مثلاً يتألف اسم الملف في MS-DOS من 1 إلى 8 محارف بالإضافة إلى امتداد اختياري من محرف إلى ثلاثة محارف. يعود حجم الامتداد في UNIX إلى رغبة المستخدم وقد يملك الملف امتدادين أو أكثر كما في prog.c.z حيث تستخدم z عادة للدلالة على أن الملف (prog.c) قد ضغط باستخدام خوارزمية ziv-lempel للضغط. يبين الشكل بعض امتدادات الملفات الأكثر شهرة ومعانيها.

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.f77	Fortran 77 program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

الشكل39

تعتبر امتدادات الملفات في بعض الأنظمة (مثل UNIX) مجرد اصطلاحات ولا تكون مفروضة من قبل نظام التشغيل. مثلاً قد يكون الملف file.txt عبارة عن نوع الملفات النصية لكن تسميته بهذا الشكل هي من اجل تسميته بهذا الشكل هي من اجل تذكير المستخدم

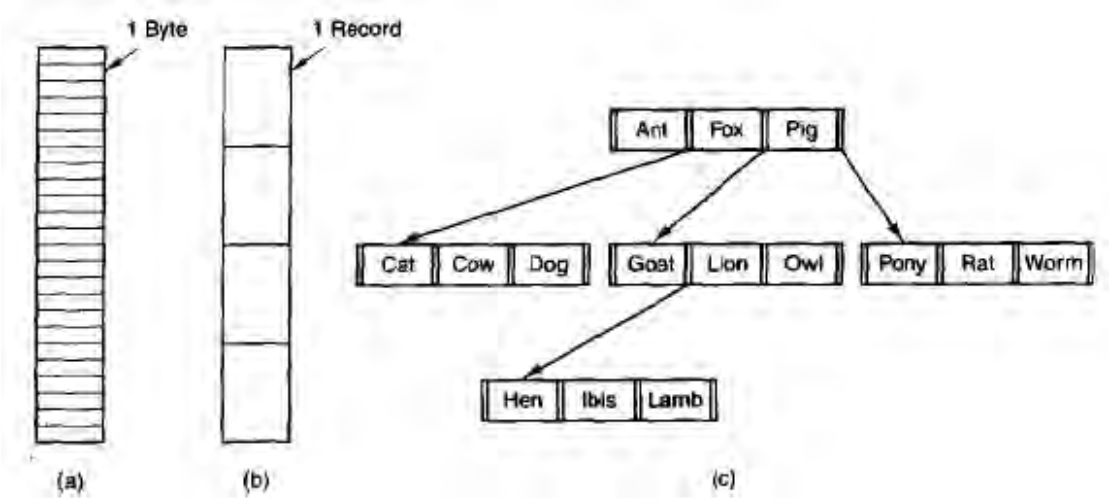
بمحتواه وليس تقديم أية معلومات للحاسب. من ناحية أخرى قد يصر مترجم C أن تنتهي الملفات بالامتداد C ويرفض ترجمتها إن لم تكن كذلك.

تصبح هذه المصطلحات مفيدة خصوصاً عندما يستطيع نفس البرنامج التعامل مع عدة أنواع من الملفات. يمكن مثلا إعطاء لائحة من الملفات إلى مترجم C كي يترجمها ويربطها مع بعضها قد يكون بعضها ملفات C وبعضها ملفات لغة التجميع ويميز الملفات الأخرى.

على العكس من ذلك يهتم Windows بامتدادات الملفات ويربطها بمعاني خاصة. يستطيع المستخدمون (أو العمليات) تسجيل الامتدادات في نظام التشغيل وتحديد لبرنامج الذي يملكها من اجل كل امتداد. عندما ينقر المستخدم نقراً مزدوجاً على اسم الملف يتم تشغيل البرنامج المرتبط بامتداد هذا الملف ويمرر اسم الملف كبارامتر له مثلا يؤدي النقر على الملف file.doc إلى تشغيل برنامج Microsoft word مع فتح الملف file doc للتحرير.

6 1 2 -بنية الملف

يمكن بناء الملفات بعدة طرق مختلفة.يبين الشكل التالي ثلاثة طرق شائعة الاستخدام. الملف المبين في الشكل (A)عبارة عن تسلسل غير مهيكل من البايتات. في الواقع لا يعرف نظام التشغيل ولا يهتم بمحتويات الملف. كل ما يراه هو مجموعة بايتات. يتم تحديد المعنى من قبل برامج مستوى المستخدم. يستخدم كل من UNIXوWindows هذه الطريقة.



الشكل 40

إن جعل نظام التشغيل يرى الملفات مجرد تسلسل بايتات لا أكثر يعطي المرونة الأكبر. تستطيع برامج المستخدم وضع أي شي تريده في ملفاتها وتسميها بأي طريقة تلائمها. لا يساعد نظام التشغيل في شي لكنه أيضاً لا يعيق حركة برامج المستخدم. وهذه ميزة هامة بالنسبة للمستخدمين الذين يريدون القيام بأعمال غير اعتيادية.

يبين الشكل (b) نموذجا أكثر هيكلية. يتألف الملف في هذا النموذج من سلسلة من السجلات الثابتة الحجم كل منها له بنية داخلية معينة. بما أن الملف مكون من تسلسل من السجلات فان كل عملية قراءة تقرأ سجلا واحدا وكل عملية كتابة تكتب أو تضيف سجلا.

يبين الشكل (c) النوع الثالث من بنى الملفات. يتألف الملف في هذا التنظيم من شجرة سجلات لا تكون سجلاتها متساوية الحجم بالضرورة يحوي كل سجل حقل مفتاح موجود في مكان ثابت من السجل. ترتب الشجرة حسب حقل المفتاح للسماح بالبحث السريع عن مفتاح معين.

العملية الأساسية هنا ليست الحصول على السجل التالي على الرغم من إمكانية القيام بذلك بل الحصول على السجل الذي يحوي مفتاحا معيناً. من اجل ملف حديقة الحيوانات المبين في الشكل (c) يمكن الطلب من نظام التشغيل أن نحصل على السجل الذي مفتاحهpony (حصان صغير) مثلا دون الاهتمام بموقعه الصحيح في الملف. أضف إلى ذلك يمكن إضافة سجلات جديدة حين يقرر نظام التشغيل وليس المستخدم المكان الذي سيوضع فيه السجل. من الواضح أن هذا النوع من الملفات مختلف جدا عن الملفات غير المهيكلة المستخدمة في UNIX وWindows لكنها مستخدمة بشكل واسع في الحواسب الكبيرة main frames التي ما زالت مستخدمة لمعالجة البيانات التجارية.

6 .1 3 -أنواع الملفات

تدعم العديد من أنظمة التشغيل أنواعا متعددة من الملفات. مثلا يملك UNIX وWindows ملفات نظامية وفهارس. يوجد في UNIX أيضاً ملفات خاصة محرفية وكتلية. الملفات النظامية هي الملفات التي تحوي معلومات المستخدم. جميع الملفات المبينة في الشكل السابق هي ملفات نظامية. الفهارس هي ملفات نظام تقوم بتنظيم بنية نظام الملفات. سندرس الفهارس بعد قليل. الملفات المحرفية الخاصة تتعلق بالدخل والخرج وتستخدم لنمذجة أجهزة الدخل\الخرج التسلسلية مثل الطرفيات والطابعات والشبكات. تستخدم الملفات الخاصة الكتلية لنمذجة الأقراص وسنهتم بشكل أساسي بالملفات النظامية.

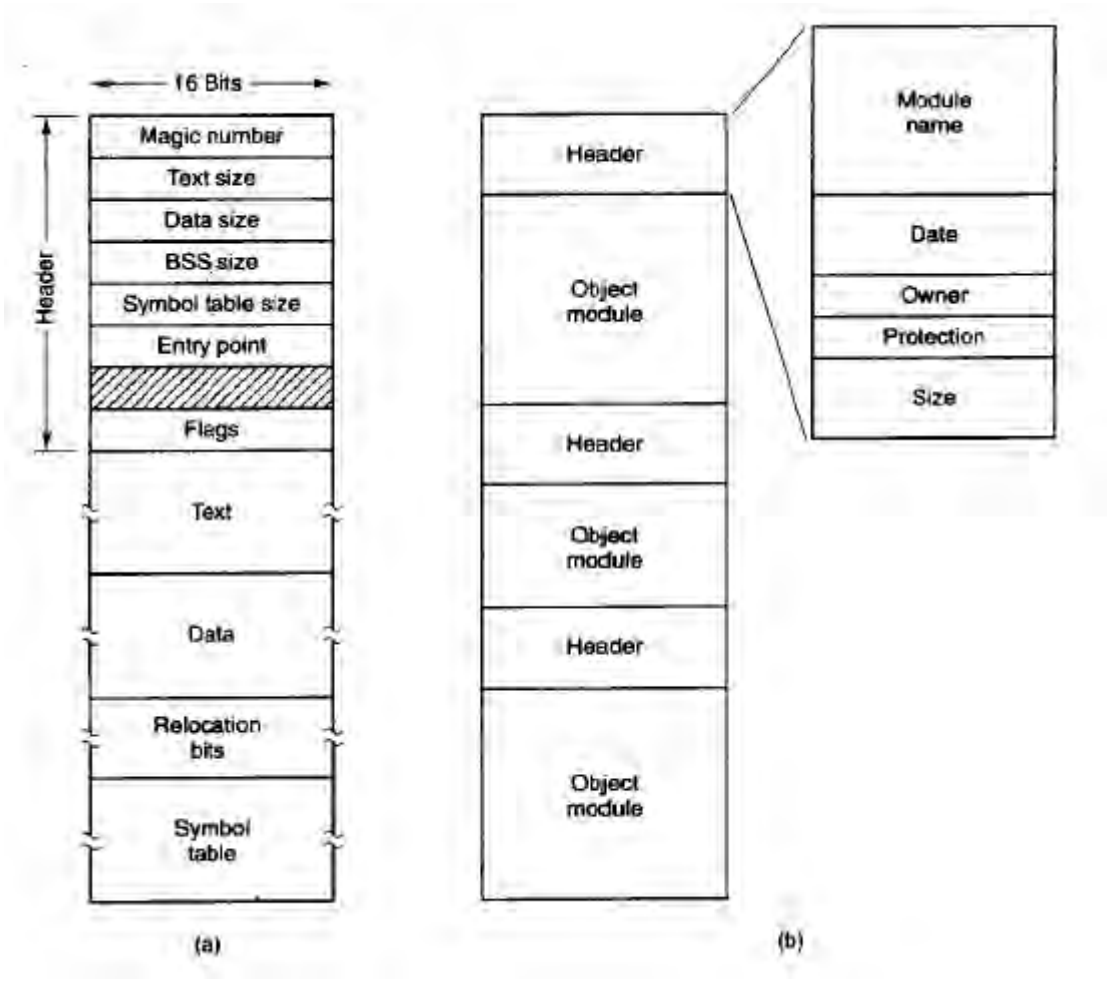
الملفات النظامية بشكل عام إما أن تكون ملفات ASCII أو ملفات ثنائية. تتألف ملفاتASCII من أسطر من النص. ينتهي كل سطر في بعض الأنظمة بمحرف رجوع الحامل وينتهي في بعضها الآخر بمحرف تغذية سطر. وتستخدم بعض الأنظمة مثل MS-DOS كلا المحرفين. ليس من الضروري أن تكون الأسطر متساوية الطول.

الميزة الكبرى لملفات ASCII أنها يمكن عرضها وطباعتها كما هي ويمكن تحريرها بأي محرر نصوص. بالإضافة إلى ذلك إذا كان عدد كبير من البرامج تستخدم ملفات ASCII للدخل والخرج من السهل وصل خرج أحد البرامج بدخل برنامج آخر كما في أنابيب سطر الأوامرpipes. أن هذا لا يسهل أبدا عملية الوصل لكنه يسهل تفسير المعلومات بسبب استخدام اصطلاح معياري مثل ASCII للتعبير عن المعلومات.

إن الملفات الأخرى هي ملفات ثنائية وهذا يعني إنها ليست ملفا ت ASCII. تعطي طباعة هذه الملفات على الطابعة خرجا غير مفهوم مليئا بالرموز الغريبة والعشوائية. عادة تكون البنية الداخلية لهذه الملفات معروفة فقط للبرامج التي تستخدمها. في الشكل (A) مثلا نرى ملفاً ثنائياً تنفيذياً بسيطاً مأخوذاً من أحد إصدارات UNIX على الرغم من أن الملف عبارة عن تسلسل من البايتات من الناحية التقنية إلا أن نظام التشغيل لن ينفذ البرنامج إلا إذا كان بالتنسيق الصحيح. ويحوي الملف خمسة أقسام: هي الرأس، النص، البيانات، بتات التوضع، جدول الرموز. يبدأ الرأس برقم يسمى الرقم السحري الذي يميز الملف على انه ملف تنفيذي (لتفادي التنفيذ غير المقصود لملف لا يملك هذا التنسيق) تأتي بعد ذلك أحجام الأجزاء المختلفة للملف

وعنوان بدء التنفيذ وبعض بتات الأعلام. يلي الرأس نص البرنامج نفسه وبياناته تحمل هذه الأجزاء إلى الذاكرة وتوضع باستخدام بتات التوضع. ويستخدم جدول الرموز من اجل التنقيح.

مثالنا الثاني عن الملفات الثنائية هو أرشيف مضغوط أيضاً من UNIX. يتألف هذا الأرشيف من مجموعة إجراءات مكتبة (وحدات نمطية) مترجمة لكنها ليست مربوطة. تسبق كل واحدة منها براس يدل على اسمها وتاريخ إنشاءها ومالكها وشفرة الحماية وحجمها. وكما هي الحال في الملف التنفيذي فان رؤوس الوحدات النمطية مليئة بالأرقام الثنائية.يؤدي نسخها إلى الطابعة إلى كتابة غير مضمومة.



الشكل 41

قد يسبب مثل هذا التنظيم القوي لأنواع الملفات مشاكل عندما يقوم المستخدم بأي شيء لم يتوقعه مصممو النظام.لندرس كمثال نظاما تملك فيه ملفات الخرج الامتداد (dat). إذا كتب مستخدم برنامج تنسيق يقرأ ملفات (c)برامج بلغة C ويحولها مثلا إلى تنسيق مسافات بادئة قياسي ثم يكتب الملف المنسق كخرج بامتداد dat إذا حاول المستخدم تقديم هذا الملف إلى مترجم C لترجمته فان النظام سيرفض ذلك لأنه يحمل امتدادا غير مناسب.كما أن محاولة نسخ file.dat إلى file.c ستفشل لان النظام سيعتبرها غير صحيحة(لحماية المستخدم من الأخطاء).

4.1.6 -الوصول إلى الملفات

قدمت أنظمة التشغيل الأولى نوعاً واحداً فقط من الوصول إلى الملفات وهو الوصول التسلسلي. تستطيع العملية في هذه الأنظمة قراءة جميع البايتات أو السجلات في الملف بالترتيب ابتداء من بداية الملف، لكنها لا تستطيع اجتياز بعضها وقراءتها بترتيب مختلف. إلا أنه من الممكن إعادة لف الملف التسلسلي إلى البداية وقراءته مرات ومرات حسب الحاجة. كانت الملفات التسلسلية مناسبة عندما كان وسط التخزين عبارة عن أشرطة مغناطيسية وليس أقراصاً. عندما بدأ استخدام الأقراص لتخزين الملفات أصبح من الممكن قراءة البايتات أو السجلات من الملف دون ترتيب أو الوصول إلى سجل بواسطة المفتاح عوضاً عن الموقع تسمى الملفات التي يمكن قراءة بايتاتها أو سجلاتها بأي ترتيب بملفات الوصول العشوائي وهي مطلوبة من قبل العديد من البرامج التطبيقية. تعتبر الملفات ذات الوصول العشوائي أساسية للعديد من التطبيقات مثل أنظمة قواعد البيانات إذا اتصل زبون بمكتب طيران وطلب حجز مقعد في رحلة معينة يجب أن يستطيع برنامج الحجز الوصول إلى سجل هذه الرحلة دون الاضطرار لقراءة سجلات آلاف الرحلات الأخرى.

هناك طريقتان لتحديد مكان بدء القراءة. تعتمد الطريقة الأولى على إعطاء كل عملية read المكان الذي يراد القراءة منه. أما الطريقة الثانية فتعتمد على عملية خاصة اسمها seek مهمتها تحديد الموقع الحالي. بعد إجراء seek يمكن قراءة الملف بشكل تسلسلي ابتداء من الموقع الحالي الجديد. في بعض أنظمة تشغيل الحواسب الكبيرة القديمة تصنف الملفات إلى ملفات تسلسلية و ملفات عشوائية عند إنشائها. يسمح ذلك لنظام التشغيل باستخدام تقنيات تخزين مختلفة لكل من الصنفين. لا تميز أنظمة التشغيل الحديثة بين نوعي الملفات فجميع الملفات تكون ذات وصول عشوائي تلقائياً.

5.1.6 سمات الملفات

كل ملف له اسم ويحوي بيانات بالإضافة إلى ذلك تربط جميع أنظمة التشغيل معلومات أخرى بكل ملف مثل تاريخ ووقت إنشاء الملف وحجم الملف. سنسمي هذه المعلومات الإضافية بسمات الملف Attributes. تختلف لائحة السمات بشكل كبير من نظام تشغيل إلى آخر. يبين الجدول في الشكل التالي بعض السمات الممكنة ولكن هناك سمات أخرى موجودة أيضاً. لا يوجد أي نظام حالي يملك جميع هذه السمات لكن كلاً منها يملك بعضها.

تتعلق السمات الأربع الأولى بحماية الملف وتحدد من يمكنه الوصول إليه ومن لا يمكنه ذلك. في بعض الأنظمة يجب على المستخدم أن يقدم كلمة مرور للوصول إلى الملف. وفي هذه الحالة يجب أن تكون كلمة المرور من سمات الملف. إن الأعلام عبارة عن بتات أو حقول قصيرة تتحكم أو تفعل بعض الخصائص المعينة. الملفات المخفية مثلا لا تظهر عند سرد جميع الملفات. علم الأرشيف يحدد إذا ما كان الملف قد تم نسخه احتياطيا أم لا. يقوم برنامج النسخ الاحتياطي بتصفير هذا البت ويقوم نظام التشغيل بتأهيله عندما يتغير الملف. يستخدم علم الملف المؤقت لتعليم الملف انه مؤقت وعندما يتم حذف الملف عند انتهاء العملية التي أنشأتها تتواجد سمات طول السجل وموقع المفتاح وطول حقل المفتاح فقط في الملفات التي يمكن الحصول على سجلاتها بالمفتاح وتقدم المعلومات المطلوبة لإيجاد المفتاح.

تحدد السمات المتعلقة بالوقت زمن إنشاء الملف وعملية الوصول الأخير والتعديل الأخير. تفيد هذه المعلومات لأغراض متنوعة مثلاً يحتاج الملف المصدري الذي تم تعديله بعد إنشاء الملف الهدفي الموافق له إلى إعادة ترجمة تقدم هذه الحقول المعلومات المطلوبة لإيجاد المفتاح. تحديد الحجم الأعظمي للملف عند إنشائه وذلك كي يحجز الكمية العظمية اللازمة

للتخزين مقدما أنظمة التشغيل للمحطات العمل والحواسب الشخصية أزكى من أن تحتاج لهذه الميزة.

Field	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	Id of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

الشكل 42

6 2 ـالفهارس

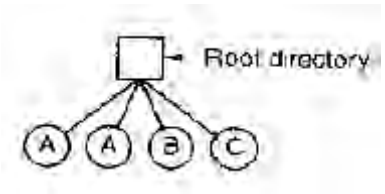
لتنظيم الملفات, تحتوي أنظمة الملفات عادة فهارس (Directories) أو مجلدات (Folders) والتي تكون في معظم الأنظمة عبارة عن ملفات أيضاً.

6 2 1 ـأنظمة الفهارس ذات المستوى الواحد

الشكل الأبسط لأنظمة الفهارس هو وجود فهرس واحد يحوي جميع الملفات. يدعى هذا الفهرس أحياناً بالفهرس الجذر Root Directory لكن باعتبار أنه الفهرس الوحيد فان الاسم لا يهم كثيراً. كان هذا النظام شائعاً في أنظمة الحواسب الشخصية الأولى وذلك لأنه لا يوجد إلا مستخدم واحد فقط.

يبين الشكل التالي مثالاً عن نظام ذي فهرس واحد. يحوي الفهرس هنا أربعة ملفات. يبين الشكل مالكي الملفات وليس أسماء الملفات (لأن مالكي الملفات مهمون بالنسبة للموضوع

الذي نحن بصدده) تتمثل محاسن هذا النموذج في بساطته وقدرته على إيجاد الملفات بسرعة لأنه يوجد فقط مكان واحد للبحث فيه.

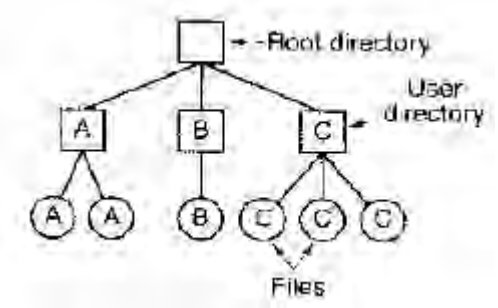


الشكل 43

المشكلة في وجود فهرس واحد فقط في نظام متعدد المستخدمين أن المستخدمين المختلفين قد يستخدمون بالصدفة نفس الأسماء لملفاتهم. مثلا, إذا أنشئ المستخدم A ملفا اسمه mailbox ثم أنشأ المستخدم B بعد ذلك, ملفا اسمه أيضاً mailbox أيضاً. فإن ملف المستخدم B سيكتب فوق الملف A نتيجة لذلك, لم تعد هذه الطريقة مستخدمة في الأنظمة المضمنة الصغيرة.

2 2 6 -أنظمة الفهرس ذات المستويين

لتجنب المتعارضات الناتجة عن تسمية مستخدمين مختلفين ملفاتهم بنفس الاسم يعطى كل مستخدم فهرساً خاصاً به، بهذه الطريقة لا تتداخل الأسماء التي يعطيها أحد المستخدمين لملفاته مع أسماء ملفات مستخدم آخر ولا يوجد مشكلة في وجود نفس اسم الملف في فهرسين مختلفين أو أكثر يبين الشكل التالي هذا التصميم. يمكن استخدام هذا التصميم مثلا في حاسب متعدد المستخدمين أو في شبكة صغيرة من الحواسيب الشخصية التي تتشارك بمخدم ملفات مشترك عبر شبكة محلية.



الشكل 44

عندما يحاول مستخدم فتح ملف في هذا النظام يعرف النظام بشكل ضمني أي فهرس يجب البحث فيه. وبالتالي نحتاج إلى شكل ما من إجراءات تسجيل الدخول حيث يحدد المستخدم اسم تسجيل دخول أو تعريف وهو شيء غير مطلوب في أنظمة الفهرس الوحيد المستوى. عند تحقيق هذا النظام في أبسط أشكاله لا يستطيع المستخدمون إلا الوصول إلى الملفات الموجودة في فهارسهم الخاصة بهم. إلا أن هناك تعديلا طفيفا ممكنا يتمثل بالسماح للمستخدمين بالوصول إلى ملفات المستخدمين الآخرين بإعطاء دلالة ما عن المستخدم الذي يملك المراد فتحه مثلا. يمكن استدعاء:

```
Open("x")
```

لفتح ملف x في فهرس المستخدم واستدعاء:

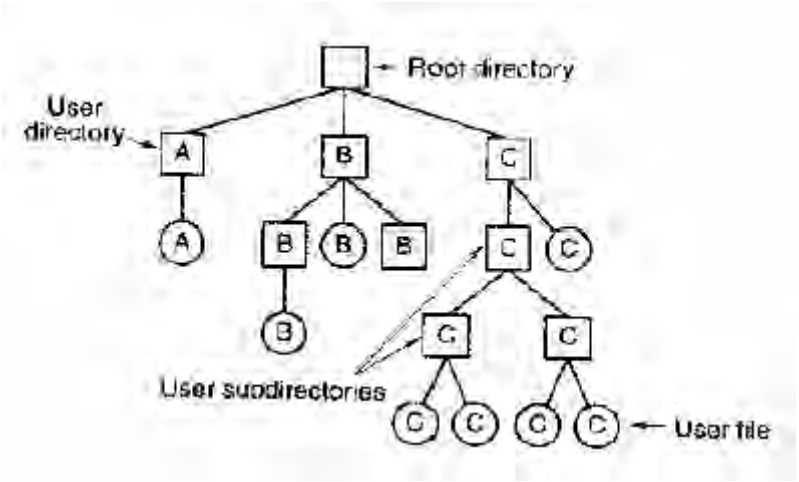
```
Open("Sammer/x")
```

لفتح ملف x في فهرس مستخدم آخر اسمه "Sammer"

من الحالات التي يحتاج فيها المستخدم إلى الوصول إلى ملفات غير ملفاته الخاصة هي تنفيذ برامج النظام الثنائية.إن وجود نسخ من جميع البرامج الخدمية في كل فهرس غير مجد على الإطلاق. يجب على الأقل السماح بالوصول إلى فهرس النظام الذي يحوي البرامج التنفيذية الثنائية.

6 2 3 -أنظمة الفهارس الهرمية

يتخلص نظام الفهارس ذو المستويين من تعارض الأسماء بين المستخدمين لكنه ليس مرضياً للمستخدمين الذين يملكون عدداً كبيراً من الملفات. حتى في الحواسب الشخصية ذات المستخدم الواحد يعتبر هذا النظام غير ملائم. من الشائع أن يرغب المستخدمون بتجميع ملفاتهم ضمن مجموعات منطقية. فالمدرس مثلا قد يملك مجموعة من الملفات التي تشكل مع بعضها كتابا من أجل منهاجه ومجموعة أخرى تحوي برامج الطلاب المرسلة من أجل منهاج آخر ومجموعة ثالثة من الملفات تحوي شفرة لنظام متقدم لكتابة المترجمات يقوم ببنائه بالإضافة إلى ملفات أخرى للبريد الالكتروني والمواعيد والأبحاث التي يكتبها والألعاب وغيرها يتطلب الأمر طريقة لتجميع هذه الملفات مع بعضها بطريقة مرنة يختارها المستخدم. إن ما نحتاجه هو النظام هرمي عام (أي شجرة من الفهارس). بهذه الطريقة يستطيع كل مستخدم إنشاء أي عدد يريد من الفهارس بحيث يمكنه تجميع الملفات بطريقة طبيعية. يبين الشكل التالي هذا الأسلوب. الفهارس A, B, C موجودة في الفهرس الجذر وينتمي كل منها إلى مستخدم مختلف اثنان منهما أنشئ فهارس فرعية للمشاريع التي يعملون عليها.



الشكل 45

تؤمن إمكانية إنشاء المستخدمين عدداً غير محدد من الفهارس الفرعية أداة هيكلية قوية للمستخدمين لتنظيم عملهم لهذا السبب تنظم جميع أنظمة الملفات الحديثة تقريبا بهذه الطريقة.

4 2 6 -أسماء المسارات

عندما ينظم نظام الملفات على شكل شجرة من الفهارس نحتاج إلى طريقة ما لتحديد أسماء الملفات. هناك طريقتان شائعتا الاستخدام، تتمثل الطريقة الأولى بإعطاء كل ملف اسم مسار مطلق absolute path name مؤلف من المسار ابتداء من الفهرس الجذر حتى الملف. كمثال عن ذلك نأخذ المسار usr/ast/mailbox ويعني أن الفهرس الجذر يحوي فهرسا فرعيا اسمه usr الذي يحوي بدوره الفهرس الفرعي ast والذي يحوي الملف mailbox. تبدأ أسماء المسارات دائما بالفهرس الجذر وتكون فريدة (متمايزة عن بعضها).تفضل مكونات المسار في نظام UNIX بالرمز /. أما في Windows فيستخدم الرمز \. وكان في mutics الرمز>. وبالتالي يكتب نفس اسم المسار في الأنظمة الثلاثة كما يلي:

```
Windows        \usr\ast\mailbox
Unix           /usr/ast/mailbox
 Multics       >usr>ast>mailbox
```

بغض النظر عن الرمز المستخدم إذا كان المحرف الأول هو الرمز الفاصل فهذا يعني أن المسار مطلق.النوع الثاني من الأسماء هي اسم المسار النسبي relative path name.تستخدم هذه الأسماء بالارتباط مع مفهوم فهرس العمل working directory والذي يسمى أحياناً بالفهرس الحالي وفي هذه الحالة تنسب جميع المسارات التي لا تبدأ بالفهرس الجذر إلى الفهرس الحالي. مثلاً إذا كان الفهرس الحالي هو /usr/ast/mailbox يمكن الإشارة إليه ببساطة بالاسم mailbox. بعبارة أخرى يعتبر أمر UNIX التالي:

```
Cp     /usr/ast/mailbox     /usr/ast/mailbox.bak
```

والأمر:

```
Cp     mailbox     mailbox.bak
```

متكافئين تماما إذا كان الفهرس الحالي هو /usr/ast. يعتبر الشكل النسبي عادة أنسب لأنه اقصر ويقوم بنفس العمل الشكل المطلق.تحتاج بعض البرامج للوصول إلى ملف معين بغض النظر عن الفهرس الحالي.في هذه الحالة يجب أن يستخدم البرنامج دائما اسم المسار المطلق. مثلا قد يحتاج برنامج مدقق إملائي إلى القراءة من الملف /usr/lib/dictionary للقيام بعمله.يجب أن يستخدم اسم المسار المطلق الكامل في هذه الحالة لأنه لا يعرف ما هو المسار الحالي عندما يستدعي. إن استخدام المسار المطلق يعمل دائما بغض النظر عن المسار الحالي. لكن إذا كان المدقق الإملائي يحتاج للكثير من الملفات الموجودة في /usr/lib يمكنه استخدام طريقة أخرى تتمثل بتنفيذ استدعاء نظام لتغيير الفهرس الحالي إلى /usr/lib عندها يستطيع استخدام الاسم dictionary كبارامتر أول للاستدعاءopen. إن تغيير الفهرس الحالي صراحة يضمن للبرنامج معرفة مكانه بدقة ضمن شجرة الفهارس وبذلك يستطيع استخدام المسارات النسبية.

كل عملية لها فهرس حالي خاص بها، لذلك عندما تغير العملية فهرسها الحالي ثم تخرج فإن العمليات الأخرى لا تتأثر بذلك ولا تبقى أي آثار لهذا التغيير في نظام الملفات. بهذه الطريقة يكون تغيير الفهرس الحالي للعملية آمنا دائما متى كان ذلك مناسبا. من ناحية أخرى إذا غير احد الإجراءات في مكتبة ما الفهرس الحالي وعاد إلى مستدعيه دون إعادة الفهرس الحالي إلى ما كان عليه فان ما تبقى من البرنامج قد لا يعمل بشكل صحيح لأنه يفترض انه موجود في

مكان ما بينما هو في مكان آخر في شجرة الفهارس. لهذا السبب من النادر أن تقوم إجراءات المكتبة بتغيير الفهرس الحالي وعندما تضطر لذلك فإنها تعيده إلى ما كان عليه قبل الرجوع.

تمتلك معظم أنظمة التشغيل التي تدعم نظام الفهرس الهرمي سجلين خاصين في كل فهرس هما "."و".."(النقطة والنقطتين).تشير النقطة إلى الفهرس الحالي وتشير النقطتان إلى الفهرس الأب.كي نرى كيف تعمل هذه النقاط لندرس شجرة الملفات المبينة في الشكل السابق.توجد عملية معينة فهرسها الحالي هوusr/ast/.تسطيع هذه العملية استخدام ".." للانتقال إلى المستوى الأعلى في الشجرة. مثلا تستطيع نسخ الملف usr/lib/dictionary/.. إلى فهرسها الخاص باستخدام الأمر:

```
Cp    ../lib/dictionary.
```

يشير المسار الأول للنظام كي ينتقل للفهرس الأب (الفهرس usr) ثم يذهب إلى الفهرس lib لإيجاد الملف dictionary. ويشير المسار الثاني (النقطة) للفهرس الحالي. عندما يرى الأمرcp اسم فهرس (بما في ذلك النقطة) كبارامتر أخير فانه يقوم بنسخ جميع الملفات فيه. طبعا يمكن نسخ الملفات بالطريقة الاعتيادية بواسطة المسارات المطلقة:

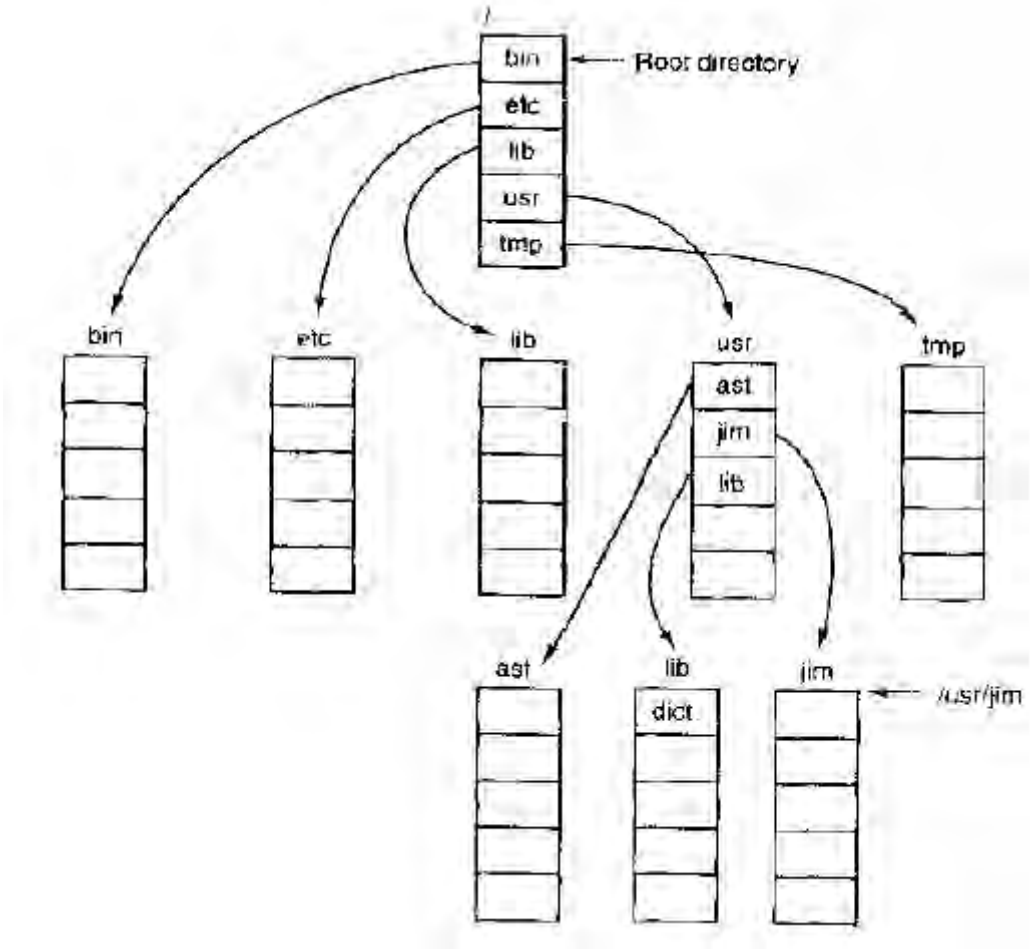
```
Cp    /usr/lip/dictionary.
```

إن استخدام كتابة هنا يوفر المستخدم عناء كتابة dictionary مرة أخرى.كذلك فان كتابة:

```
Cp    /usr/lib/dictionary dictionary
```

وكذلك:في بالغرض.وكذلك:

جميع هذه الأسطر تقوم بنفس العمل تماماً.



الشكل 46

6 2 5 -تنظيم نظام الملفات

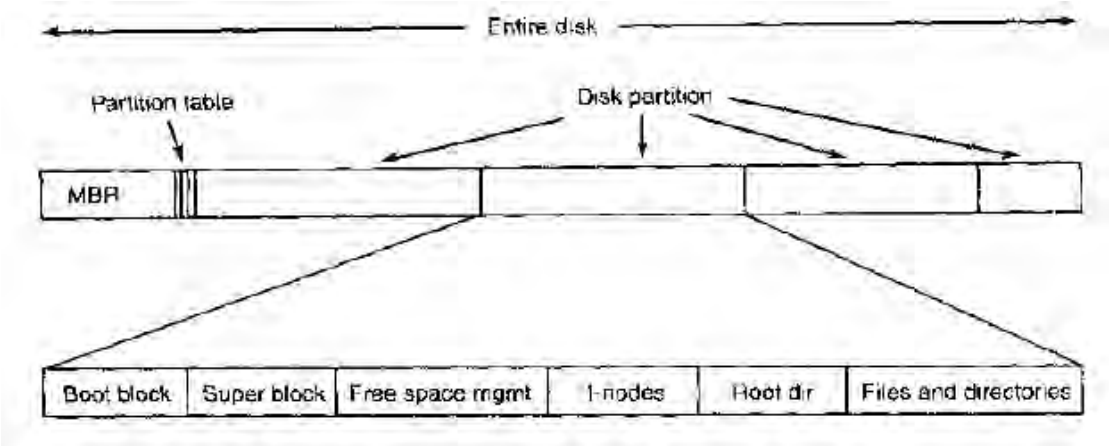
تخزن أنظمة الملفات على الأقراص. معظم الأقراص يمكن تقسيمها إلى قسم واحد يسمى Partition أو عدة أقسام حيث يخزن في كل منها نظام ملفات مستقل. يسمى القطاع 0 على القرص بسجل الإقلاع الرئيسي MBR (Master Boot Record) ويستخدم لإقلاع الحاسب. يحوي MBR في نهايته جدول الأقسام.يعطي هذا الجدول عناوين بداية ونهاية كل قسم.يعلم أحد الأقسام في القرص على أنه نشط. عندما يقلع الحاسب يقرأ BIOS قطاع MBR وينفذه.

أول شي يقوم به برنامج MBR هو تحديد مكان القسم النشط وقراءة الكتلة الأولى منه، و تسمى كتلة الإقلاع (Boot Block) ثم يقوم بتنفيذها،.يحمل البرنامج الموجود في كتلة إقلاع حتى أن لم يكن يحوي نظاما قابلا للإقلاع.بالإضافة إلى ذلك قد يوضع فيه نظام تشغيل في المستقبل لذلك من الأفضل حجز كتلة إقلاع في جميع الأحوال.

فيما عدا ابتداء كل قسم بكتلة إقلاع يختلف تنظيم القسم بشكل كبير من نظام ملفات إلى آخر.يحوي نظام الملفات غالبا بعض العناصر المبينة في الشكل التالي. العنصر الأول هو الكتلة الكبرى(super block).تحوي هذه الكتلة البارامترات الأساسية المتعلقة بنظام الملفات وتقرا إلى الذاكرة الرئيسية عند إقلاع الحاسب أو عند الوصول إلى نظام الملفات لأول مرة. تشمل

المعلومات النموذجية المحتواة في الكتلة الكبرى رقما سحريا لتعريف نوع نظام الملفات وعدد الكتل في نظام الملفات وغيرها من المعلومات الإدارية الهامة.

يأتي بعد ذلك معلومات عن الكتل الحرة في نظام الملفات على شكل صورة نقطية أو لائحة من المؤشرات.قد يأتي بعد ذلك عدد من العقد التي تسمى i-node على شكل مصفوفة من بني البيانات، بنية لكل ملف. تحوي جميع المعلومات المتعلقة بالملفات. وبعد ذلك يأتي الفهرس الجذر والذي يحوي قمة شجرة نظام الملفات. أخيرا يحوي ما تبقى من القرص جميع الفهارس والملفات الأخرى.



الشكل 47

6 2 6 -تحقيق الملفات

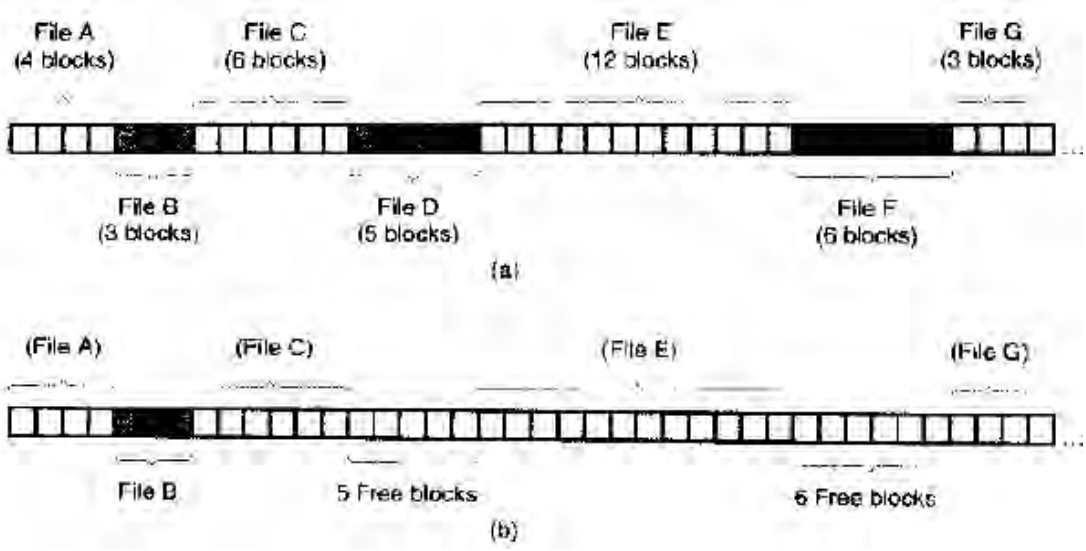
لعل الموضوع الأكثر أهمية في مسألة تحقيق تخزين الملفات هو كيفية إدارة العلاقة بين كتل القرص والملفات، بعبارة أخرى كيفية تحديد الكتل التي تشكل ملف ما. هناك عدة طرق متنوعة مستخدمة في عدة أنظمة تشغيل سندرس في هذا الفصل كلا منها على حدة.

6 2 6 1- التخصيص المتجاور

تتمثل الطريقة الأسهل لتخصيص الملفات بتخزين كل ملف على شكل تسلسل من الكتل المتجاورة. وبالتالي يحتاج ملف حجمه 50 كيلوبايت في قرص ذي كتل حجمها 1 كيلوبايت إلى حجز 50 كتلة متجاورة. وإذا كان حجم الكتلة 2 كيلوبايت فانه سيحتجز 25 كتلة متجاورة.

نرى في الشكل التالي(A) مثالا عن تخصيص المساحة المتجاورة يبين الشكل الـ40 كتلة الأولى من القرص. ابتداء من الكتلة 0 على اليسار. في البداية يكون القرص فارغاً وبعد ذلك كتب ملف A طوله أربع كتل ابتداء من البداية (الكتلة0). بعد ذلك كتب ملف B ذو ست كتل ابتداء من الكتلة التي تلي A مباشرة. لاحظ أن كل ملف يبدأ عند بداية كتلة جديدة لذلك إذا كان الملف A طوله في الحقيقة $3^{1/2}$ كتلة، فان بعض المساحة ستهدر في نهاية الكتلة الأخيرة. يظهر في الشكل ما مجموعه سبعة ملفات يبدأ كل منها عند الكتلة التي تلي الكتلة الأخيرة من الملف السابق. استخدمنا التظليل فقط لتمييز الملفات عن بعضها.

يتميز التخصيص المتجاور لمساحة القرص بميزتين مهمتين:الأولى انه سهل التحقيق لان عملية تتبع مكان كتل الملف قد اختصرت بحيث نحتاج لتذكر رقمين فقط هما عنوان القرص للكتلة الأولى وعدد الكتل في الملف بعملية جمع بسيطة.



الشكل 48

تتمثل الميزة الثانية في الأداء الجيد عند القراءة لان الملف يمكن قراءته بأكمله من القرص بعملية واحدة. نحتاج فقط إلى حركة أذرع واحدة (إلى الكتلة الأولى).بعد ذلك لا حاجة لأي عمليات انتقال أو تأخيرات دوران لذلك تأتي البيانات بالسرعة الكاملة للقرص. نستنتج أن التخصيص المتجاور بسيط التحقيق، بسيط التحقيق، وعالي الأداء.

لسوء الحظ يعاني التخصيص المتجاور أيضاً من مساوئ هامة:وهي أن القرص يصبح مبعثرا مع الزمن.كي ترى كيف يحصل ذلك انظر إلى الشكل التالي (B). حذف هنا الملفان f و d. عندما يحذف ملف يتم تحرير كتلته مما يخلف سلسلة من الكتل الفارغة على القرص. لا يتم ضغط القرص للتخلص من هذه الفجوة لان هذا يعني نسخ جميع الكتل التي تلي الفجوة و التي ربما يبلغ عددها الملايين من الكتل.نتيجة لذلك يصبح القرص مؤلفاً من ملفات وفجوات كما هو موضح في الشكل.لا تشكل هذه البعثرة في البداية مشكلة حقيقية لان كل ملف جديد يمكن كتابته في نهاية القرص بعد الملف السابق.لكن القرص سيمتلئ في النهاية ويصبح من الضروري إما ضغط كتل القرص إلى جانب بعضها وهو خيار مكلف جدا أو إعادة استخدام المساحة الحرة الموجودة في هذه الفجوات. تتطلب إعادة استخدام المساحة تأسيس لائحة من الفجوات الفارغة وهذا يمكن إنجازه بسهولة. إلا انه عند إنشاء ملف جديد يجب معرفة حجمه النهائي لاختيار الفجوة التي يمكن أن تتسع له.

لنتخيل نتائج مثل هذا التصميم.يشغل المستخدم محرر نصوص لكتابة مستند، أول ما يطلبه البرنامج هو حجم المستند النهائي مقدرا بالبايتات.يجب الإجابة على هذا السؤال وإلا لن يتابع البرنامج العمل.إذا أتضح أن الرقم المعطى صغير جدا سينتهي البرنامج فجأة بسبب امتلاء فجوة القرص المخصصة له وعدم وجود أي مكان لتخزين بقية الملف. وإذا حاول المستخدم تجنب هذه الحالة بإعطاء حجم كبير جدا كحجم نهائي 100 ميغابايت مثلا فان محرر النصوص قد لا يجد أي فجوة بهذا الحجم ويعلن عن عدم إمكانية إنشاء الملف. طبعا يستطيع المستخدم إعادة

تشغيل البرنامج وإعطاء حجم 50 ميغابايت هذه المرة. وهكذا حتى إيجاد فجوة مناسبة. لكن هذا النظام يبقى غير مناسب ولا يجعل المستخدمين سعداء.

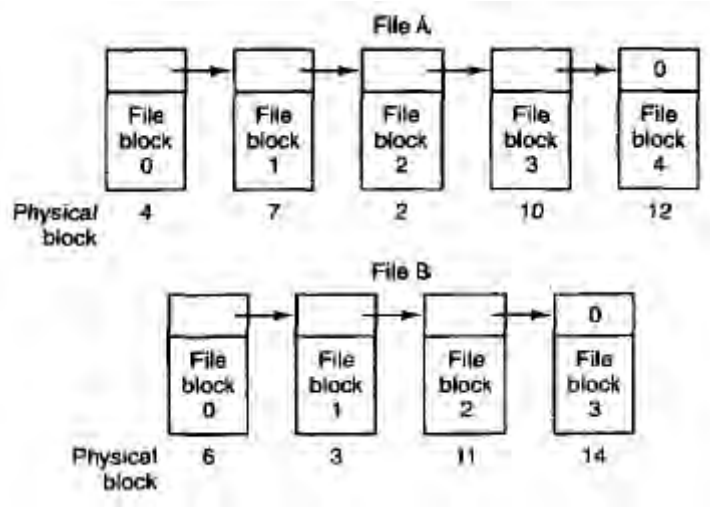
إلا أن هناك حالة واحد يكون فيها التخصيص المتجاور جيداً وواسع الاستخدام وهي حالة أقراص CD-ROM. جميع أحجام الملفات في هذه الحالة معروفة مسبقا ولن تتغير مع الاستخدام المتلاحق لنظام ملفات القرص المدمج.

كما ذكرنا في الفصل الأول التاريخ غالبا ما يعيد نفسه في علم الحاسب مع ظهور أجيال جديدة من التقنية. كان التخصيص المتجاور مستخدما في الواقع في أنظمة ملفات الأشرطة المغناطيسية قبل سنين بسبب بساطته وأدائه الجيد (لم تكن سهولة الاستخدام عاملا مهما آنذاك).انخفضت شعبية هذه الفكرة بعد ذلك بسبب الحاجة لتحديد حجم الملف عند إنشائه.لكن مع ظهور أقراص CD-ROM DVD وغيرها من الوسائط الضوئية التي تكتب لمرة واحدة فقط أصبحت الملفات المتجاورة جيدة مرة أخرى. لذلك من المهم دراسة الأنظمة القديمة والأفكار التي كانت بسيطة وجيدة لأنها قد تصبح قابلة للتطبيق في الأنظمة المستقبلية بطرق مدهشة.

6 2 2 6 -التخصيص باللائحة المترابطة

تتمثل الطريقة الثانية لتخزين الملفات بتخزين كل ملف على شكل لائحة مترابطة من كتل القرص كما في الشكل التالي. تستخدم الكلمة الأولى من كل كتلة كمؤشر إلى الكتلة التالية وتخزن البيانات فيما تبقى من الكتلة.على العكس من التخصيص المتجاور يمكن في هذه الطريقة استخدام كل كتلة في القرص. لا تضيع أي مساحة من القرص بسبب البعثرة (ما عدا البعثرة الداخلية في الكتلة الأخيرة من الملف).كما انه يكفي تخزين عنوان الكتلة الأولى فقط في الفهرس حيث يمكن إيجاد باقي المعلومات ابتداء من الكتلة الأولى.من ناحية أخرى على الرغم من أن القراءة التسلسلية بسيطة وسريعة، إلا أن الوصول العشوائي بطي جداً. فللوصول إلى الكتلة n يجب أن يبدأ نظام التشغيل من البداية ويقرا n-1 كتلة التي قبلها واحدة في كل مرة. من الواضح أن القيام بذلك عند كل عملية قراءة سيعطي أداء بطيئا جدا.

كما أن كمية البيانات المخزنة في كل كتلة لم تعد من مضاعفات العدد 2 لأن المؤشر يستهلك منها عدة بايتات. مع أن هذا ليس بالمشكلة الخطيرة لكنه أقل كفاءة لان العديد من البرامج تقرا وتكتب البيانات على شكل كتل حجمها من مضاعفات العدد2. بوجود المؤشر في البايتات الأولى من الكتلة، تتطلب عملية قراءة كتلة كاملة الحصول على المعلومات من كتلتين متتاليتين ثم لصق البيانات مع بعضها مما ينتج عبئاً كبيرا بسبب عملية النسخ.



الشكل 49تخصيص اللائحة المترابطة باستخدام جدول في الذاكرة:

يمكن التخلص من كلا سيئتي التخصيص باللائحة المترابطة بأخذ المؤشر من كل كتلة قرص ووضعها في جدول الذاكرة. يبين الشكل التالي كيف يبدو الجدول من أجل المثال المبين في الشكل السابق لدينا في كلا الشكلين ملفان. يستخدم الملف A كتل القرص 4-7-2-10-12 على الترتيب. ويستخدم الملف B كتل القرص 6-3-11-14 على الترتيب باستخدام الجدول في الشكل التالي يمكننا البدء بالكتلة 4 وتتبع السلسلة حتى نهايتها. يمكن فعل نفس الشيء ابتداء من الكتلة 6. تنتهي كلا السلسلتين بعلامة خاصة (مثلا1-) لا تكون رقم كتلة صالحا. يسمى مثل هذا الجدول بجدول توضع الملفات FAT (FILE ALLOCATION TABLE).

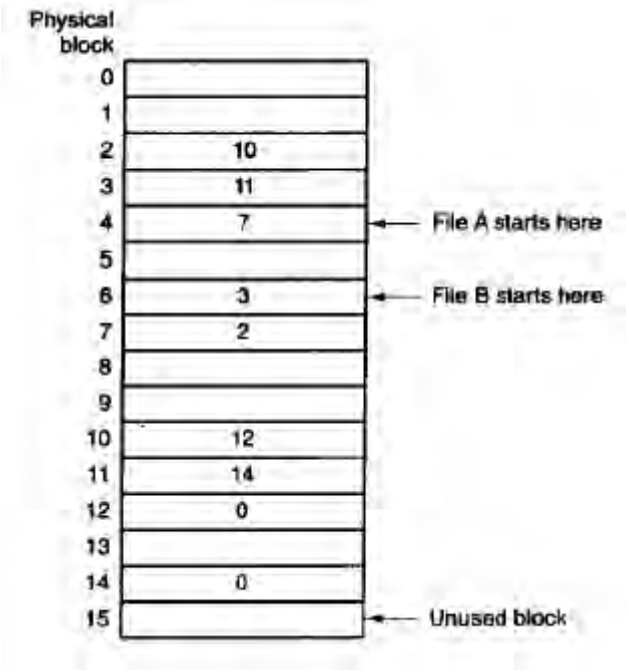
باستخدام هذا التنظيم تصبح الكتلة بأكملها متاحة لتخزين البيانات. بالإضافة إلى ذلك يصبح الوصول العشوائي أسهل بكثير. فعلى الرغم من الحاجة لتتبع السلسلة لإيجاد الإزاحة المعطاة ضمن الملف فإن السلسلة بأكملها موجودة في الذاكرة، لذلك يمكن تتبعها دون الرجوع إلى القرص. وكما في الطريقة السابقة، يكفي تخزين عنوان الكتلة الأولى في الفهرس حيث يمكن بواسطتها إيجاد جميع كتل الملف مهما كان حجمه.

السيئة الأساسية لهذا الطريقة أن الجدول بأكمله يجب أن يبقى في الذاكرة طوال الوقت كي تعمل من أجل قرص بسعة 20GB وحجم كتلة 1KB يحتاج الجدول 20مليون سجل. واحد لكل كتلة من 20 مليون كتلة في القرص. يجب أن يكون كل سجل بطول 3 بايت على الأقل وإذا كانت سرعة البحث ضرورية. يجب أن يكون بطول 4 بايت. وبالتالي سيحتل الجدول من 60MB حتى 80MB من الذاكرة الرئيسية حسب طريقة الأمثلة التي يتبعها النظام. إذا ما كانت تتبع للمساحة أم للزمن يمكن طبعا وضع هذا الجدول في ذاكرة قابلة للتصفيح. لكنها ستحتل مساحة كبيرة من الذاكرة الظاهرية ومساحة القرص بالإضافة إلى توليد حركة تصفيح إضافية

عقد i-node:

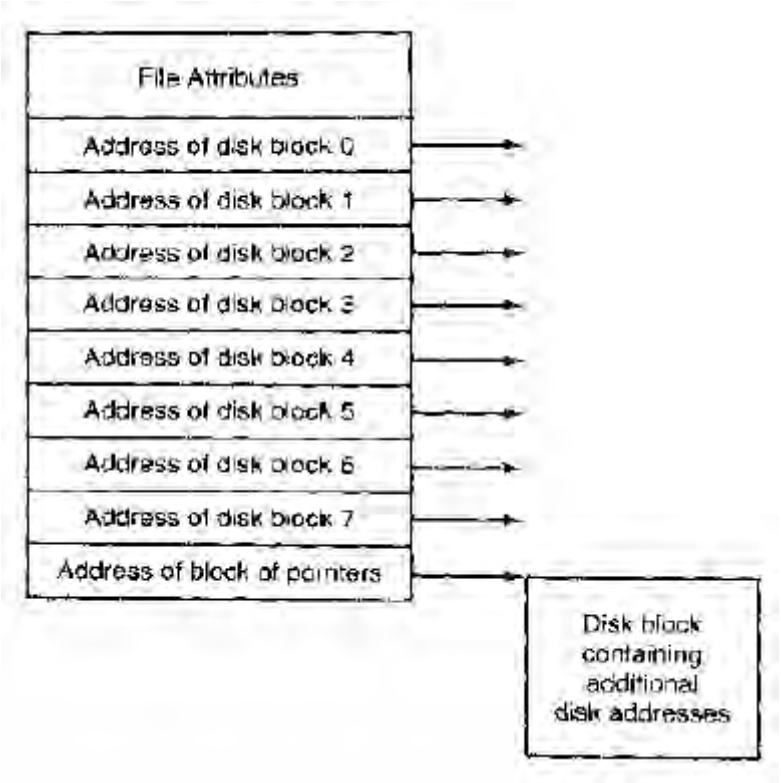
تتمثل الطريقة الأخيرة لتتبع أي كتل قرص تنتمي لأي ملف باقتران كل ملف ببنية بيانات تسمى i-node (عقد دليل INDEX-NODE) والتي تحتوي سمات وعناوين كتل الملف. يبين الشكل التالي مثالا بسيطاً. بوجود I-NODE لملف يمكن إيجاد جميع الكتل التابعة للملف. الميزة الكبيرة التي تمتلكها هذه الطريقة على الملفات المترابطة باستخدام جدول في الذاكرة إن I-NODE يجب أن تكون في الذاكرة فقط عندما يكون الملف الموافق لها مفتوحاً. إذا

كانت كل I-NODEتحتاج إلى n بايت و كان العدد الأعظمي للملفات التي يمكن فتحها في نفس الوقت هو k، فإن إجمالي الذاكرة المطلوبة للمصفوفة التي تحتوي الملفات المفتوحة هو kn. لا نحتاج إلا الحجز هذه الكمية من الذاكرة مسبقاً.



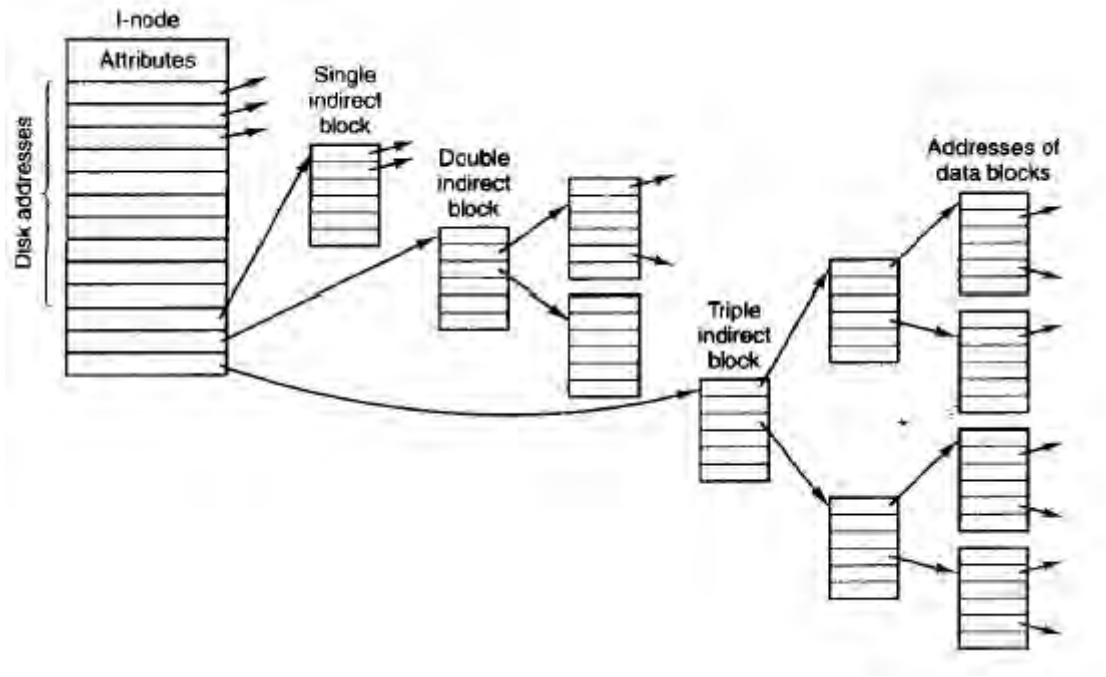
الشكل 50

عادة ما تكون هذه المصفوفة أصغر بكثير من الحجم الذي يحتاجه جدول الملفات الذي شرحناه في القسم السابق. والسبب بسيط يتناسب حجم الجدول المستخدم للاحتفاظ باللائحة المترابطة لجميع كتل القرص طردا مع حجم القرص نفسه، فإذا كان في القرص n كتلة يحتاج الجدول إلى n سجل.عندما يكبر حجم القرص يكبر معه حجم الجدول بشكل خطي. على العكس من ذلك تحتاج طريقة i-node إلى مصفوفة في الذاكرة حجمها متناسب مع العدد الأعظمي للملفات التي يمكن فتحها في وقت واحد. لا يهم إذا ما كان القرص حجمه 1GB أو10GB أو 100GB.



الشكل 51

إحدى المشاكل التي تصادفنا في i-node أنه إذا كانت كل عقدة فيها مساحة كافية لعدد ثابت من عناوين القرص ماذا يحدث إذا كان حجم الملف اكبر من هذا الحد؟ من الحلول الممكنة لهذه الطريقة عدم حجز عنوان القرص الأخير من اجل كتلة بيانات بل من اجل عنوان كتلة تحوي عناوين كتل قرص إضافية كما في الشكل التالي. كما يمكن وجود اثنتين أو أكثر من هذه الكتل، بحيث تحوي عناوين قرص أو كتل قرص تشير إلى كتل قرص أخرى فيها عناوين.

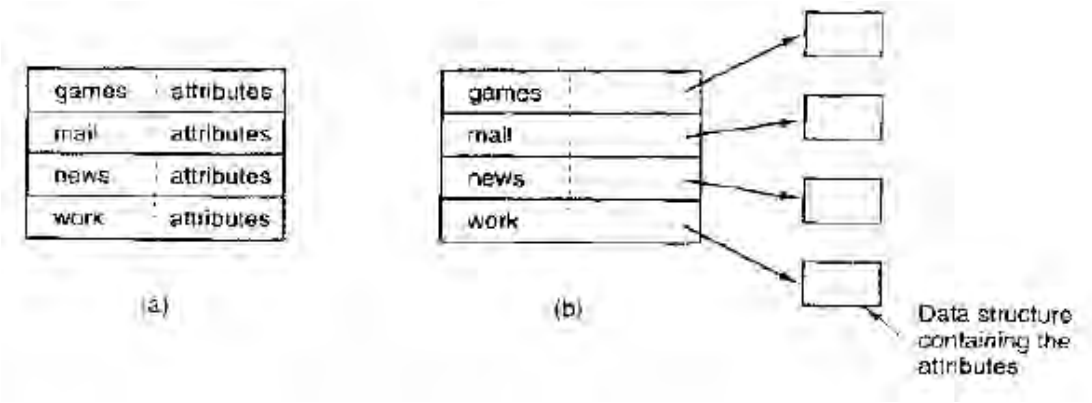


الشكل 52

6 2 7 -تحقيق الفهارس

قبل التمكن من قراءة ملف يجب أن يفتح. وعند فتح ملف يستخدم نظام التشغيل اسم المسار المزود من قبل المستخدم لتحديد سجل الفهرس الموافق للملف. يقدم سجل الفهرس المعلومات اللازمة لإيجاد كتل القرص. حسب النظام قد تكون هذه المعلومات إما عنوان القرص للملف بأكمله (تخصيص متجاور) أو رقم الكتلة الأولى (طريقتي تخصيص اللائحة المترابطة) أو رقم i-node. في جميع الحالات المهمة الأساسية لنظام الفهارس هي تحويل اسم الملف النصي إلى المعلومات اللازمة لإيجاد البيانات.هناك أيضاً موضوع هام متعلق بما ذكرناه وهو أين سيتم تخزين السمات. يحوي كل نظام ملفات سمات لكل ملف مثل مالك الملف وزمن الإنشاء ويجب تخزين هذه السمات في مكان ما.

إحدى الإمكانيات البسيطة هي تخزينها مباشرة في سجل الفهرس. تقوم العديد من الأنظمة بذلك.يبين الشكل التالي(a)هذا الخيار. يتألف الفهرس في هذا التصميم البسيط من لائحة السجلات المتساوية الحجم، سجل واحد لكل ملف. ويحوي اسم الملف (ذو طول ثابت) وبنية تحوي سمات الملف وعنوان قرص أو أكثر (حتى حد أعظمي معين) لتحديد مكان كتل قرص.

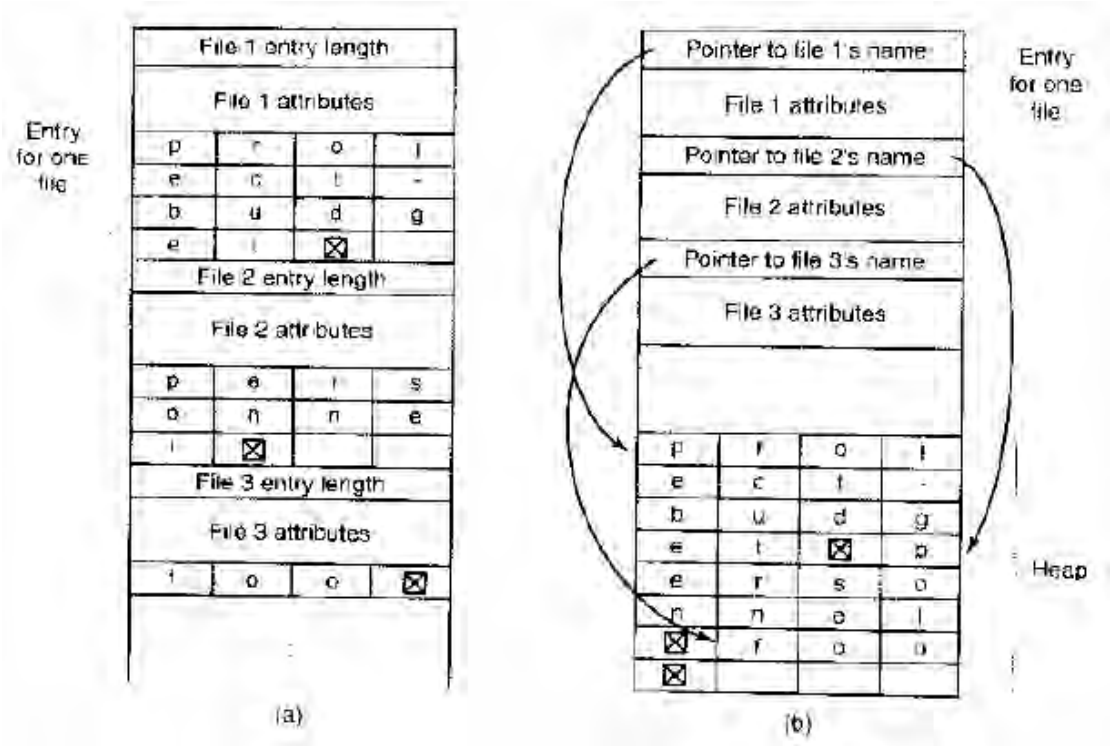


الشكل53

في الأنظمة التي تستخدم i-node عوضا عن تخزينها في سجلات الفهرس. في هذه الحالة يصبح سجل اقصر حيث يحوي اسم الملف ورقم i-node فقط.يوضح الشكل السابق(b)هذه الطريقة. تقابل الطريقتان المبينتان في الشكل السابق نظامي MS-DOS و UNIX على الترتيب.افترضنا حتى الآن أن الملفات لها أسماء قصيرة ثابتة الطول.ملفات MS-DOS مثلا لها أسماء مؤلفة من 1 حتى 8 محارف مع امتداد اختياري من 1 حتى 3 محارف. وفي الإصدار 7 من UNIX كان طول أسماء الملفات حتى 14 محرف بما فيها الامتدادات. إلا أن جميع أنظمة التشغيل الحديثة تقريبا تدعم أسماء الملفات أطول ذات طول متغير. كيف يمكن تحقيق ذلك؟

إن الطريقة الأبسط هي وضع حد أعلى لطول اسم الملف وعادة ما يكون 255 محرفا ثم استخدام احد التصاميم الموضحة في الشكل السابق مع حجز 255 محرفا لكل اسم ملف. تتميز هذه الطريقة ببساطتها لكنها تهدر كمية كبيرة من مساحة الفهرس لأن القليل من الملفات لها أسماء طويلة إلى هذا الحد. من اجل كفاءة التخزين يفضل استخدام طريقة أخرى.

احد الأساليب الممكنة هو التخلي عن فكرة أن جميع سجلات الفهرس متساوية الطول. بهذه الطريقة يحوي كل سجل فهرس قسما ثابتا يبدأ عادة بطول السجل ويتبع ببيانات لها تنسيق ثابت وتتضمن عادة مالك الملف وزمن الإنشاء و معلومات الحماية وغيرها من السمات. بعد هذه الترويسة الثابتة الحجم يأتي اسم الملف الفعلي مهما كان طوله كما في الشكل التالي(a) بالتنسيق المنتهي بالأكبر (big-endian) (مثل نظام sparc).لدينا في هذا المثال ثلاثة ملفاتfoo و personnel وproject budget. ينتهي اسم كل ملف بمحرف خاص (عادة الصفر) والممثل في الشكل بمربع فيه إشارة x للسماح لكل سجل بالبدء اعتبارا من حدود كلمة، يُكمل اسم الملف ليصبح حجمه عددا صحيحا من الكلمات، وهذا موضح بالشكل بواسطة المربعات المظللة.



الشكل 54

من مساوئ هذه الطريقة انه عند إزالة ملف تنتج فجوة متغيرة الطول في الفهرس، قد لا تتسع للملف الذي سيكتب تالياً. تشبه هذه المشكلة تلك التي رأيناها في حالة ملفات القرص المتجاورة إلا أن ضغط الفهرس الآن ممكن لأنه موجود بأكمله في الذاكرة. مشكلة أخرى تتمثل بان سجل فهرس واحد قد يمتد لعدة صفحات مما قد يؤدي إلى حدوث أخطاء صفحات إثناء قراءة اسم الملف.

هناك طريقة أخرى لمعالجة أسماء الملفات متغيرة الطول وتتمثل بجعل سجلات الفهرس نفسها ثابتة الطول وتخزين أسماء الملفات مع بعضها ضمن كومة في نهاية الفهرس كما في الشكل السابق(b). تتميز هذه الطريقة بأنها عند حذف ملف فإن الملف التالي الذي يدخل سيتسع دائما في المكان الفارغ. طبعا يجب إدارة الكومة، وأخطاء الصفحات مازالت محتملة الحدوث أثناء معالجة أسماء الملفات.يوجد مكسب بسيط هنا وهو أن أسماء الملفات لا حاجة لان تبدأ عند حدود الكلمات لذلك لا توجد أي محارف تعبئة في الشكل بينما كانت موجودة في الشكل السابق(a).

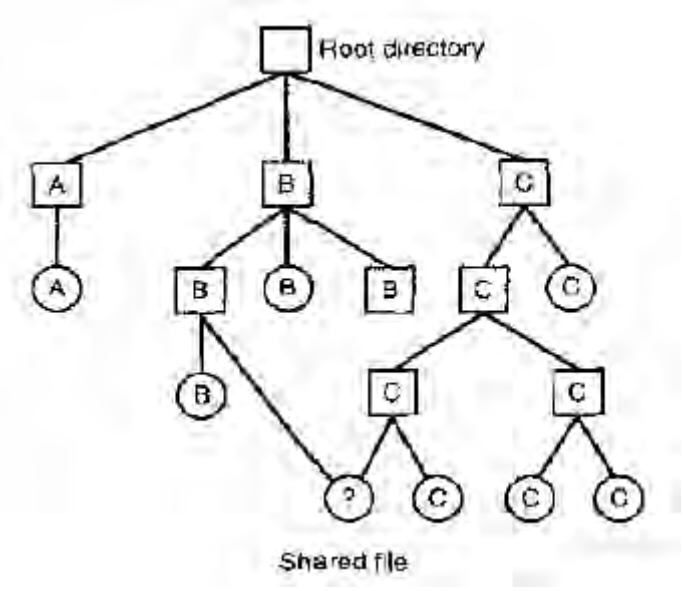
في جميع تصاميم الفهارس التي رأيناها حتى الآن يتم التحدث بشكل خطي من بداية الفهرس وحتى نهايته لإيجاد اسم الملف. من اجل الفهارس الطويلة جدا يصبح البحث الخطي بطيئا. يمكن تسريع البحث باستخدام جدول مجزئ في كل فهرس.ليكن حجم الجدول n. لإدخال اسم ملف، يحول اسم الملف إلى قيمة بين n-1 و0 وذلك مثلا بالتقسيم على n أو أي طريقة أخرى مشابهة.بكلا الطريقتين يتم فحص سجل الجدول الموافق لقيمة التجزئة المحسوبة. إذا لم يكن السجل مستخدما يوضع فيه مؤشر إلى سجل الملف في الفهرس. توضع سجلات الملفات بعد جدول التجزئة. إذا كان السجل مستخدما، يتم إنشاء لائحة مترابطة، ويوضع رأسها في سجل الجدول المحدد وتضم جميع السجلات التي لها نفس قيمة التجزئة من اسم الملف لاختيار سجل جدول التجزئة المناسب..يتم فحص جميع السجلات في السلسلة المخزن رأسها في هذا السجل

لرؤية إذا ما كان اسم الملف موجودا.إذا لم يكن اسم الملف موجودا في السلسلة فانه غير موجود في الفهرس.يتميز استخدام جدول التجزئة بسرعة البحث، لكنه يعاني من مشكلة صعوبة الادارة. ويعتبر مرشحا حقيقيا فقط في الأنظمة التي يتوقع فيها أن تحتوي الفهارس عادة على مئات أو آلاف الملفات.

هناك طريقة مختلفة كليا لتسريع البحث في الفهارس الكبيرة، وتتمثل بتخزين نتائج في ذاكرة مخبئية، فإذا كان كذلك يمكن إيجاد الملف بسرعة وتجنب البحث الطويل في الفهرس. طبعا لا تكون هذه التقنية فعالة إلا إذا كان عدد صغير نسبيا من الملفات تؤلف معظم عمليات البحث.

6 2 8 -الملفات المشتركة

عندما يعمل عدة مستخدمين مع بعضهم على مشروع واحد، فإنهم يحتاجون غالبا إلى التشارك بالملفات. نتيجة لذلك من الأنسب غالبا أن يظهر نفس الملف في عدة فهارس تابعا لعدة مالكين بنفس الوقت.يبين الشكل التالي نفس الملفات في الشكل السابق بتسع مرات مرة أخرى، لكن بوجود أحد ملفات المستخدم C في احد فهارس المستخدم B. يسمى الاتصال بين فهرس B والملف المشترك بالوصلة (link). يعتبر نظام الملفات الآن غرافا موجها غير حلقي (Directed Acyclic Graph)DAG وليس شجرة.



الشكل55

مشاركة الملفات جيدة لكنها تولد بعض المشاكل أيضاً. أولاً.إذا كانت الفهارس في الواقع تحوي عناوين قرص يجب وضع نسخة عن عناوين القرص ضمن فهرس B عند ربط الملف. إذا قام B أو C بالإضافة إلى الملف لاحقا، فان الكتل الجديدة ستسجل فقط في سجل المستخدم الذي قام بالإضافة. ولن تكون هذه التغييرات مرئية للمستخدم الآخر وهذا يتناقض مع الغرض من المشاركة.

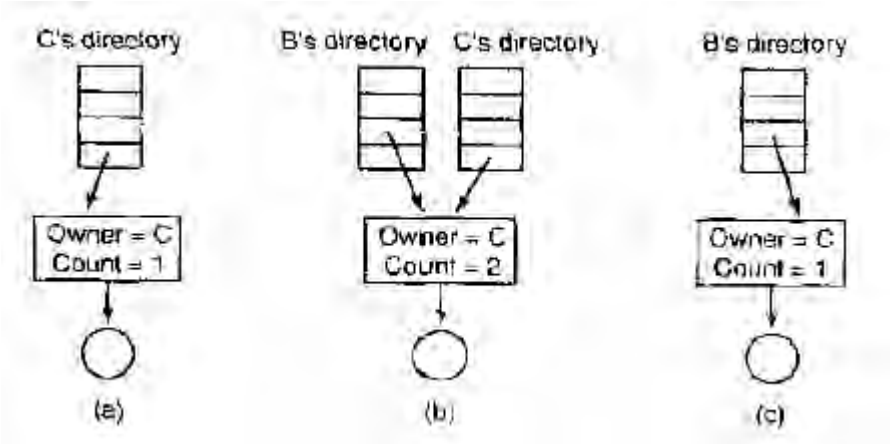
يمكن حل هذه المشكلة بطريقتين: يعتمد الحل الأول على عدم سرد كتل القرص في الفهارس، بل ضمن بنية بيانات صغيرة مقترنة بالملف نفسه. تشير الفهارس عند ذلك إلى بنى

البيانات الصغيرة فحسب.هذه الطريقة مستخدمة في UNIX (حيث تسمى بنية البيانات الصغيرة هذه i-node).

في الحل الثاني، يقوم B بالارتباط بأحد ملفات C من خلال إنشاء ملف جديد من النوع link وإدخال هذا الملف في فهرس B. يحوي الملف الجديد اسم المسار للملف المرتبط معه فقط. عندما يقرا B من الملف المرتبط، يرى نظام التشغيل أن الملف الذي يقرا منه من النوع link وعندها يقوم بالبحث عن اسم الملف ويقرا ذلك الملف..تسمى هذه الطريقة بالربط الرمزي (symbolic linking).

لكل من هذه الطرق مساوئها الخاصة. في الطريقة الأولى عندما يتصل B بالملف المشترك، يكون المالك المسجل في i-node هو C. إن إنشاء ارتباط لا يغير الملكية كما في الشكل التالي، لكنه يزيد عداد الارتباط في i-node، بذلك يستطيع النظام معرفة عدد سجلات الفهرس التي تشير إلى الملف حالياً.إذا حاول C لاحقا حذف الملف سيواجه النظام مشكلة. إذا حذف الملف وأزال عقدةi-node، فان B سيحصل في فهرسه على سجل يشير إلى عقدة i-node غير صالحة. إذا تمت إعادة إسناد هذه العقدة لاحقا إلى ملف آخر فإن ارتباط B سيشير إلى الملف الخطأ. يستطيع النظام معرفة أن الملف مازال مستخدما من العداد الموجود في عقدة i-node لكنه لا يستطيع إيجاد جميع سجلات الفهرس التي تشير إلى الملف من اجل مسحها. لا يمكن تخزين مؤشرات إلى الفهارس في عقدة i-node لأن عددها غير محدود.

الشيء الوحيد الممكن فعله هو إزالة سجل الفهرس في فهرس المستخدم C وترك عقدة i-node موجودة مع جعل العداد يساوي 1 كما في الشكل التالي(c). لدينا الآن حالة فيها المستخدم B هو الوحيد الذي لديه سجل في فهرسه يشير إلى الملف الذي يملكه C. إذا كان النظام يحوي نظام محاسبة أو حصص فإن C سيستمر في تحمل أعباء هذا الملف حتى يقررb إزالته وعندها يصبح العداد يساوي 0 ويتم حذف الملف.



الشكل56

لا تظهر هذه المشكلة مع الارتباطات الرمزية لأن المالك الحقيقي فقط لديه مؤشر إلى عقدة i-node. أما المستخدمون المرتبطون بالملف فلديهم أسماء المسارات فقط. وليس مؤشرات إلىi-node. عندما يزيل المالك الملف يتم تدميره. وستفشل أي محاولة لاحقة لاستخدام الملف عبر الارتباطات الرمزية لأن النظام عند ذلك لن يستطيع إيجاد الملف في مكانه المحدد. أما إزالة الارتباط الرمزي فإنها لا تؤثر على الملف إطلاقا.

المشكلة في الارتباطات الرمزية العبء الإضافي الذي تحتاجه..يجب قراءة الملف الذي يحوي المسار ثم يجب تفسير المسار وتتبعه عنصرا بعد آخر حتى وصول إلى عقدةi-node. قد تتطلب كل هذه العمليات عددا لا باس به من عمليات الوصول إلى القرص.بالإضافة إلى ذلك توجد عقدة i-node إضافية من اجل كل ملف ارتباط رمزي وكذلك كتلة قرص إضافية لتخزين المسار مع أنه إذا كان اسم المسار قصيرا يمكن تخزينه في العقدة i-node نفسها. تتميز الارتباطات الرمزية بإمكانية استخدامها للارتباط بملفات موجودة في أي جهاز في العالم وذلك ببساطة بتزويد عنوان الشبكة للجهاز الذي يحوي الملف بالإضافة إلى مساره في هذا الجهاز.

هناك أيضاً مشكلة أخرى للارتباطات، أكانت رمزية أم غير رمزية. عند السماح بوجود الارتباطات يمكن أن تملك الملفات مسارين أو أكثر. البرامج التي تبدأ في فهرس معين و تقرأ جميع الملفات في هذا الفهرس والفهارس المتفرعة عنه ستجد الملفات المرتبطة عدة مرات. مثلا البرنامج الذي يخزن جميع الملفات الموجودة في فهرس مع فهارسه الفرعية على شريط، قد يقوم بنسخ ملف مرتبط عدة مرات.بالإضافة إلى ذلك إذا قرأ الشريط في جهاز آخر فان الملف سينسخ مرتين على القرص عوضا عن إنشاء ارتباطات إلا إذا كان برنامج النسخ ذكيا.

6 3 -أنظمة الملفات EXT &FAT

تم اعتماد كل من نظامي الملفات FAT12 ونظام الملفات Ext2. حيث يتم استخدام نظام الملفات FAT12 عند التعامل مع القرص المرن وهذا ضروري جداً من أجل عمليات القراءة من القرص المرن حيث أن جميع أنظمة الملفات تعتمد نظام الملفات FAT12 للتعامل مع القرص المرن.

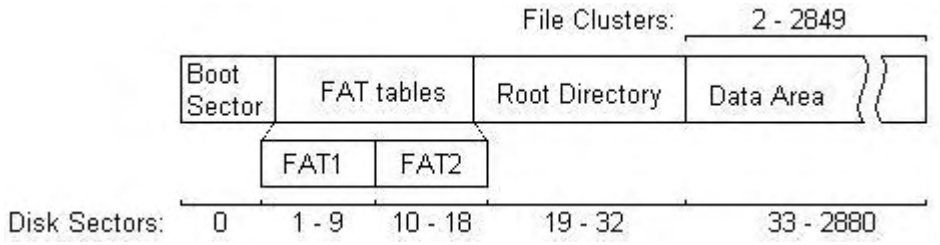
أما من أجل التعامل مع القرص الصلب فإن نظام الملفات FAT12 غير ملائم على الإطلاق كما سنرى لاحقاً، لذلك تم اعتماد نظام الملفات Ext2 المعتمد من قبل أنظمة التشغيل Linux, Unix. وذلك لما له من ميزات كثيرة سنراها عند شرحه.

6 4 -نظام الملفات FAT12

6 4 1 -أقسام نظام الملفات FAT12

يتألف نظام الملفات FAT12 من أربع أقسام رئيسية و هي:

قطاع الإقلاع Boot sector
جداول FAT FAT Tables
الفهرس الجذر Root Directory
منطقة المعطيات Data Area



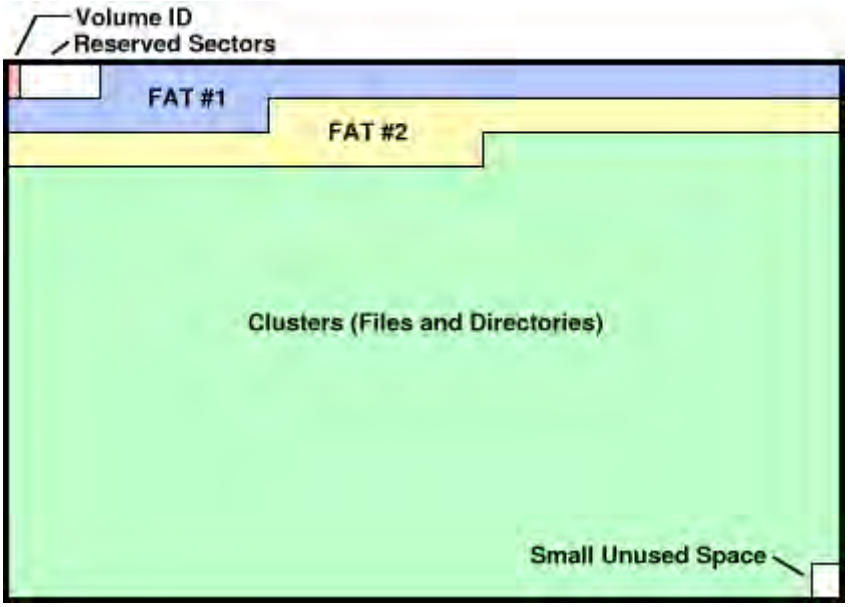
الشكل57

قطاع الإقلاع Boot sector: و هو القطاع الأول (Sector 0) على القرص، و يحتوي معلومات محددة عن كامل منظومة القرص الصلب و نظام الملفات، بما في ذلك كم يحتوي القرص من نسخ جداول FAT، و كم هي سعة القطاع، و كم قطاع موجود في العنقود Clusterالواحد و غيره.

جداول FAT Tables FAT: و هي مؤشرات لكل عنقود على القرص، و تحدد كل من رقم العنقود التالي في سلسلة العناقيد الحالية، و نهاية سلسلة العناقيد، و إذا ما كان العنقود فارغاً أو فيه أخطاء. و جداول FAT هي الطريقة الوحيدة لإيجاد مواقع الملفات و الفهارس على القرص، و لذلك إذا تم تخريب معلومات جدول FAT فإنه سيتم فقدان المعلومات من القرص. و هناك نسختان من جداول FAT على القرص من أجل ضمان أمن المعلومات و إمكانية الاستعادة.

الفهرس الجذر Root Directory: و هو الفهرس الرئيسي على القرص. و هو ليس كباقي الفهارس المتواجدة في منطقة المعطيات Data Area، و للفهرس الجذر حجم محدود، والفهرس الجذر حجم محدود14 قطاع X 16 مدخل فهرس في كل قطاع = 224 مدخل ممكن، (و سيتم شرح التفاصيل لاحقاً) و بذلك فإن عدد الملفات أو الفهارس التي يمكن إنشاؤها على فهرس لجذر محدود.

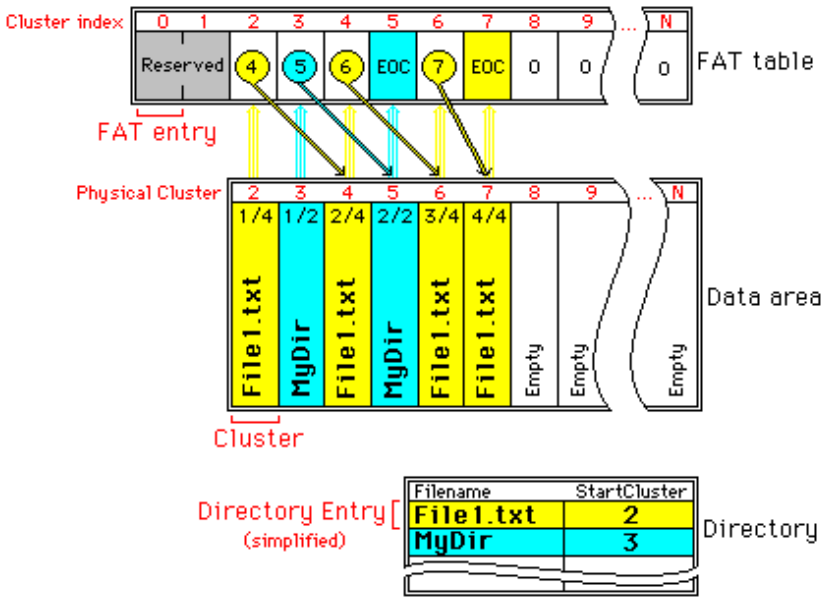
منطقة المعطيات Data Area: أول قطاع من منطقة المعطيات هو القطاع 33 وفقاً للعنقود 2 من نظام الملفات (العنقود الأول هو دائماً العنقود 2). و تحتوي منطقة المعطيات على معطيات الملفات و الفهارس.



الشكل **58**

2 4 6 -بنية نظام الملفات FAT12

جداول FAT12 منظمة كمصفوفة خطية من مداخل FAT. مدخل FAT عبارة عن قيمة من 12 خانة و التي توافق قيمتها عنقود متوضع في منطقة المعطيات على القرص. و بما أن أول عنقود في منطقة المعطيات على القرص توافق العنقود 2، فإن أول مدخلي FAT في جدول FAT محجوزين و لا يستخدمان. و يمكن تصور جدول FAT على أنه مصفوفة كبيرة تحتوي مؤشرات على منطقة المعطيات في القرص. و هي أيضاً عبارة عن لائحة مترابطة لأنه بالإمكان الاستحصال على الملفات و التي حجمها أكبر من عنقود واحد عبر مؤشر في جدول FAT و التي يشير إلى العنقود التالي من الملف أو الفهرس.



الشكل **59**

و يمكن أن يأخذ مدخل FAT و المكون من 12 خانة أحدى القيم التالية:

000:عنقود غير مستخدم
FF0-FF6: عنقود محجوز (مثل المدخلين 0،1)
FF7: عنقود معطوب
FF8-FFF: نهاية ملف أو فهرس و يرمز له EOC (End Of Cluster chain)
أرقام أخرى: و هي عبارة عن مؤشرات إلى العنقود التالي للملف أو الفهرس.

سلاسل العناقيد عبارة عن سلاسل من مداخل FAT تشير إلى العنقود التالي من الملف أو الفهرس، و تنتهي بقيمة EOC تماماً مثل اللوائح المترابطة. ممثلاً في الصورة السابقة سلسلة العناقيد للملف File1.txt يمكن اشتقاقها بالبدء من العنقود الأول و المحدد في الفهرس الذي يحتوي الملف (الصندوق السفلي في الصورة السابقة) و بملاحقة المؤشرات في جدول FAT حتى الوصول إلى القيمة EOC. السلسلة ستعطي القيم 2,4,6,7. سلسلة عناقيد الفهرس MyDir هي 3,5.

عند تعديل أو تحديث جدول FAT يجب التأكد من تعديل أو تحديث نسختي جدول FAT. و تستخدم النسخة الثانية للتأكد من عدم وجود أخطاء. فمثلاً برنامج fsck.msdos في Windows يتأكد من صلاحية المعطيات على القرص عبر مقارنة جدولي FAT للتأكد من أنهما متطابقين. ويوجد 3072 مدخل FAT في كل جدول FAT. (512 بايت في كل قطاع X 9 قطاعات = 4608 بايت / 1.5 بايت لكل مدخل FAT = 3072 مدخل FAT)

6 4 3 -تسمية الملفات و الامتدادات في FAT12

الأسماء في نظام الملفات FAT12 محدودة بـ 8 محارف للاسم و 3 للامتداد. و هناك بعض المفاهيم لابد من مراعاتها:

أسماء الملفات/الفهارس و امتداداتها غير منتهية ضمن مدخل الفهرس
أسماء الملفات/الفهارس يحتل دائماً 8 بايتات حتى ولو كان طول الاسم أقصر من 8 بايتات، و نفس الشيء بالنسبة للامتداد ذو الثلاث محارف
أسماء الملفات/الفهارس و امتداداتها دائماً محارف كبيرة و يتم دائماً تحويل الأسماء إلى محارف كبيرة في نظام DOS
أسماء الفهارس يمكن أن تحتوي على امتدادات أيضاً
الملف FILE1 و الملف FILE!.TXT مختلفان في الاسم (الامتداد مهم)
لا يمكن أن يكون للملفات و الفهارس نفس الاسم حتى ولو كانت سماتهما مختلفة.

و فيما يلي بعض الأمثلة توضح كيف يتم التحويل إذا 11 بايت محجوز لاسم الملف وامتداده في مدخل الفهرس(الفراغ ضمن الفاصلتين العلويتن يعتبر محرف فراغً).

```
filename provided [01234567012]
"foo.bar"      -> "FOO     BAR"
"FOO.BAR"      -> "FOO     BAR"
"Foo.Bar"      -> "FOO     BAR"
"foo"          -> "FOO       "
"foo."         -> "FOO       "
"PICKLE.A"     -> "PICKLE  A  "
"prettybg.big" -> "PRETTYBGBIG"
".big"         -> illegal! file/directory names cannot begin with a "."
```

4 4 6 -بنية الفهارس و حقولها

يتضمن مدخل الفهرس كل المعلومات الضرورية ليشير إلى محتوياته من الملفات و الفهارس على القرص، بما في ذلك معلومات إضافية حول سمات الفهرس أو الملف. وتتضمن هذه المعلومات: وقت إنشاء/تعديل الملف، تاريخ الإنشاء، حجم الملف بالبايتات. كل هذه المعلومات تكون مجموعة ضمن بنية صغيرة من 32 بايت. الفهارس التي تحتوي أكثر من 16 مدخل تحتاج إلى أكثر من قطاع واحد(عدد المداخل الأعظمي لكل قطاع =512 بايت في كل قطاع / 32 بايت لكل مدخل فهرس = 16 مدخل فهرس). يمكن أن يكون يأخذ الفهرس في منطقة المعطيات أي حجم، و لكن بما أن حجم الفهرس الجذر ثابت فسيكون هناك عدد محدد من مداخل الفهارس.

تتضمن الفهارس الفرعية (الفهارس ضمن الفهرس الجذر) مداخل لفهرسين فرعيين ضمنها تلقائياً. مداخل الفهارس هذه هي نقطة [.] و نقطتين متتاليتين [..]، هذه المداخل عبارة عن مؤشرات،الأول [.] يشير إلى الفهرس الحالي، المدخل الثاني [..] يشير إلى الفهرس الأب. وهو الفهرس الذي يتم تحديده عند كتابة التعليمة CD.. ويعتبر الفهرس الذي يحتوي فقط هذين المدخلين فارغاً.

يتألف مدخل الفهرس من الحقول التالية:

```
unsigned char  Name[8]; /* File name in capital letters. Padded with
    spaces, not NULL terminated. */
unsigned char  Extension[3]; /* Extension in capital letters. Also
    padded with spaces. There is no '.' separator. The '.' is added for
    readability when filenames with extensions are displayed */
unsigned char  Attributes;/* Holds the attributes code */
unsigned char  Reserved[10];/* Reserved for Windows NT. Set to zero
    when creating files/directories. */
unsigned short Time; /* Time of last write (file/dir creation is a write)
    */
unsigned short Date; /* Date of last write */
unsigned short startCluster; /* Pointer to the first cluster of the file. */
unsigned long  fileSize; /* File size in bytes. Make sure this is updated
    when the file is modified */
```

و تحتوي سمات الملف أو الفهرس على القيم التالية:

الشرح	البت المقابل	السمة	القيمة الست عشرية
لا يسمح بالكتابة (للقراءة فقط)	0 (LSB)	Read Only	1
لا يتم عرض الملف	1	Hidden	2
الملف خاص بنظام التشغيل	2	System	4
اسم الملف عبارة عن اسم لسواقة	3	Volume ID	8
عند تحديده يشير إلى أنه فهرس و إلا فهو ملف	4	Directory	10
تم تعديل الملف	5	Archive	20

السمة directory و السمة archive لا يمكن أن تتحددان في نفس الوقت، فالسمة directory هي الطريقة الوحيدة للتفريق بين الملف و الفهرس. من أجل الفهارس الفرعية التي تشير إلى الفهرس الجذر فإن قيمة عنقود البدء تكون 0، و لكن كما ذكرنا سابقاً فإن القيمة 0 محجوزة ضمن جدول FAT. لذلك يجب الانتباه إلى أن القيمة صفر لا تعني المدخل رقم 0 من جدول FAT و لكنها تشير إلى الفهرس الجذر.

6 4 5 -حذف الملفات و الفهارس في FAT12

لا يتم محي كامل الفهرس أو الملف مباشرة عند القيام بالحذف. و لكن في الواقع فإن عملية الحذف لا يعدل أي شيء في العناقيد التي تحتوي المعلومات. و لكن يتم تصفير سلسلة العناقيد للملف/الفهرس من جدول FAT، ويتم وضع قيمة خاصة هي e5 في أول بايت من مدخل الفهرس، للإشارة إلى أن هذا المدخل تم حذفه. وبذلك من الممكن أن يتم إعادة الكتابة فوق مدخل الفهرس و عندها يتم حذف الفهرس بشكل فعلي من القرص.

6 4 6 -تنفيذ نظام الملفات FAT12

فيما يلي مجموعة التوابع المستخدمة ضمن نظام التشغيل، بالإضافة إلى شرح مبسط عنها. و هذا التوابع هي التي تؤمن التعامل نظام التشغيل مع نظام الملفات FAT12 (بإمكانك رؤية هذه التوابع ضمن الملف fat12.c وترويسته fat.h).

6 4 6 1- بنى المعطيات

أهم بنى المعطيات المستخدمة.

```
typedef struct fat12_super_block
{
	byte Jump[ 3 ];
	unsigned char Name[ 8 ];
	word BytesPerSector;
	byte SectorsPerCluster;
	word ReservedSectors;
	byte Fats;
	word RootDirectoryEntries;
	word LogicalSectors;
	byte MediumDescriptorByte;
	word SectorsPerFat;
	word SectorsPerTrack;
	word Heads;
	word HiddenSectors;
	byte code[ 482 ];
}
```

ويحتوي هذا السجل معلومات قطاع الإقلاع

```
typedef struct FAT12
{
	byte data[ FAT_SECTOR_SIZE * FAT_PHYS_SIZE ];
}
```

يحتوي معلومات جدول FAT12 حيث FAT_SECTOR_SIZE = 512 و FAT_PHYS_SIZE = 9

```
typedef struct logical_FAT12
{
	word data[ 3072 ];
}
```

وهو عدد مداخل FAT12 ضمن جدول FAT حيث عند قراءة جدول FAT يتم تحويله بحيث توضع قيمة كل مدخل في حقل من المصفوفة حيث يتم التحويل من 12 خانة للمدخل إلى 16 خانة لتسهيل التعامل ويتم إعادة كتابة القيمة وفي جدول FAT بعد تحويلها إلى 12 خانة عند إلغاء تحميل القرص المرن umount.

```
typedef struct fat12_file_attr
{
	unsigned long read_only: 1;
	unsigned long hidden	: 1;
	unsigned long system	: 1;
	unsigned long label : 1;
	unsigned long directory: 1;
	unsigned long archive	: 1;
	unsigned long __res: 2;
}
```

سمات الملف كما هي معرفة في نظام الملفات FAT12

```
typedef struct FileEntry
{
        unsigned char Name[8];
        unsigned char Extension[3];
        fat12_file_attr_t Attribute;
        byte Reserved[10];
        word Time;
        word Date;
        word StartCluster;
        dword FileLength;
}
```

معلومات مدخل الملف/الفهرس كما تتواجد في الفهرس الحاوي للملف. حيث كل مدخل ملف يتكون من 32 خانة

```
typedef struct SectorDir
{
        FileEntry_t Entry[ FAT_SECTOR_SIZE / sizeof(FileEntry_t) ];
}
```

عبارة عن سجل يحتوي مصفوفة تتضمن مداخل الملفات ضمن قطاع ما.

2 6 4 6 -التوابع

فيما يلي شرح عن عمل بعض التوابع الهامة التي تم استخدامها في أجل تنفيذ التعامل مع نظام الملفات FAT12 ضمن نظام التشغيل:

int fat12_check(void)

يتأكد من صلاحية جدول FAT12 الموجود على القرص المرن.

void fat12_invalidate_mount()

إلغاء تحميل جداول FAT وذلك بمسح المعلومات التي تم تحميلها عن قطاع الإقلاع و تحرير fat12_super_block.

bool next_sector(word *next, word actual)

للحصول على القطاع التالي الذي يحتوي تتمة بيانات الملف/الفهرس، حيث actual هو القطاع الحالي و next سيكون فيه النتيجة ويشير للقطاع التالي

int how_many_cluster(word start)

بإعطائه رقم قطاع بداية الملف يقوم بحساب مجموع القطاعات التي يمتلكها الملف، وبالتالي يكون حجم الملف هو ناتج هذا التابع * 512 بايت

int fat12_read_clusters(char *buf, size_t size, uint16_t cstart, uint16_t cnum)

قراءة بعض العناقيد من الملف ويعيد عدد العناقيد التي تم قراءتها.

int fat12_write_clusters(char *buf, size_t size, uint16_t cstart, uint16_t cnum)

كتابة بعض العناقيد من الملف ويعيد عدد العناقيد التي تم كتابتها.

int fat12_get_file_entry(FileEntry_t *f, int dir, char *filename)

بتمرير مدخل الفهرس ورقم بداية قطاعه، الذي يحتوي الملف المراد الحصول على مدخله الفهرس و اسم ذلك الملف فيعيد هذا التابع رقم قطاع بداية هذا الملف.

int fat12_set_path(int dir, const char *buf)

يجعل مسار الملف buf هو الفهرس الحالي. ويعيد رقم عنقود بداية الفهرس أو رسالة خطأ.

int __fat12_locate_file(FileEntry_t *fe, int dir, char *path)

تحديد موقع الملف وذلك بإعادة رقم عنقود البداية اعتماداً على المسار path

int __fat12_ls(int dir, char *path)

طباعة سمات وأسماء الملفات الموجودة ضمن مسار معين، و يعيد عدد المداخل التي تم طباعتها.

int fat12_write()

كتابة مصفوفة السجل logical_FAT12 ضمن جدول FAT للقرص المرن. وذلك بعد تحويل كل حقل فيه من 16 خانة إلى 12 خانة ليتوافق مع بنية جدول FAT12.

int __fat12_rm(int dir, char *path)

حذف ملف من القرص المرن.

int __fat12_sync(void)

كتابة جدول FAT في حال تم تغييره.

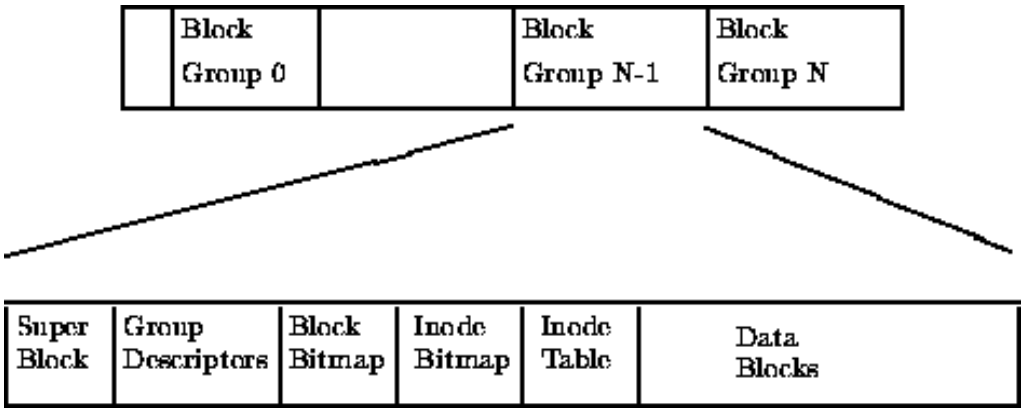
int fat12_mount(void)

إجراء عملية تحميل جدول FAT وتهيئة المتحولات المتعلقة به

int fat12_umount(void)

إلغاء تحميل جدول FAT

6 5 -نظام الملفات Ext2
نظام الملفات الممتد الثاني (The Second
(Extended File System



الشكل60

تم تطوير نظام الملفات Ext2 من قبل Rémy Card نظام ملفات فعال و قوي ليعمل على نظام Linux. وهو أكثر نظام ملفات ناجح حتى الآن في عالم Linux و أساس كل نظم ملفات توزيعات Linux المختلفة.

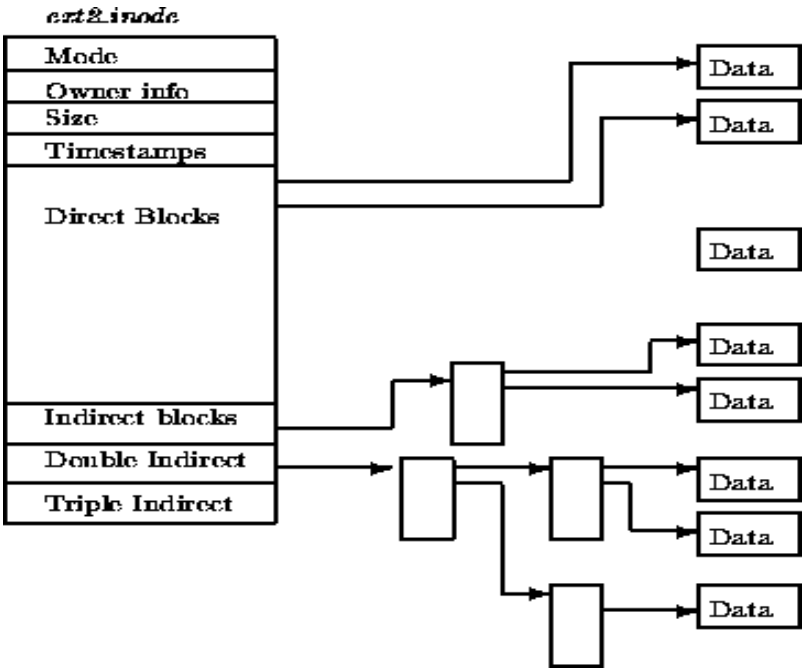
نظام الملفات Ext2 مبني كمعظم أنظمة الملفات الأخرى على أساس حفظ المعطيات في ملفات يتم حفظها على القرص ككتل من المعطيات. كتل المعطيات جميع كتل المعطيات تمتلك نفس الطول بحيث يمكن أن تختلف أطوال كتل المعطيات من نظام ملفات Ext2 لآخر حيث يتم تحديده عند إنشاء نظام الملفات مع نظام التشغيل. و يتم تدوير حجم الملف إلى القيمة الصحيحة الأعلى من الكتل. فإذا كان حجم الكتلة 1024 بايت، فسيحتل الملف ذو الحجم 1025 بايت كتلتين بحجم 1024 بايت. لسوء الحظ هذا يعني أنه بالمتوسط سيتم هدر نصف كتلة لكل ملف.

لا تحتوي جميع الكتل في نظام الملفات على معطيات، فبعضها يستخدم لحفظ المعلومات التي تصف بنية نظام الملفات. ويعرف Ext2 طبولوجية نظام الملفات بوصف كل ملف من ملفات النظام بنية معطيات تدعى عقدة ملف inode. تصف عقدة الملف أي من الكتل تشغلها معطيات الملف بالإضافة إلى حقوق الوصول إلى الملف ووقت تعديل الملف و نوع الملف. يتم وصف كل ملف في نظام الملفات Ext2 عن طريق عقدة ملف واحدة و لكل عقدة ملف رقم وحيد معرف لها. يتم حفظ عقد ملفات النظام ضمن جداول العقد inode tables. فهارس Ext2 هي ببساطة عبارة عن ملفات خاصة يتم وصفها بعقد ملفات و تحتوي على مؤشر إلى عقد الملفات ضمنها.

الصورة السابقة تظهر خرج نظام الملفات Ext2 وهو يحتل سلسلة من الكتل. يقسم نظام الملفات Ext2 الجزء المنطقي الذي يحتلها إلى مجموعات كتلة Block Groups. كل مجموعة تكرر المعطيات الهامة من أجل ضمان تكاملية نظام الملفات و لحفظ الملفات و الفهارس الحقيقية ككتل من المعلومات و المعطيات. التكرار ضروري لأنه من الممكن أن تحدث مشاكل و بالتالي ليتمكن نظام الملفات من إجراء عملية استعادة.

1 5 6 -عقدة ملفات نظام الملفات Ext2

The EXT2 Inode



الشكل 61

عقدة الملف هي وحدة البناء الأساسية في نظام الملفات Ext2. كل ملف وفهرس في نظام الملفات يتم وصفه بعقد ملف واحدة ووحيدة. يتم حفظ عقد الملفات لكل مجموعة كتلة في جدول العقد مع الخريطة Bitmap و التي تسمح للنظام بمتابعة عقد الملفات الموجودة. و الصورة السابقة توضح هيئة عقدة الملفات في نظام الملفات Ext2. ومن أهم معطيات عقدة الملفات:

mode: و فيه يتم حفظ معلومتين، ماذا تصف هذه العقدة وما هي السماحيات التي يملكها المستخدمون. و في نظام الملفات Ext2 يمكن أن تصف العقدة إما ملفاً أو فهرساً أو رابط رمزي symbolic link أو جهاز دخل أو FIFO.

Owner Information: الأرقام المعرفة للمستخدمين والمجموعات لهذا الملف أو الفهرس. هذا يتيح لنظام الملفات بالسماح فقط للمستخدمين الصحيحين الوصول للملف أو الفهرس.

Size: حجم الملف بالبايتات.

Timestamps: الوقت الذي تم فيه إنشاء العقدة ووقت آخر تعديل تم إجراءه على العقدة.

Datablocks: مؤشر إلى الكتل التي تحتوي المعطيات للملف الذي تمثله هذه العقدة. أول 12 مؤشر تشير إلى الكتل الفيزيائية التي تحتوي على المعطيات الممثلة بهذه العقدة، وآخر 3 مؤشرات تحتوي مستويات أكثر من المعطيات في كتل غير مباشرة. فعلى سبيل المثال المؤشرين غير المباشرين يشيران إلى كتلة مؤشرات إلى كتل معطيات. وهذا يعني أن الملف الذي حجمه أقل من 12 كتلة معطيات يمكن الوصول إليه بشكل أسرع من الملفات الكبيرة الحجم.

6 5 2 -الكتلة الكبرى في نظام الملفات Ext2 The EXT2 Superblock

تحتوي الكتلة الكبرى Superblock وصفاً للحجم و الشكل الأساسي لنظام الملفات الحالي. وتتيح المعلومات الموجودة ضمنها لمدير نظام الملفات استخدام و صيانة نظام الملفات. عادة تقرأ فقط الكتلة الكبرى من مجموعة الكتلة رقم0 و ذلك عندما يتم تحميل mount نظام الملفات، ولكن كل مجموعة كتلة تحتوي نسخة مكررة من أجل ضمان عدم ضياع المعلومات. ومن أهم المعلومات التي توجد في الكتلة الكبرى:

Magic Number: وهذا الرقم يتيح للبرنامج الذي يقوم بتحميل نظام الملفات من التأكد بأن هذه الكتلة الكبرى موجودة في نظام الملفات Ext2. حيث يكون الرقم من أجل نسخة Ext2 هو EF53.

Revision Level: تتيح المستويات الرئيسية و الصغرى للنسخة revision الكود الذي يقوم بالتحميل أن يحدد إذا ما كان نظام الملفات يدعم مواصفات متاحة فقط في نسخ معينة من نظام الملفات.

Mount Count and Maximum Mount Count: ويتيحان معاً للنظام تحديد إذا ما كان نظام الملفات يجب التأكد من عدم وجود أخطاء فيه. فعداد التحميل Mount Count يزداد كل مرة يتم تحميل نظام الملفات ولا يجوز أن تتجاوز قيمته القيمةMaximum Mount Count.

Block Group Number: رقم مجموعة الكتلة التي تحوي الكتلة الكبرى الحالية.

Block Size: حجم الكتلة بالبايتات.

Blocks per Group: عدد الكتل في المجموعة. وهو مثل سابقه قيمة ثابتة يتم تحديدها عند إنشاء نظام الملفات.

Free Blocks: عدد الكتل الفارغة في نظام الملفات.

Free Inodes: عدد عقد الملفات الفارغة في نم.

First Inode: رقم أول عقدة ملف في نم. أول عقدة ملفات في نظام الملفات Ext2 هو مدخل الفهرس '/'.

6 5 3 -واصف المجموعة في Ext2 The EXT2 Group Descriptor

كل مجموعة كتلة تحتوي بنية معطيات تصفها. وكما في الكتلة الكبرى فإن كل واصفات الكتل لكل مجموعات الكتلة تكون مكررة ضمن كل مجموعة كتلة لضمان استعادة المعطيات في حال حدوث خطأ. و يحتوي واصف المجموعة كل مما يلي:

Blocks Bitmap: رقم الكتلة لخريطة توضع الكتلة من أجل مجموعة الكتلة الحالية.

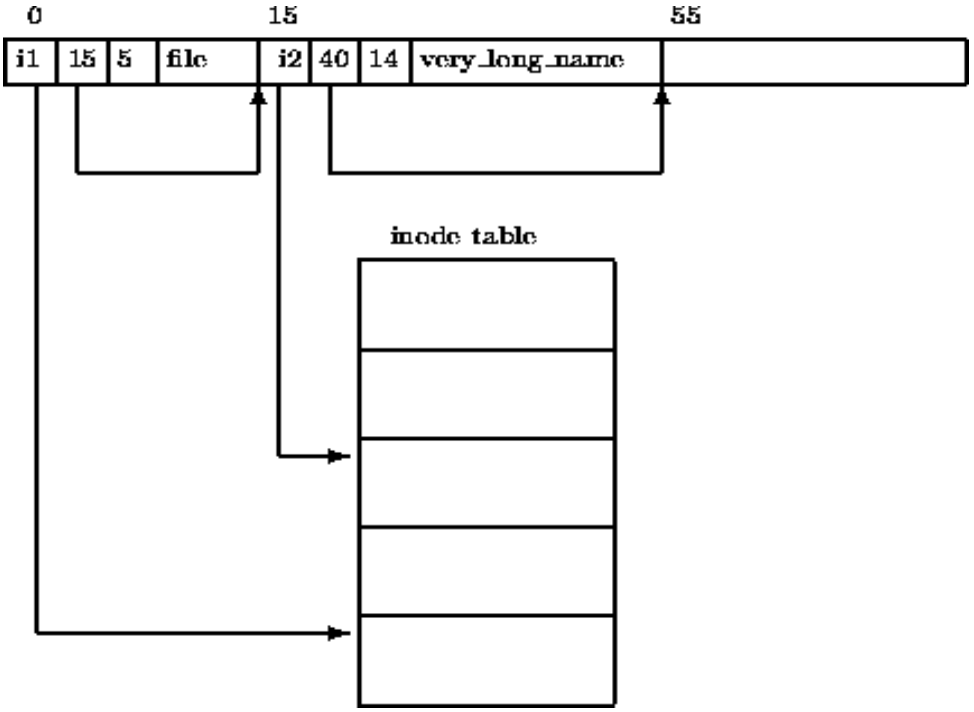
Inode Bitmap: رقم الكتلة لخريطة توضع عقد الملفات من أجل مجموعة الكتلة الحالية.

Inode Table: رقم الكتلة الأولى ضمن جدول العقد من أجل مجموعة الكتلة الحالية.

Free blocks count, Free Inodes count, Used directory count: تتوضع واصفات المجموعات مع بعضها لتشكل جدول واصفات المجموعة. وتحتوي كل مجموعة كتلة كامل جدول واصفات المجموعة بعد نسخة الكتلة الكبرى.

6 5 4 -فهارس نظام الملفات Ext2

EXT2 Directories



الشكل62

فهارس نظام الملفات Ext2 عبارة عن ملفات خاصة والتي يتم استخدامها لإنشاء والحفاظ على مسارات الوصول للملفات ضمن نم. و تظهر الصورة السابقة توضع الفهرس في الذاكرة.

ملف الفهرس عبارة عن لائحة بمداخل الفهارس، و كل منها يحتوي ما يلي:

inode: عقدة الملف لمدخل الفهرس. وهو عبارة عن مؤشر لمصفوفة من عقد الملفات المحفوظة في جدول عقد الملفات في مجموعة الكتلة.
name length: حجم الفهرس بالبايتات.
name: اسم مدخل الفهرس.

أول مدخلي فهرسين ضمن كل فهرس يكونان دائماً '.' و '..' حيث يعنيان 'الفهرس الحالي' و 'الفهرس الأب'.

6 5 5 -تنفيذ نظام الملفات Ext2

فيما يلي مجموعة التوابع المستخدمة ضمن نظام التشغيل، بالإضافة إلى شرح مبسط عنها. و هذا التوابع هي التي تؤمن التعامل مع نظام التشغيل مع نظام الملفات Ext2 (بإمكانك رؤية هذه التوابع ضمن الملف ext2.c وترويسته ext2.h).

1-5-5-6 -بنى المعطيات

```
struct super_block{
 dword s_inodes_count; //total number of inodes used
 dword s_blocks_count; //total number of blocks used
 dword s_r_blocks_count; //total number of blocks reserved to super user
 dword s_free_blocks_count; //total number of free blocks
 dword s_free_inodes_count; //total number of free inodes
 dword s_first_data_block;
 dword s_log_block_size;
 dword s_log_frag_size;
 dword s_blocks_per_group; //total number of blocks per group
 dword s_frags_per_group; //total number of fragments per group
 dword s_inodes_per_group; //total number of inodes per group
 dword s_mtime; // last time for mounting the file system
 dword s_wtime; // last time for writing access time to the file system
 word s_mnt_count; // number of times the fs was mounted
 word s_max_mnt_count; //maximum numbers of allows to mount the fs
 word s_magic; // magic number
 word s_state;
 word s_errors; //in case of errors
 word s_minor_rev_level; //
 dword s_lastcheck; // last check of file system
 dword s_checkinterval; // maximum times between tow checks
 dword s_creator_os; // identifier to the OS creator
 dword s_rev_level; // level of review
 word s_def_resuid; // default user id for reserved blocks
 word s_def_resgid; // default group id for reserved blocks

 dword s_first_ino; // index of the first inode
 word s_inode_size; // inode size if not dynamic = 128
 word s_block_group_nr; // block group number in the superblock
 dword s_feature_compact; //
 dword s_feature_incompact; //
 dword s_feature_ro_compact; //
 byte s_uuid[16]; // id of volume 128 bit
 word s_volume_name; //
 byte s_last_mounted[8]; // path directory where the superblock is mounted
 dword s_algo_bitmap; //
 byte s_reserved[886]; // reserved
};
```

يحتوي هذا السجل بنية الكتلة الكبرى لنظام الملفات Ext2

```
struct group_descriptor{
 dword bg_block_bitmap; // block that contains block bitmap
 dword bg_inode_bitmap; // block that contains inode bitmap
 dword bg_inode_table; // block that contains inode table
 word bg_free_blocks_count; //total number of free blocks
 word bg_free_inodes_count; // total number of free inodes
 word bg_used_dirs_count; // number of inode allocated in dir
 word bg_pad; // used for padding of the structure
 dword bg_reserved[3]; // reserved for future implementation
};
```

يحتوي هذا السجل بنية واصف المجموعة لنظام الملفات Ext2

```
struct i_node{
 word i_mode; // format of files and access rights
 word i_uid; // file creator user id
 dword i_size; // file size in bytes
 dword i_atime; // last access in seconds starting form 1/1/1970
 dword i_ctime; // creation data starting form 1/1/1970
 dword i_mtime; // last modification starting form 1/1/1970
 dword i_dtime; // cancellation of files 1/1/1970
 word i_gid; // group id that has access rights
 word i_links_count; // number of references to inode
 dword i_blocks; //
 dword i_flags; // flags of inode
 dword i_osd1; //
 dword i_block[15]; //array to the blocks
 dword i_generation; // virsion of file
 dword i_file_acl; // block that contains extensive attributes
 dword i_dir_acl; // indecates high size of the file
 dword i_faddr; //
 byte l_i_frag; //fragment number
 byte l_i_fsize; // fragment dimesion
 word reserved1; // reserved
 word l_i_uid_high; // user id
 word l_i_gid_high; // group id
 dword reserved2; // reserved
};
```

يحتوي هذا السجل بنية عقد الملفات inode

```
struct i_node_tab
{
  dword i_node_n; // number of inode
  struct i_node inode; // information of inode
  word ref
}
```

يعتبر كذاكرة cache من أجل عقد inode التي تم قراءتها. و المتحول ref عند إضافة العقدة يأخذ القيمة 0x8000 ومن ثم من أجل كل عملية قراءة عقدة ينتقل البت 1 الذي على أقصى اليسار خانة باتجاه اليمين. لباقي العقد و بضع القيمة 0x8000 للعقدة المقروءة. وبالتالي يستخدم هذا المتحول لتحديد العقدة التي هي أقل استخداماً واستدعاءً بين العقد المخزنة في هذه cache.

6 5 5 2 -التوابع

فيما يلي شرح عن عمل بعض التوابع الهامة التي تم استخدامها في أجل تنفيذ التعامل مع نظام الملفات Ext2 ضمن نظام التشغيل:

bool ReadGroupDescriptor(dword grp,struct group_descriptor* data)
يقوم بفحص عدد المجموعات
bool init_group_desc_table()
يهيئ جدول واصف المجموعة Group Descriptor Table
dword Inode2Block(dword ino)
رقم الكتلة التي تحتوي بداية العقدة ino
bool ReadInode(dword ino, struct i_node* data)
قراءة معطيات العقدة ino ضمن المتحول data
bool init_inode_table()
تهيئة المصفوفة inode_table حيث لدينا المتحول dim_inode_table يحتوي العدد الأعظمي للعقد في cache.
int inode_LRU()
يعيد رقم العقدة الأقل استخداماً ضمن cache نظام الملفات، وذلك باستخدام المتحول ref.
struct i_node *get_inode(dword i_node_number)
الحصول على بيانات العقدة إذا كانت مقروءة من قبل يستحصل عليها من cache نظام الملفات.
bool isFile(struct i_node* ino)
يعيد True إذا ما كانت عقدة ملف
bool isFastSymbolicLink(struct i_node* ino)
يعيد True إذا ما كانت عقدة وصلة مرنة
bool isDir(struct i_node* ino)
يعيد True إذا ما كانت عقدة فهرس
dword FindFile(char* cmp)
البحث عن الملف cmp
char *pwd_ext2()
يعيد المتحول path_ext2 والذي يحتوي فهرس العمل الحالي
void ls_ext2()
عرض الملفات المحتواة ضمن الفهرس الحالي
void cd_ext2(char *param)
تغيير الفهرس الحالي إلى الفهرس param
void cat_ext2(char *param)
عرض محتويات الملف param
bool read_ext2()
يقرأ الكتلة الكبرى لنظام الملفات Ext2
bool check_ext2()
يتأكد من أن النظام الملفات صالح وذلك بفحص قيمة magic number لنظام الملفات Ext2
bool init_ext2()

تهيئة متحولات الخاصة بنظام الملفات Ext2 وهي:

حجم الكتلة	dim_block
عدد القطاعات في الكتلة	spb
موضع الكتلة الكبرى	sbpos
عدد الكتل في المجموعة	bpg
حجم واصف المجموعة في الكتلة	gdpb
عدد العقد في الكتلة حيث حجم العقدة 128 بت	ipb
عدد المجموعات	number_of_group
عدد الفهارس في الكتلة	dir_entries_per_block
الفهرس الحالي	ino_current_dir

6 6 -المراجع :

Modern Operating Systems (Chapter 6)	written by: Andrew Tanenbaum Second Edition 1992
Microsoft Extensible Firmware Initiative FAT32 File System Specifecation	written by: Microsoft version 1.03 December 6, 2000
The second extende file system	written by: David Rusling www.science.unitu 2000

7 ـدعم الشبكة

مُقدِّمة

إن الشبكة هي مجموعة من الحواسب المتصلة فيما بينها فيزيائياً بحيث يمكن لأي منها الوصول إلى الآخر واستخدام موارده من تطبيقات وقواعد معطيات وغيرها من المصادر. إن الهدف الدائم من الشبكة هو التشارك في المصادر (Resource sharing) ويُقصد بالمصادر كل ما يمكن لمستثمر يعمل على حاسب ما أن يصل إليه على حاسب آخر وهي تتضمن ما يلي :

- الملفات وقواعد البيانات
- البرامج
- الأقراص الصلبة
- السواقات الليزرية
- سواقات الأقراص المرنة
- المودمات

يتم وصل الحواسيب بواسطة أسلاك تختلف من شبكة إلى أخرى تبعاً للتقنية المعتمدة والبنية المستخدمة. أما عن وصل السلك بالحاسب فيتم عن طريق مأخذ في بطاقة خاصة تزرع في الحاسب ويوصل بها السلك، وتسمى هذه البطاقة بطاقة الواجهة الشبكية Network Interface Card (NIC). عند وصل مجموعة متقاربة من الحواسيب مع بعضها فإننا ندعوها شبكة محلية (Local Area Network) أو اختصاراً LAN ، وتكون هذه الشبكة صغيرة نسبياً وتتباعد عناصرها لمسافات قصيرة تتعلَّق بنوع الأسلاك المستعملة، وهي غالباً شبكة في بناء أو مجموعة متقاربة من الأبنية كأبنية المعاهد والجامعات.

7 .1 ـأهمية الشبكة في نظم التشغيل

هنالك عدد من الأهداف التي يتم من أجلها بناء نظم تشغيل شبكية و تتضمن :

- مشاركة البرامج والملفات **(Program & File Sharing).**
- مشاركة المصادر **(Network Resource Sharing).**
- استعمال برامج شبكية **(Network Software).**
- البريد الإلكتروني **(Electronic Mail).**
- الإدارة المركزية **(Centralized Management).**

7 .1 .1 -مشاركة البرامج والملفات

يمكن وضع ملفات على مخدم مركزي ومشاركة هذه الملفات بحيث يستطيع المستثمرون الوصول إليها من كافة الحواسيب المربوطة إلى الشبكة. إن قواعد البيانات هي من الأمثلة الجيدة على مشاركة البيانات والملفات بين المستثمرين. فإن ما يدخله مستثمر في قاعدة بيانات ما يصبح متاحاً لجميع المستثمرين. كذلك يمكن للمستثمرين أن يخزنوا ملفاتهم ضمن أدلة شخصيةPersonal directories ، ويمكنهم تخزين ملفاتهم ضمن أدلة عامة حيث يستطيع المستثمرون الآخرون قراءة هذه الملفات وتعديلها

7 .1 .2 -مشاركة المصادر

تتضمن مصادر الشبكة الطابعات والراسمات وأجهزة التخزين والمودمات وأنظمة حاسبية أخرى. يمكن التشارك بكل سهولة بهذه المصادر عبر الشبكات .

7 .1 .3 -استعمال برامج شبكية

إن برامج إدارة قواعد المعطيات هي الأكثر شيوعاً للاستعمال على الشبكات. هنالك اليوم برامج تدعى GroupWare وهي مخصصة للاستعمال من قبل مجموعات من المستثمرين الذين يحتاجون للاتصال فيما بينهم عبر الشبكة .

7 .1 .4 -البريد الإلكتروني

يستعمل البريد الإلكتروني لإرسال الرسائل أو الوثائق إلى مستثمرين أو مجموعات من المستثمرين على الشبكة. ويمكن للمستثمرين أن يتصلوا مع بعضهم البعض. توضع الرسائل ضمن علب البريد الخاصة بكل مستخدم mailbox ويتم إعلام المستثمر كلما وصل بريد جديد .

7 .1 .5 -الإدارة المركزية

تكون أغلب مصادر الشبكة عادةً مركزة حول المخدم، مما يجعل إدارة هذه المصادر أمراً سهلاً إذ تصبح الإدارة مركزية في موقع واحد .

7 .2 -بطاقة الشبكة

إن بطاقة الشبكة (Network Adapter Card) تدعى أحياناً Network Interface Card (NIC)هي بطاقة مادية يتم تنصيبها ضمن الحاسب وتسمح له بالعمل على الشبكة الموصولة إليها. تؤمن بطاقة الشبكة الربط الفيزيائي بين الحاسب والشبكة. إن معظم بطاقات الشبكة هي بطاقات داخلية يتم تنصيبها ضمن الحاسب في إحدى فتحات التوسيع (Expansion Slot). لبطاقة الشبكة دور كبير في تحديد سرعة النقل على الشبكة وأداؤها. تقوم البطاقة بنقل البيانات من الحاسب إلى سلك الشبكة وبالعكس. فهي تقرأ بشكل دائم الإشارات الكهربائية الموجودة على السلك وتختبرها ثم تحولها إلى صيغة حاسبية وتقدمها إلى نظام التشغيل كذلك عند الإرسال فإنها تحول البيانات المرسلة من نظام التشغيل إلى إشارات كهربائية مناسبة ثم تضعها

على السلك. هنالك عدد من أنواع البطاقات المتوفرة التي يمكن استخدامها وأشهرها بطاقات "إثرنت" Ethernet وبطاقات "توكن رينغ" Token Ring وبطاقات "لوكال توك LocalTalk". تشير الإحصائيات إلى أن النوع "إثرنت" هو أكثرها انتشاراً .

7 2 1- أنواع الشبكات

هنالك ثلاثة أنواع شائعة للشبكات :

- شبكة اثرنيت Ethernet.
- شبكة آرك نيت ArcNet.
- شبكة حلقة العلام Token Ring.

7 2 2- بطاقة الشبكة "Ethernet"

بطاقة "Ethernet" هي أكثر أنواع البطاقات استخداماً في عالم التشبيك وهي يمكن أن تحوي مداخل لأسلاك الأزواج المجدولة أو الأسلاك المحورية أو أسلاك الألياف الضوئية. كي يمكن استخدام البطاقة مع أسلاك الأزواج المجدولة يجب أن تملك البطاقة مخرجاً من النوع RJ-45ولاستخدام الأسلاك المحورية النحيفة يجب أن تملك مخرجاً من النوع BNC. يمكن أن تملك البطاقة كذلك مخرجاً من النوع AUI وهذا يسمح باستخدام أسلاك من الأنواع الثلاثة المذكورة: الأزواج المجدولة والمحورية النحيفة والثخينة والألياف الضوئية. عند استخدام المخرج AUI فيجب استخدام موائم (Adapter) وظيفته التحويل بين نمط البيانات على المخرج AUI ونمط البيانات على السلك الموصول. ندعو هذا الموائم ترانسيفر.(Transceiver) يبين الشكل بطاقة "إثرنت "تملك الأنواع الثلاثة من المخارج وهي من الأعلى إلى الأسفل: مخرج RJ-45 ومخرج AUIومخرج.BNC

بطاقة "إثرنت" وتبدو عليها مخارجها

الشكل 63

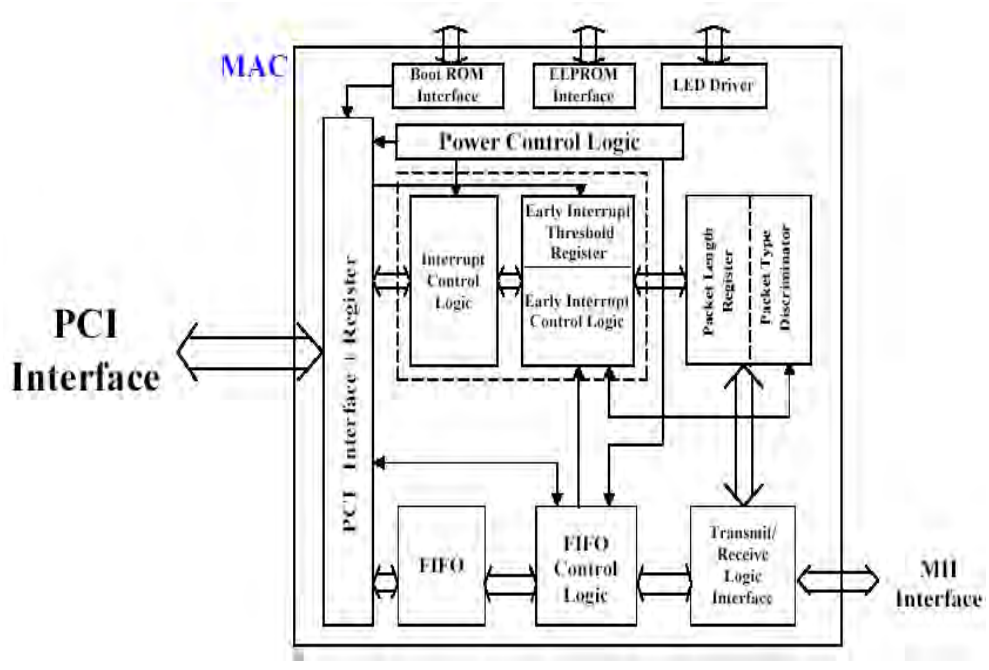
اثرنيت هي بروتوكول اتصال منطقي يسمح لعدة أجهزة أن تتصل مع بعضها من خلال العديد من الوسائط الفيزيائية. هذا البروتوكول المنطقي يدعى "نفاذ متعدد بتحسس

الحامل/كشف التصادم "Carrier Sense Multiple Access with Collision Detection" (CSMA/CD) تتصل كل المحطات الطرفية بشكل مباشر مع نفس السلك المستعمل لنقل البيانات بين أي محطتين، لذلك نقول أن السلك يعمل بنمط النفاذ المتعدد (Multiple Access). تقوم المحطة المرسلة بتضمين البيانات المراد إرسالها ضمن طرد (Frame) وتضع ضمنه عنوان المحطة المرسل إليها. ثم تقوم ببث الطرد على السلك.كل المحطات الأخرى الموجودة على نفس السلك تكتشف أن طرداً قد تمّ وضعه على الشبكة. عندما تكتشف محطة ما أن عنوانها موجود في مقدمة الطرد تستمر بقراءة المعلومات ضمن الطرد. وتستجيب تبعاً لنوع البروتوكول المستخدم. إن عنوان المحطة المرسلة موجود أيضاً ضمن الطرد.بهذا الأسلوب يمكن لمحطتي عمل أن تحاولا إرسال الطرود على السلك في نفس الوقت مما يسبب تشوه معلومات الطرود. لتقليل هذا الاحتمال، تقوم المحطة المرسلة بالتصنت على السلك قبل وضع الطرد على السلك لتكتشف إن كان يوجد نشاط على الشبكة أم لا.

إذا تم تحسس وجود إشارة على الحامل، فإن المحطة تؤجل عملية الإرسال مدة عشوائية من الزمن ثم تعيد الكرة. ومع ذلك فإن محطتان قد ترغبان بالإرسال وتقرران في نفس الوقت، أنه لا يوجد نشاط على السلك وتبدآن بالإرسال فيحصل ما يدعى بالتضارب (Collision) وتشوه المعطيات. لاكتشاف التضارب تقوم كل محطة بمراقبة إشارة المعطيات على السلك عند إرسال محتويات الطرد، إذا كان الطرد المرسل مختلف عن الطرد المراقب فهذا يفترض وجود تضارب. لضمان أن المحطة الأخرى (المتورطة في التضارب) قد لاحظت التضارب، تقوم المحطة الأولى بتعزيز هذا التضارب عن طريق الاستمرار بإرسال سلسلة من البتات العشوائية لمدة قصيرة وهو ما يدعى بسلسلة الازدحام (Jam Sequence). تنتظر المحطتان لمدة عشوائية قصيرة قبل أن تقررا الإرسال من جديد.

7 2 3 -بنية متحكم الشبكة Ethernet Controller

يبين الشكل التالي بنية متحكم الشبكة:



الشكل 64

7.2.3.1- وصف المسجلات

سنقدم فيما يلي شرحاً لأهم المسجلات ضمن متحكم الشبكة Ethernet و لمزيد من المعلومات يمكن مراجعة Datasheet.

7.2.3.2- مسجل حالة الاستقبال (Receive Status Register)

7.2.3.3 -

الخانة	R/W	الوصف
15	R	عندما تكون قيمة هذه الخانة مساوية 1 فان حزمة بيانات قد استقبلت
14	R	عندما تكون قيمة هذه الخانة مساوية 1 فان حقل الجهة المستقبلة بالحزمة يطابق قيمة ID Registers
13	R	عندما تكون قيمة هذه الخانة مساوية 1 فان حزمة بيانات معممة قد استقبلت
12-6	-	محجوزة
5	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يشير إلى وقوع خطأ خلال استقبال الحزمة
4	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يدل على أن الحجم الحزمة المستقبلة اقل من 64 بايت
3	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يشير إلى أن حجم الحزمة المستقبلة يتجاوز 4 كيلو بايت
2	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يدل على حدوث خطأ CRC في الحزمة المستقبلة
1	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يشير إلى حدوث خطأ في إطار في الحزمة المستقبلة
0	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يشير إلى أن عملية الاستقبال تمت بنجاح

7 2 3 4 -2. مسجل حالة الارسال (Transmit Status Register)

الخانة	R/W	الوصف
31	R	عندما تكون قيمة هذه الخانة مساوية 1 فان الأمل قد فقد خلال إرسال الحزمة
30	R	عندما تكون قيمة هذه الخانة مساوية 1 فان عملية الارسال قد قطعت
29	R	عندما تكون قيمة هذه الخانة مساوية 1 فان هذا يدل على حدوث Out of window collision
28	R	CD Heart Beat
27-24	R	يدل على عدد التصادمات التي حدثت خلال عملية الارسال
23-22	-	محجوز
21-16	R/W	يحدد مستوى عتبة في TX FIFO لبداية عملية الارسال
15	R	عندما تكون قيمة هذه الخانة مساوية 1 فان عملية الارسال تمت بنجاح
14	R	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يدل على أن TX FIFO قد انتهك خلال عملية إرسال الحزمة
13	R/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يدل على أن عمليات DMA قد انتهت
12-0	R/W	الحجم ألأعظمي بالبايت للبيانات ضمن الـDescriptor

7 2 3 5 -3. مسجل الأوامر (Command Register)

لخانة	/W	الوصف
-5		محجوز
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يؤدي إلى تعطيل عمليات الارسال و الاستقبال.
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يؤدي إلى تفعيل عمليات الاستقبال
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يؤدي إلى تفعيل عمليات الارسال
		محجوز
		عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يدل على أن ذاكرة RX فارغة

7 2 3 6 - مسجل حالات المقاطعة (Interrupt Status Register)

يشير هذا المسجل إلى مصدر المقاطعة و يعكس حالة المقاطعة.

لخانة	/W	الوصف
5	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذلك يشير إلى حدوث خطأ في ممر PCI
4	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان يشير إلى حدوث Time Out
3	/W	طول الكبل قد تغير بعد تفعيل الارسال
2-7		محجوز
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان Rx FIFO Overflow
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان حالة الوصلة تكون قد تغيرت
	/W	عندما تكون قيمة هذه الخانة مساوية 1 فان ذاكرة RX قد امتلأت Rx buffer overflow
	/W	يشير إلى حدوث خطأ CRC ضمن الحزمة المرسلة
	/W	يشير إلى استكمال عملية الارسال بنجاح
	/W	يشير إلى حدوث خطأ CRC ضمن الحزمة المستقبلة
	/W	يشير إلى استكمال عملية الاستقبال بنجاح

7 2 4 -الوصف الوظيفي

7 2 4 1- 1. عمليات الإرسال

يقوم المعالج CPU بتهيئة عملية الإرسال بتخزين كافة البيانات المراد إرسالها في الذاكرة الرئيسية. بعدها عند الانتهاء من إرسال كافة البيانات إلى ذاكرة Tx يقوم متحكم Ethernet بنقل البيانات من ذاكرة Tx إلى FIFO الإرسال الداخلي و عند وصول كافة البيانات أو الوصول إلى مستوى العتبة يقوم المتحكم بإرسال البيانات.

7 2 4 2- 2. عمليات الاستقبال

حزمة البيانات المستقبلة يتم وضعها في Rx FIFO. بعدها يقوم متحكم Ethernet بإنجاز عملية فلترة العنوان للحزمة حسب خوارزمية معينة وعند وصول كمية البيانات المستقبلة إلى المستوى المعرف يطلب المتحكم من ممر PCI نقل البيانات إلى ذاكرة Rx.

7 2 5 -بطاقة الشبكة كبطاقة PCI

تمتلك جميع بطاقات PCI (ومن ضمنها بطاقة الشبكة)عدداً من التوابع من اجل عمليات التهيئة و الإعداد ومعالجة الأخطاء القاتلة.

- **VID**: يسمى هذا الحقل رقم المصنع **Vender ID** حيث يحوي على قيمة رقم المصنع في **EEPROM** الخارجية.
- **DID**: يسمى هذا الحقل رقم الجهاز **Device ID** حيث يحوي على قيمة رقم الجهاز في **EEPROM** الخارجية.
- **Command:**: وهو مسجل بـ15 خانة ويؤمن تحكم على قدرات الجهاز لتوليد و استجابة دورات **PCI** **(PCI cycles)**.
- **RID**: يسمى هذا الحقل رقم التنقيح **(Revision ID)** وهو مسجل بـ8 خانات و الذي يحدد رقم تنقيح المتحكم.
- **PIFR**: وهو مسجل الواجهة البرمجية **Programming Interface** وهو مسجل بـ8 خانات و الذي يعرف الواجهة البرمجية للمتحكم.
- **SCR**: مسجل الصنف الفرعي **Sub-Class** و هو مسجل بـ8 خانات الذي يعرف وظيفة الجهاز و القيمة **SCR=00h** تدل على أن الجهاز هو **Ethernet Controller**.
- **BCR**: و مسجل الصنف الفرعي **Base-Class** و هو مسجل بـ8 خانات الذي يعرف الوظيفة العامة الجهاز و القيمة **BCR=02h** تدل على أن الجهاز هو متحكم الشبكة.
- **CLS**: يمثل حجم سطر الكاش.
- **LTR**: مسجل مؤقت التأخير **Latency Timer** ويحد وحد تردد ساعة ممر **PCI**.
- **HTR**: مسجل نوع الرأس **Header Type**.
- **BIST**: مسجل الفحص الذاتي **Built-in Self Test**.
- **IOAR**: يحدد هذا المسجل عنوان الدخل/الخرج الأساسي المستخدم في بناء خارطة العناوين.
- **SVID**: يسمى هذا الحقل رقم المصنع للنظام الفرعي **Subsystem Vendor ID**.

- **SMID**: يسمى هذا الحقل رقم النظام الفرعي **Subsystem ID**.

ILR: مسجل سطر المقاطعة Interrupt Line Register وهو مسجل ب_8 خانات يستخدم للتواصل مع برامج خدمة المقاطعة.

أصناف PCI:

هنالك عدة أصناف مدعومة من قبل PCI ولكل منها شفرة معينة تستخدم للتعرف على الجهاز والى أي صنف ينتمي فيما يلي نذكر بعض منها:

الأصناف الأساسية:

الصنف	الوصف
0x01	أجهزة التخزين الصلبة
0x02	أجهزة الشبكة
0x03	أجهزة العرض
0x04	أجهزة الوسائط المتعددة
0x05	متحكمات الذاكرة
0x06	الجسور Bridges

الأصناف الفرعية لأجهزة الشبكة:

7-2-5-1- الصنف	7-2-5-2- الوصف
0x00	Ethernet
0x01	Token Ring
0x02	FDDI
0x03	ATM
0x04	ISDN
0x80	Other

7 3 -نموذج OSI

يتألف نموذج OSI من الطبقات السبعة التالية:

- Application
- Presentation
- Session
- Transport
- Network
- Data link
- Physical

كل طبقة تعرف عدد من التوابع أو الوظائف (Functions) التي نحتاجها في الاتصال.تقسم توابع OSI إلى مجموعتين من الطبقات الطبقة العليا و الطبقة السفلى. تتضمن الطبقة العليا (و التي تتضمن الطبقات 4،5،6،7) خدمات متعلقة بالتطبيقات مثل متصفحات الانترنت. أما الطبقة السفلى (والتي تضمن الطبقات 1،2،3) خدمات الاتصال التي تحدث بين الطرفين. وبمعنى آخر إذا كانت الخدمة هي معالجة بيانات المستخدمة فهي في الطبقة العليا من نموذج OSI أما إذا كانت الخدمة هي نقل للبيانات فهي في الطبقة السفلى من نموذج OSI.

7 3 1- الطبقة السابعة التطبيقات

تؤمن هذه الطبقة خدمات الشبكة المتعلقة بالتطبيقات. جميع التطبيقات تكون متضمنة في هذه الطبقة وعلى كل تطبيق أن يوظف برتوكوله الخاص ليتواصل مع برتوكولات الطبقة السفلى Lower-Layer Protocols (LLPs).

7 3 2- الطبقة السادسة العرض

هذه الطبقة مسئولة عن ترميز المعطيات المتبادلة بين الطرفين حيث يمكن أن يتخاطب حاسبين من نمطين مختلفين وبالتالي لابدَّ من ترميز مشترك يتم تبادله يتوافق مع الحاسبين معاً. غالباً تستخدم جميع الأنظمة نموذج ASCII لتمثيل البيانات ومكن إظهارها بغض النظر عن منصة العمل.

7 3 3- الطبقة الخامسة الجلسة

تدير الطبقة الخامسة عمليات تبادل البيانات بين التطبيقات. حيث تتحكم بالاتصال و تتحكم بالتدفق و تزود خدمة الكشف عن الأخطاء.

7 3 4- الطبقة الرابعة النقل

تتأكد الطبقة الخامسة من سلامة البيانات Integrity كما تبقي التطبيقات على علم بمجريات الاتصال. أما الطبقة الرابعة فهي تهتم بعملية النقل من نقطة إلى نقطة. عملية النقل

يمكــن أن تــدار مــن خــلال اتصــال موجــه Connection-oriented آو غيــر موجــه Connectionless.عملية النقل من خلال الاتصال الموجه يجب أن يكون قادرا على تنفيذ توابع معالجة البيانات التالية:

- التوزيع (Multiplexing): حيث يجب أن يكون قادر على نقل البيانات دخول و خروجا من الطبقة الثالثة.
- التقسيم (Segmenting): في معظم الحالات نحتاج لإرسال البيانات ضمن وحدات متعددة. حيث يتم تقسيم البيانات إلى إجراء و إعادة تجميعها في الجهة المستقبلة.
- التجميع (Blocking): بعض البيانات تكون مقسمة إلى أجزاء صغيرة جدا لذلك من الأفضل تجميعها ضمن وحدة بيانات واحدة و من ثم إعادة تقسيمها في الجهة المستقبلة.
- (Concatenating): و هي عملية وضع عدة وحدات من البيانات ضمن حامل واحد في الطبقة الثالثة ومن ثم تفكيكها في الجهة المستقبلة.
- الكشف عن الأخطاء وإصلاحها: حيث يجب أن يكون هنالك طريقة للكشف عن حدوث أي ضرر للبيانات عن وصولها إلى الطبقة الثالثة بالإضافة لوجود طريقة لإعادة الارسال.
- التحكم بالتدفق (Flow Control):حيث يجب أن يكون هنالك طريقة للتنظيم نقل البيانات وتمرريها إلى الطبقات المجاورة.
- نقل سريع للبيانات: حيث يجب أن يؤمن خدمة نقل خاصة لأنواع محددة من البيانات وتجاهل التحكم بالتدفق.

أما الاتصال غير الموجه و الذي يسمى Datagram Transport فهو لا يحتاج إلى عمليات تكامل البيانات أو الكشف عن الأخطاء، ويستخدم في حالات النقل السريع للبيانات مثل نقل الصوت.فعليا تبدأ الشبكة في الطبقات السفلى من نموذج OSI (1،2،3).

7 3 5 -الطبقة الثالثة الشبكة

في طبقة الشبكة تحدث عملية النقل الفعلية لوحدات البيانات. حيث تؤمن هذه الطبقة خدمات التوصيل و العنونة التي نحتاجها لنقل وحدات البيانات بيم الطرفين. ويتم إنجاز ذلك بواسطة سلسلة البيانات ضمن حزم Packets و إضافة معلومات التوصيل و عنوان المرسل و المستقبل وأي معلومات أخرى نحتاجها.

كما أن الطبقة الثالثة مسئولة عن توصل الحزم التي تحتاج إلى بعض خدمات التوجيه. عمليات التوجيه تتضمن عمليتان التحويل و التوجيه. حيث أن التوجيه هو عملية البحث و إيجاد معلومات الشبكة. و التحويل هو استخدام هذه المعلومات لنقل البيانات.باستخدام النقل بالاتصال الموجه، الإعلام بإيصال الحزم مطلوب بعد كل عملية توصيل. أما الاتصال غير الموجه فان الحزم تصل خلال أفضل طريق معروف.

7 3 6 -الطبقة الثانية ربط البيانات

تؤمن الطبقة الثانية سهولة بالتحكم في نقل بتات البيانات من برتوكولات الطبقة العلياUpper Layer Protocol (UPL) إلى الطبقة الفيزيائية. حيث يتم تغليف حزم البيانات ضمن الطبقة الثانية ضمن إطارات Frames ثم بعد ذلك يتم إرسالها. تملك الطبقة الثانية تابعين

لنقل البيانات. الأول يدعى Media Access Control (MAC) و الذي يعرف التمثيل المنطقي لبيانات ULP. اما التابع الثاني يدعى Link Control (LC) و الذي يعمل كواجهة بين برتوكولات الطبقة الثالثة و MAC. وفي حالة الشبكات المحلية LAN يعرف معيار IEEE الأشياء التي يجب أن ينفذها هذا التابع حسب نوع الشبكة.

7 3 7 -الطبقة الأولى الفيزيائية

تتعامل الطبقة الفيزيائية مع موصفات وسط النقل المستخدم لنقل بت بيانات من نقطة إلى نقطة.جميع الوسائط الفيزيائية تملك معايير كنوع الوصل و طول الكبل وآية أمور أخرى مهمة.

من أهم معايير الطبقة الفيزيائية:

- IEEE 10-BaseT 802.3 Ethernet
- IEEE 100-BaseT 802.3 Fast Ethernet

TCP/IP Stack- 4 7

البروتوكول TCP/IP هو مجموعة من البروتوكولات يطلق عليها طقم بروتوكولات TCP/IP وقد تمَّ البدء بتطويره عام 1970 من قبل وكالة مشاريع الأبحاث المتقدمة الدفاعية (DARRPA) في وزرة الدفاع الأمريكية بهدف ربط عدد من الشبكات مع بعضها. إن أشهر بروتوكولين في طقم بروتوكولات TCP/IP هما البروتوكول TCP ويدعى بروتوكول ضبط النقل (Transmission Control Protocol) والبروتوكول IP ويدعى بروتوكول إنترنت (Internet Protocol). الميزة الأساسية لهذا البروتوكول هو أنه يوفر إمكانية الاتصال عبر شبكات عديدة مرتبطة مع بعضها وإمكانية الاتصال بين أنظمة تشغيل مختلفة وبني عتادية مختلفة، مثل UNIX و Windows NT و Windows 2000 و Novell Netware وغيرها. كذلك يتوافق TCP/IP مع شبكة الإنترنت ويعتبر هو البروتوكول الرئيسي والوحيد على هذه الشبكة.

من أهم ميزاته:

- يؤمن الاتصال بين عدة أنظمة تشغيل وبنت عتادية مختلفة، فهو متوفر في كافة أنظمة التشغيل.
- يدعم تقنيات التوجيه.
- البروتوكول الوحيد المستخدم على شبكة الإنترنت.
- مرن بشكل كبير ويمكن التحكم بأدائه.
- يدعم بروتوكول SNMP.

7 4 1- البروتوكول TCP

يؤمن البرتوكول TCP اتصالا موجها لنقل البيانات. يعمل هذه البرتوكول على أي نوع من أنواع الشبكات وصمم أيضا ليؤمن نقل موثوق للبيانات بين الحواسيب التي قد تملك شبكة غير موثوقة.

يؤمن TCP الخدمات التالية لتحقيق عملية النقل الموثوق:

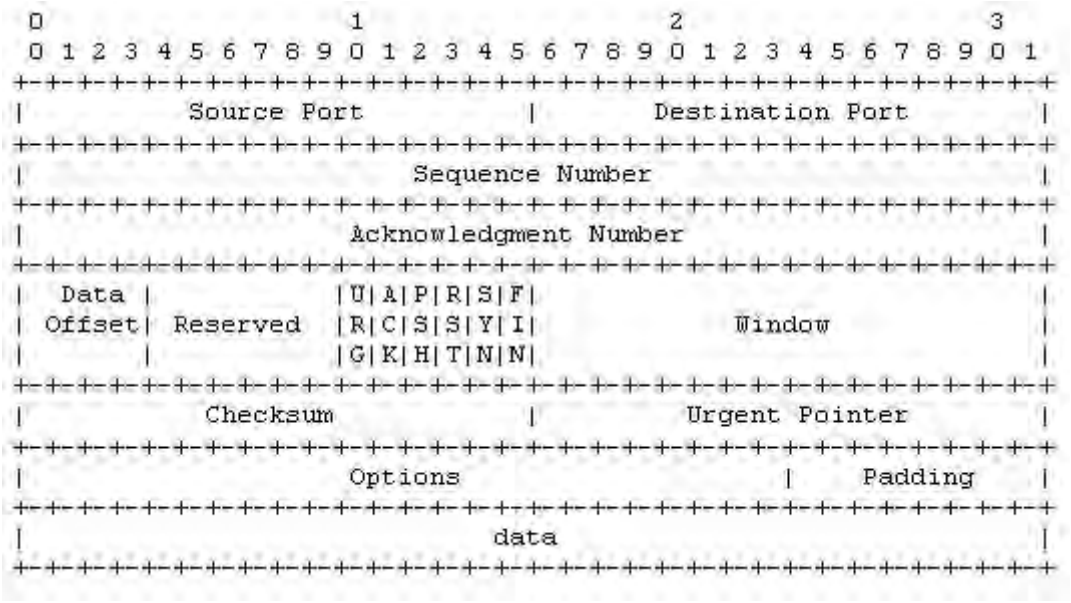
النقل في الاتجاهين معا (Full duplex): يقوم TCP بإجراء اتصال ذو اتجاهين ليؤمن عملية مرور البيانات من والى الطبقة العليا.
سلسلة البيانات (Data sequencing):.يتم تسليم البيانات إلى الطبقة العليا بنفس الترتيب الذي أرسلت به، حيث يقوم بترقيم البيانات المرسلة حيث يستخدم هذا الرقم في عملية تجميع البيانات عند الاستقبال. ولا يتم تسليم البيانات إلى الطبقة العليا حتى وصول جميع البيانات المرسلة.

التحكم بالتدفق (Flow control): يستخدم TCP إستراتيجية النافذة المنزلقة في عملية إرسال البيانات يقوم TCP أولا بتحديد سرعة الاتصال ثم بعدها يقوم بإنشاء نافذة يعتمد حجمها على سرعة الاتصال بين المصدر و الهدف. يستخدم TCP النوافذ لتأطير Frame كمية البيانات التي يجب إرسالها في كل مرة ويستخدم الرقم التسلسلي لمحافظة على الترتيب. بعد إرسال البيانات ينتظر TCP إعلام الجهة المستقبلة بنجاح عملية الارسال. يستخدم TCP المؤقتات Timers (لقياس الزمن بين عملية الارسال واستقبال الأعلام بنجاح عملية الارسال).

تصحيح الأخطاء (Error correction): يستخدم TCP تقنية Checksum ليتأكد من سلامة البيانات.

7 4 1 1- TCP Header

يؤمن TCP Header وسلية لتواصل بين نقطتين يستخدمان TCP لنقل البيانات. كما يؤمن TCP Header ترقيم البيانات و معلومات الإعلام ويسهل عملية إنشاء الاتصال و قطعه.



الشكل 65

يمكن أن يكون حجم حزمة TCP من 65KB فما فوق. فيما يلي شرح مبسط لحقول حزمة TCP:

- Source Port: يدل هذا الحقل على مصدر البيانات.
- Destination Port: يدل على الحقل على وجهة البيانات.
- Sequence Number: الرقم التسلسلي للبيانات المحتواة ضمن الحزمة.
- Acknowledgment Number: يحوي هذا الحقل الرقم التسلسلي الذي يتوقع مرسل الحزمة استقباله من الوجهة.
- Data Offset: يدل هذا الحقل على حجم TCP Header.
- Control Flags: تشير إلى حالة الاتصال:
- SYN: إعداد اتصال TCP.
- ACK: تشير فيما إذ كانت المعلومات ضمن حقل الإعلام Acknowledgment ذات علاقة.
- RST: إعادة تهيئة اتصال TCP.
- PSH: يخبر هذا الحقل الوجهة بأنه يتوجب تسليم البيانات المستقبلة إلى الطبقة العليا مباشرة.
- FIN: إنهاء اتصال TCP.
- Window: يستخدم هذا الحقل لتأمين معلومات تحكمية. تكون قيمة هذا الحقل كمية البيانات التي يستطيع المرسل استقبالها.

TCP Connection Setup- 2 1 4 7

إن عملية الاتصال تمر بسلسلة من الحالات خلال حياة الاتصال lifetime هذه الحالات هي: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, و الحالة الخيالية CLOSED.

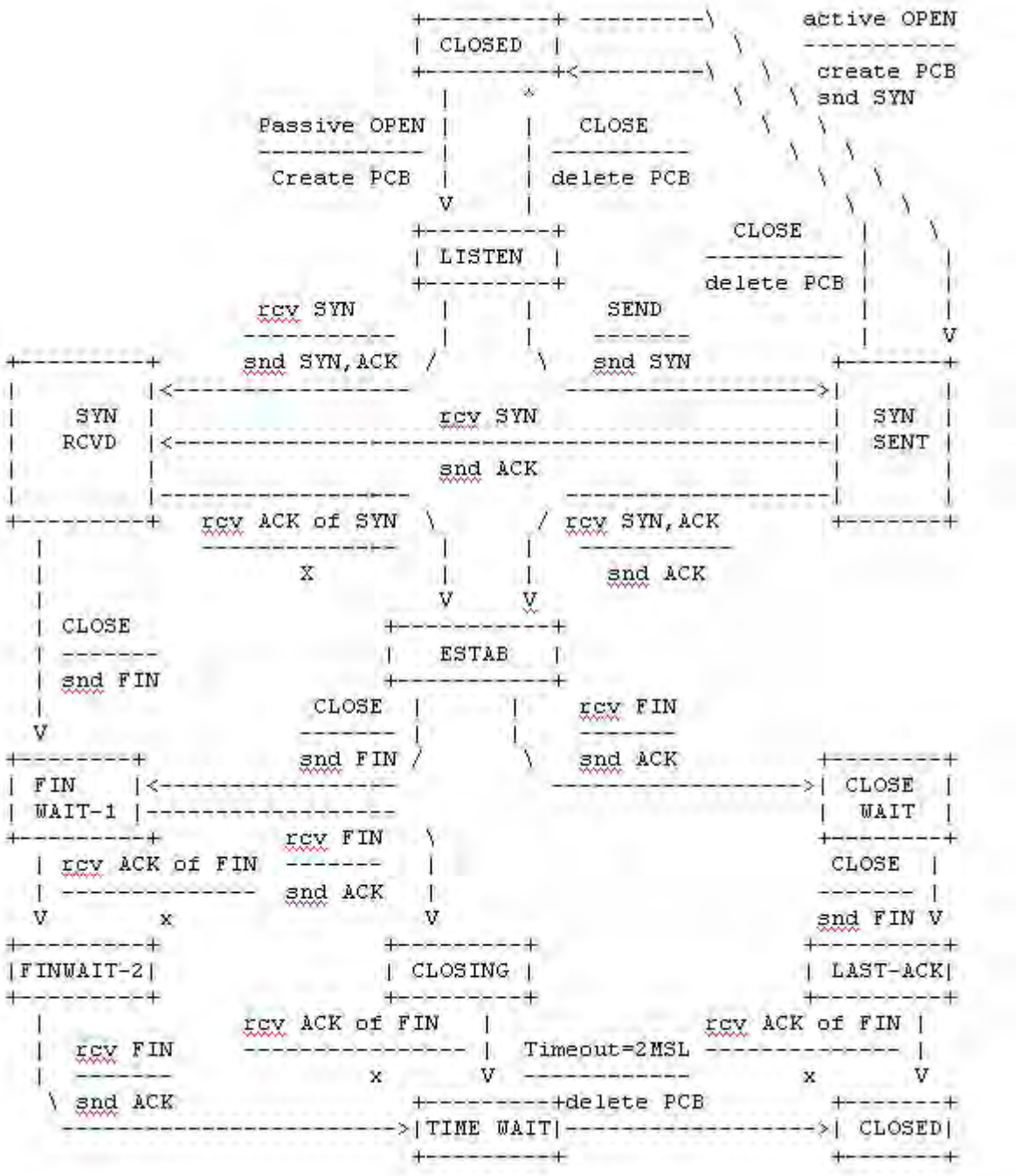
تعتبر CLOSED حالة خيالية لأنها تمثل الحالة عندما لا يكون هنالك كتلة تحكم بالبرتوكول (PCB) وبالتالي إنهاء الاتصال.

فيما يلي ملخص عن معاني هذه الحالات:

- **LISTEN**: تمثل حالة انتظار لطلب اتصال من أي وجهة بعيدة.
- **SYN-SENT**: تمثل حالة انتظار لطلب اتصال مماثل بعد إرسال طلب اتصال.
- **SYN-RECEIVED**: تمثل حالة انتظار إعلام بطلب الاتصال بعد استقبال و إرسال طلب الاتصال.
- **ESTABLISHED**: تمثل حالة فتح الاتصال حيث يتم تسليم البيانات المستقبلة إلى المستخدم و هي الحالة الطبيعية التي يتم فيها نقل البيانات خلال فترة الاتصال.
- **FIN-WAIT-1**: تمثل حالة انتظار طلب إنهاء الاتصال من الوجهة البعيدة أو إعلام بطلب إنهاء اتصال قد أرسل من قبل.
- **FIN-WAIT-2**: تمثل حالة انتظار طلب إنهاء الاتصال من الوجهة البعيدة.
- **CLOSE-WAIT**: تمثل حالة انتظار طلب إنهاء الاتصال من المستخدم المحلي.
- **CLOSING**: تمثل حالة انتظار إعلام بطلب إنهاء الاتصال من الوجهة البعيدة.
- **LAST-ACK**: تمثل حالة انتظار إعلام بطلب إنهاء اتصال قد أرسل من قبل إلى الوجهة البعيدة.
- **TIME-WAIT**: تمثل حالة انتظار لوقت كافي ليتم التأكد من أن الوجهة البعيدة قد استلمت إعلامها بطلب إنهاء الاتصال.
- **CLOSED**: تمثل حالة عدم وجود اتصال.

يتم الانتقال من حالة إلى حالة خلال عملية الاتصال كاستجابة إلى الإحداث. هذه الإحداث هي عبارة عن استدعاء المستخدم للاجرائيات OPEN, SEND, RECEIVE, CLOSE, ABORT, STATUS بالإضافة إلى الحزمة المستقبلة التي تحوي على الأعلام SYN, ACK, RST, FIN بالإضافة إلى TIMEOUT.

مخطط الحالة المبين في الشكل التالي يظهر فقط تغير الحالات عن طريق الإحداث لكنه لا يبين كيفية معالجة الأخطاء.



الشكل66

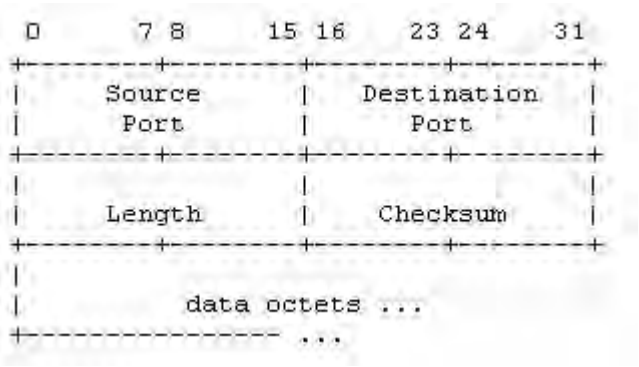
7 4 2 -بروتوكول UDP

يؤمن UDP عمليات النقل المعتمدة على الاتصال غير الموجهة بمعنى أن الاتصال غير الموثق. وهو لا يضمن وصول البيانات بنفس الترتيب الذي أرسلت به وهو عكس TCP ولكن هذا البرتوكول له مميزات تجعل يستحب استخدامه في بعض الحالات مثل عند إرسال

بيانات جماعية عامة وعند الحاجة إلى السرعة حيث لا يملك UDP أي معلومات تحكمية. ومن أهم الأسباب التي أدت إلى إنشاء البروتوكول UDP أن الإرسال عبر هذا البروتوكول لا يتطلب إلا القليل من الحمل و الوقت إذ أن حزمة UDP لا تحتوي على كل البيانات التي ذكرت مع البروتوكول TCP للتحكم بعملية الإرسال.

7 4 2 1- ترويسة UDP UDP Header

بين الشكل التالي بنية حزمة UDP:



الشكل67

فيما يلي شرح لحقول UDP Header:

- Source Port: يدل هذا الحقل على مصدر البيانات.
- Destination Port: يدل على الحقل على وجهة البيانات.
- Length: يحوي هذا الحقل على حجم الحزمة.

7 4 3- بروتوكول IP

يعتبر برتوكول IP البرتوكول المسؤول عن تسليم حزم البيانات المشكلة من قبل برتوكولات الطبقة العليا. هذه البيانات يمكن أن تكون TCP أو UDP.و يؤمن تقسيم و إعادة تجميع البيانات الطولية إذا كان ذلك ضروريا. ولا يهتم برتوكول IP بنوع البيانات التي يجب أن يرسلها و يمكن أن يتكيف مع أي حجم لحزم بيانات الطبقة الثانية، التي يتم تقسيمها إلى حجم إطار مدعوم من قبل الطبقة الثانية. ثم بعد ذلك يتم إرسال هذه الأجزاء إلى الوجهة المحددة حيث يتم تجميعها قبل تسليمها إلى برتوكولات الطبقة العليا. هذا بالإضافة إلى أن برتوكول IP لا يضمن تسليم البيانات وفي حال ضياع أية حزم فهي مسؤولية برتوكولات الطبقة العليا الموجودة بالجهة المرسلة لإعادة إرسالها مرة أخرى.

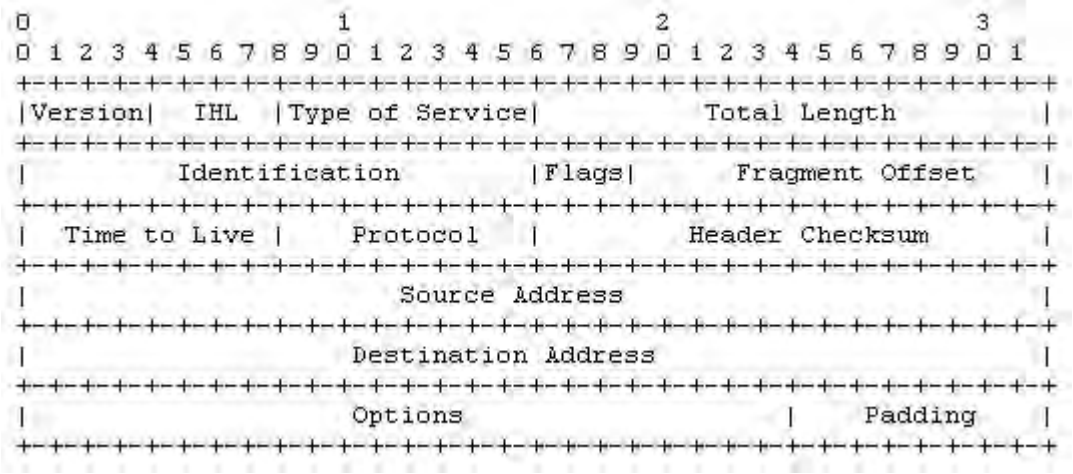
7 4 3 1- علاقة برتوكول IP (الطبقة الثالثة) مع برتوكول ARP (الطبقة الثانية)

تعتمد الطبقة الثالثة في النقل الفعلي للبيانات على الطبقة الثانية حيث تؤمن الطبقة الثانية عملية ترجمة عناوين الشبكة في الطبقة الثالثة IP network address إلى عناوين من اجل نقل البيانات. حيث صمم برتوكول ARP لانجاز هذه المهمة.يقوم برتوكول IP لتجميع

البيانات القادمة إليه من برتوكولات الطبقة العليا كما يحصل على عنوان الوجهة بعدها يقوم بالبحث في جدول التوجيه Host route table ليحدد فيما إذا كانت الوجهة هي محلية أو بعيدة، فإذا كانت الوجهة محلية يقوم بتسليم البيانات مباشرة إلى الحاسب المحلي و إلا يقوم بإرسالها إلى الوجهة البعيدة. وفي كلا الحالتين يتم تغليف حزم IP ضمن حزم ARP.

إن الوظيفة الأساسية للبرتوكول IP (و جميع برتوكولات الطبقة الثالثة) هي تأمين واجهة تراسل عامة بين برتوكولات الطبقة العليا ULP و بين وسط التراسل في الطبقة الثانية. هذا يؤدي إلى قدرة على نقل البيانات بطريقة مستقلة عن برتوكول الارسال. بمعنى يمكننا تبادل البيانات بين أوساط نقل مختلفة و باستخدام عناوين مختلفة وتنسيق بيانات مختلف. لا تظهر هذه الميزة ضمن الشبكات المحلية LAN لأنه على الأغلب يكون أوساط النقل متطابقة ولكن تظهر عند الارسال إلى وجهة بعيدة جدا حيث تمر البيانات ضمن أوساط نقل مختلفة.

2 3 4 7 -ترويسة IP Header IP



الشكل68

- **Version** (الحجم 4 بت): يعرّف هذا الحقل نسخة **IP** التي أنشأت حزمة البيانات هذه لذلك تستطيع جميع الجهات معرفة تنسيق هذه الحزمة لمعالجتها.
- **IHL** (الحجم 4 بت): يصف هذا الحقل (**Internet Header Length**) الحجم الكلي لـ **IP Header**.
- **Type of Service** (الحجم 8 بت): يؤمن هذا الحقل على تعريف خدمات معالجة.
- **Total Length** (الحجم 16 بت): يصف هذا الحقل الحجم الكلي لحزم البيانات المتضمنة بيانات **ULP** و **IP Header**.
- **Identification** (الحجم 16 بت): تخبر قيمة هذا الحقل الجهة المستقبلة أي قسم من البيانات المجزأة يحوي على البيانات.
- **Flags** (الحجم 3 بت): تحدد قيمة هذا الحقل فيما إذا كان التقسيم مسموح أو لا.
- **Fragment Offset** (الحجم 13 بت): يستخدم هذا الحقل لتحديد موقع الحزمةالمجزئة ضمن كامل البيانات حيث تكون الإزاحة للحزمة الأولى هي 0.
- **Time to Live** (الحجم 8 بت): يستخدم هذا الحقل لحذف الحزم الضائعة
- **Protocol** (الحجم 8 بت): يؤمن معلومات عن البيانات المحتواة ضمن الحزمة.

- **Header Checksum** (الحجم 16 بت): يحوي هذا الحقل على الـ**Checksum** لـ **IP Header**.
- **Source Address** (الحجم 32 بت): يحوي عنوان الجهة المرسلة.
- **Destination Address** (الحجم 32 بت): عنوان الجهة المستقبلة.
- **Option** (الحجم متغير): حقل اختياري يستخدم من اجل أغراض التطوير و التنقيح.
- **Padding** (الحجم متغير): هذا الحقل يستخدم لإضافة بيانات إضافية حتى يتلاءم حجم الحزمة المرسلة مع الحجم المطلوب للرسم.

7 4 3 3 -تقسيم البيانات في IP

تؤمن عملية التقسيم في IP القدرة على إرسال حزم البيانات عبر شبكات مختلفة و التي تملك قيود على حوم حزم البيانات. بما أن بيانات حزم IP تصل من برتوكولات الطبقة العليا ULP فان حجم الحزم يكون متغير. مختلف أوساط النقل تملك تقييد على حجم البيانات المرسلة لذلك حتى يتمكن IP من إيصال هذه البيانات عبر هذه الأوساط المختلفة فعليه تقسيم هذه الحزم إلى أجزاء اصغر بحيث تتلاءم مع وسط النقل في الطبقة الثانية (Layer 2). الطبقة الثانية تنقل بيانات الطبقة الثالثة باستخدام الأطر frames حيث أن كل برتوكول في الطبقة الثانية (Ethernet, X25, Frame Relay) يملك حجم معين للأطر التي يقوم بإرسالها. تسمى هذه القيمة MTU (Maximum Transmission Unit) و التي يعبر عنها بالبايت.

يتم استخدام الحقول Identification, Flags, Fragment Offset للمساعدة في عملية التقسيم. عندما يتم استخدم التقسيم فان رقم الحزمة المحتوى ضمن الحقل Identification يعرف جميع حزم IP المجزئة عن البيانات الأصلية. الحزمة الأولى التي تحوي على البيانات المجزئة ترسل مع وضع قيمة "More Fragment" ضمن الحقل Flags. هذا يخبر الجهة المستقبلة انه يجب عليها تخزين cache الحزم التي تملك نفس الرقم Identification وانه هنالك المزيد من الحزم قادمة ليتم تجميعها مع بعضها وهنا يستخدم أيضا الحقل Fragment Offset ليعرف موقع الحزمة المجزئة ضمن البيانات الرئيسية، وترسل آخر جزء من الحزمة بقيمة "More Fragment off" ضمن الحقل Flags، وعند وصول جميع الحزم إلى الجهة المستقبلة يتم إعادة تجمعها و إيصالها إلى برتوكولات الطبقة العليا ULP.

إجرائية التقسيم:

إذا كان الحجم الكلي اصغر أو يساوي MTU يتم إرسال الحزمة إلى خطوة المعالجة التالية. أما إذا كانت اكبر من MTU يتم تقسيم الحزمة إلى جزئين حيث يكون حجم الحزمة الأولى يساوي MTU و الثانية تكون بقية الحجم، ثم يتم إرسال الحزمة الأولى إلى خطوة المعالجة التالية بينما يتم إدخال الحزمة الثانية إلى إجرائية التقسيم مرة أخرى إذا كانت مازال اكبر من MTU.

دلالات الرموز:

- FO: Fragment Offset.
- IHL: Internet Header Length.
- DF: Don't Fragment flag.
- MF: More Fragments flag.

- Total Length. :TL
- Old Fragment Offset. :OFO
- Old Internet Header Length. :OIHL
- Old More Fragments flag. :OMF
- Old Total Length. :OTL
- Number of Fragment Blocks. :NFB
- Maximum Transmission Unit. :MTU

الإجرائية:

تكون الخوارزمية على النحو التالي:

```
إذا كان MTU > = TL قم بإرسال الحزمة إلى خطوة المعالجة التالية
وإلا إذا كان DF=1 قم بإلغاء عملية التقسيم
وإلا قسّم كالتالي (الخطوة الأولى):
انسخ IP Header الأصلي.
OIHL ← IHL; OTL ← TL; OFO ←FO; OMF ←MF
NFB ← (MTU −IHL*4)/8
قم بربط البيانات بحجم NFB*8
صحح IP Header
MF ← 1; TL ← (IHL*4) + (NFB*8)
قم بإعادة حساب Checksum
قم بإرسال هذا الجزء إلى خطوة المعالجة التالية
```

(الخطوة الثانية):

```
إضافة البيانات المتبقية
تصحيح IP Header:
IHL ← (((OIHL*4)-(length of options not copied)) +3)/4;
TL ← OTL - NFB*8 - (OIHL-IHL)*4;
FO ← OFO + NFB; MF ← OMF;
```

إرسال هذا الجزء إلى الخطوة الأولى.

إجرائية التجميع :

من اجل كل حزمة مستقبلة يتم حساب معرف ذاكرة التجميع لها باستخدام الحقول التالية source, destination, protocol, identification أذا كانت هذه الحزم كاملة (أي أنها ليست جزء لحزمة أخرى) فانه تيم تحرير ذاكرة التجميع لهذه الحزمة وإرسالها إلى خطوة المعالجة التالية.

إذا لم يكن هنالك ذاكرة محجوزة من اجل أول جزء مستقبل نقوم بحجزها، تتألف هذه الذاكرة من ذواكر فرعية هي data buffer, header buffer, fragment block bit table, total data length field, timer حيث يتم وضع البيانات التي نحصل عليها من الجزء

المستقبل في Data Buffer بحسب رقم الإزاحة fragment offset و الطول Length والبتات الموجودة ضمن جدول بت الكتلة المجزئة fragment block bit الموافقة للكتل المجزئة المستقبلة.

إذا كان هذا الجزء الأول من الحزمة نقوم بنسخ الـHeader إلى Header buffer أو إذا كان هذا الجزء الأخير يتم حساب حجم البيانات الكلي. ثم ترسل الحزمة المجمّعة إلى خطوة المعالجة التالية. وإلا يتم تشغيل المؤقت و إنهاء عملية التجميع.إذا انتهى الوقت يتم تحرير جميع الموارد المحجوزة لتجميع هذه الحزمة.

دلالات الرموز:

- **FO: Fragment Offset.**
- **IHL: Internet Header Length.**
- **MF: More Fragments flag.**
- **TTL: Time to Live.**
- **NFB: Number of Fragment Blocks.**
- **TL: Total Length.**
- **TDL: Total Data Length.**
- **BUFID: Buffer Identifier.**
- **RCVBT: Fragment Received Bit Table.**
- **TLB: Timer Lower Bound.**

الإجرائية:

تكون الخوارزمية على النحو التالي:

BUFID ← source|destination|protocol|identification;

إذا كان FO = 0 و MF = 0
وإذا كانت ذاكرة BUFID محجوزة قم بإرسال الحزمة إلى خطوة المعالجة التالية.
وإلا إذا لم يكن هنالك ذاكرة محجوزة لـ BUFID قم بحجزها من اجل BUFID.
TIMER ← TLB; TDL ← 0
ضع البيانات المأخوذة من الأجزاء ضمن ذاكرة BUFID من FO*8 octet إلى (TL- octet (IHL*4))+FO*8.
انقل RCVBT من FO إلى FO+((TL-(IHL*4)+7)/8)
إذا MF = 0 قم بوضع TDL ← TL-(IHL*4)+(FO*8)
إذا FO = 0 قم بوضع الـHeader ضمن الذاكرة.
إذا IDL <> 0 و تم نقل جميع RCVBT من 0 إلى (TDL+7)/8
TL ← TDL+(IHL*4)
قم بإرسال الحزمة المجمعة إلى خطوة المعالجة التالية.
قم بتحرير ذاكرة BUFID.
TIMER ← MAX (TIMER, TTL)
قم بإنهاء عملية التجميع عند وصول آخر حزمة أو عند انتهاء الوقت.
انتهى الوقت (timeout) قم بار سال الحزم المجمّعة إلى خطوة المعالجة التالية.

7 4 3 4 -عناوين IP

يتم التمييز بين الأشياء من خلال الأسماء، أو العناوين حيث يشير الاسم على ماذا نريد البحث و العنوان يشير إلى أين نجده. يتعامل IP بشكل رئيسي مع العناوين حيث أن مهمة تطبيقات الطبقة العليا لترجمة الأسماء إلى عناوين.إن عناوين IP هي ثابتة الحجم تتألف من أربعة ثمانيات (32 بت) حيث يبدأ العنوان برقم الشبكة متبوع بالعنوان المحلي. هنالك ثلاثة أصناف لعناوين IP صنف A، صنف B، صنف Cحسب الجدول التالي:

البت الأكثر أهمية	التنسيق	الصنف
0	البتات7 الأولى لرقم الشبكة و الباقي 24 للحواسيب	A
10	البطات 14 الأولى لرقم الشبكة و الباقي 16 للحواسيب	B
110	البطات 21 الأولى لرقم الشبكة و الباقي 8 للحواسيب	C

7 4 3 5 -طريقة إرسال حزم البيانات في IP

عندما يقوم أي حاسب بإرسال بيانات إلى حاسب آخر فان تتم في البداية عملية التوجيه routing لتحديد فيما إذا كانت الوجهة تقع ضمن الشبكة المحلية أو إذا كانت خارج الشبكة المحلية ومن ثم تتم عملية الإرسال الفعلية للبيانات.

هنالك نموذجين لتسليم البيانات:

التسليم ضمن الشبكة المحلية ويسمى Local Delivery.
التسليم إلى وجهة بعيدة خارج الشبكة المحلية Remote Delivery.

يتم في التسليم المحلي إرسال البيانات من حاسب إلى آخر ضمن الشبكة المحلية أي أن عنواني IP المصدر و الوجهة يملكان نفس قناع الشبكة. أما التسليم عن بعد يحدث عند إرسال بيانات إلى عنوان IP لا يملك نفس قناع عنوان IP المصدر حيث يتم استخدام Gateway افتراضية لإتمام عملية التوجيه.هنا تظهر الحاجة إلى جدول التوجيه routing table والذي يحوي على أربع حقول أساسية:

الواجهة Interface التي ستخدم في عملية الإرسال.
عنوان الشبكة أي الجزء الخاص بالشبكة في عنوان IP.
قناع الشبكة لتمييز عناوين IP هل تقع ضمن الشبكة أو لا.
Gateway و التي تعبر عن وجهة البيانات أذا كانت سترسل إلى وجهة بعيدة.

عند إيجاد الوجهة المناسبة للبيانات تبدأ عملية الإرسال. فإذا كانت الوجهة ضمن الشبكة المحلية تتم عملية الإرسال كالتالي:

تنتقل حزمة البيانات من الطبقة الثالثة (طبقة الشبكة) إلى الطبقة الثانية.
يتم تغليف الحزمة ضمن تنسيق طرد الطبقة الثانية.
يتمم إرسال الحزمة إلى الوجهة مباشرة باستخدام وسيط النقل الفيزيائي.
أما إذا كانت الوجهة بعيدة فتتم عملية الإرسال على النحو التالي:
تنتقل حزمة البيانات من الطبقة الثالثة (طبقة الشبكة) إلى الطبقة الثانية.

يتم تغليف الحزمة ضمن تنسيق طرد الطبقة الثانية.
يتم إرسال الحزمة مباشرة إلى Gateway الافتراضية.

7 4 4 -بروتوكول ICMP

هو اختصار لـ Internet Control Message Protocol و يشكل خدمة رسائل الأخطاء في IP. حيث يستخدم لإرسال تقارير أخطاء إرسال حزم البيانات في كل طرفي المرسل و المستقبل. إحدى المشاكل هي أن الجهة المستقبلة غير متوفرة آو أن الشبكة معطلة. فإذا مثلا حاولت الاتصال بخدمة FTP وتم إخبارك بأن الخدمة غير متوفرة " Service was unavailable" أو أن لم يتم العثور على المخدم "Server not Found" فأنت قد استقبلت رسالة ICMP.

بما أن البرتوكول IP يعتمد على النقل من خلال اتصال غير موجه Connectionless transport فإن المستقبل لا يعرف فيما إذا كانت الجهة المستقبلة أو الخدمة متوفرة أم لا عند إرسال حزم بيانات IP لذلك نحتاج إلى البرتوكول ICMP للحصول على تقارير بالأخطاء عند حدوثها.

أحد أهم الأشياء التي نستخدم فيها ICMP هي معرفة مدى توفر مخدم معين عن طريق Packet Internet Groper Application (PING) حيث تمكن من معرفة مدى القدرة على الوصول إلى الجهة المستقبلة كما تخبرك PING بإحصائيات عن البيانات التي أرسلت وإذا تم فقدانها و ما هو الزمن الذي استغرقته لوصولها. يعمل ICMP كخدمة في الطبقة الرابعة ويتم استخدام برتوكول IP لإرسال حزم ICMP.

1 4 4 7 -تنسيق رسائل ICMP

كما ذكرنا يتم إرسال رسائل ICMP باستخدام برتوكول IP حيث يتم استخدام ثماني بتات الأولى في قسم البيانات للحقل ICMP type حيث تحدد قيمة هذا الحقل تنسيق البيانات كما إن أي حقل مسمى unused محجوز للاستخدامات المستقبلية و يجب أن تكون قيمتها صفرية عند الإرسال لكن على المستقبل ألا يستخدم هذه الحقول ما عدا انه يجب تضمنيه في Checksum أما قيم حقول IP تكون كالتالي:

Version = 4

IHL = Internet header length in 32-bit words

Type of Service = 0

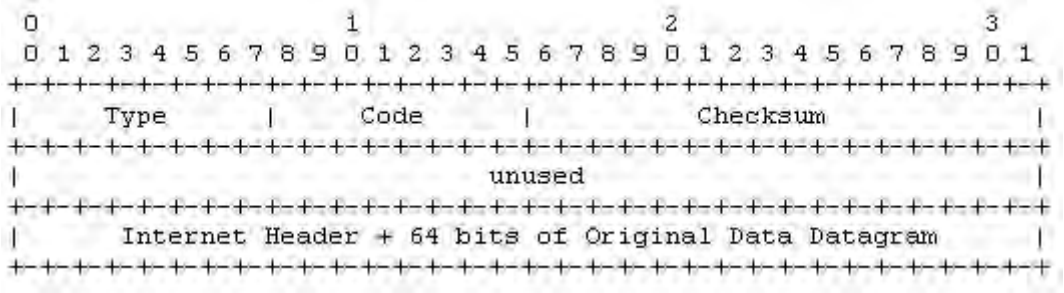
Total Length = Length of internet header and data in octets

Identification, Flags, Fragment Offset.هي حقول غير مستخدمة

Time to Live يتم زيادة قيمة هذا الحقل عند مرور حزم البيانات ضمن كل حاسب وتكون قيمة هذا الحقل في النهاية عدد البوابات gateways التي مرت خلالها الحزمة.

Protocol وتكون قيمته ICMP = 1
Header Checksum: تحوي على قيمة checksum ويجب أن يتم تهيئـه هذا الحقل بقيمة صفرية.
أنواع رسائل ICMP:

رسالة عدم الوصول للوجهة Destination Unreachable Message:



الشكل 69

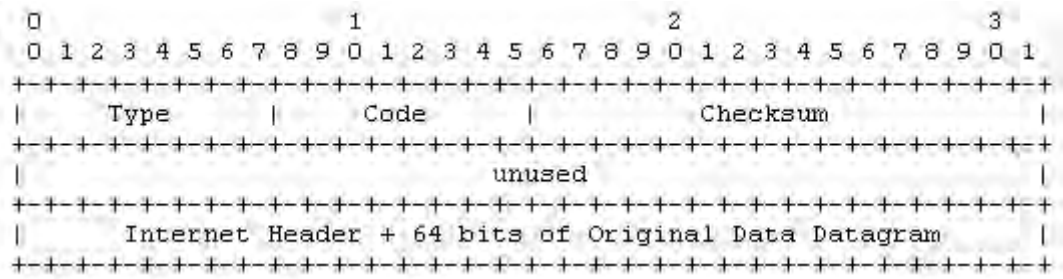
حقول IP:

Destination Address: يمثل عنوان IP الجهاز المستقبل والمأخوذ من حزمة البيانات الأصلية

حقول ICMP:

- **Type = 3**
- **Code**:
- **0 = net unreachable**
- **1 = host unreachable**
- **2 = protocol unreachable**
- **3 = port unreachable**
- **4 = fragmentation needed and DF set**
- **5 = source route failed**
- **Checksum**
- **Internet Header + 64 bits of Data Datagram**

2 4 4 7 -رسالة انتهاء الوقت Time Exceeded Message


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Code
Checksum
unused
Internet Header + 64 bits of Original Data Datagram

70 الشكل

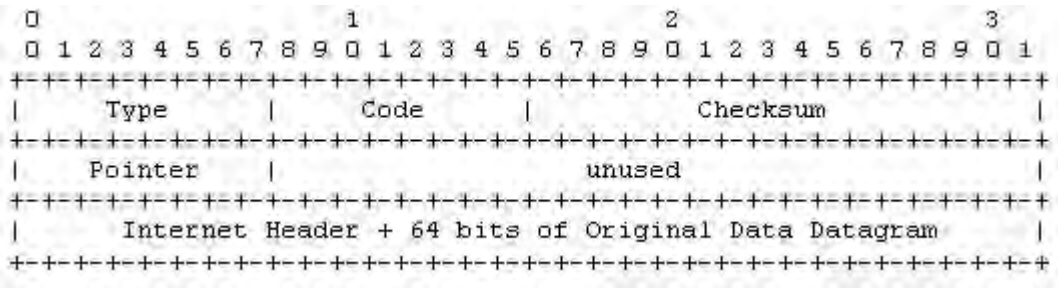
حقول IP:

Destination Address: يمثل عنوان IP الجهاز المستقبل والمأخوذ من حزمة البيانات الأصلية

حقول ICMP:

Type = 11
Code:
0 = time to live exceeded in transit
1 = fragment reassembly time exceeded
Checksum
Internet Header + 64 bits of Data Datagram

3 4 4 7 -رسالة مشكلة المتحول Parameter Problem Message:



الشكل 71

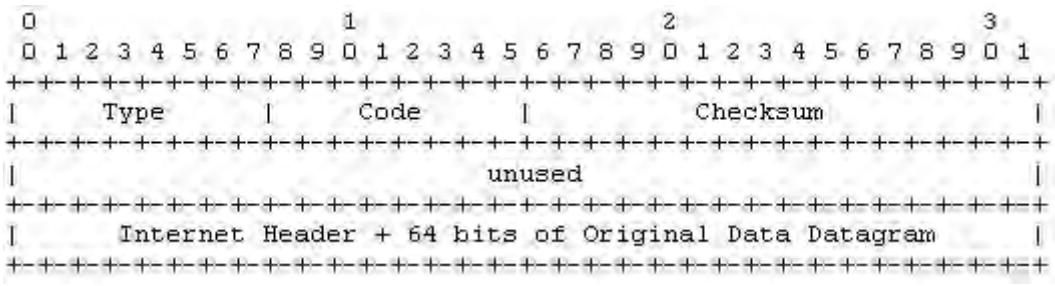
حقول IP:

Destination Address: يمثل عنوان IP الجهاز المستقبل والمأخوذ من حزمة البيانات الأصلية

حقول ICMP:

Type = 12
Code:
0 = pointer indicates the error
Checksum
Internet Header + 64 bits of Data Datagram

Source Quench Message- 4 4 4 7



الشكل 72

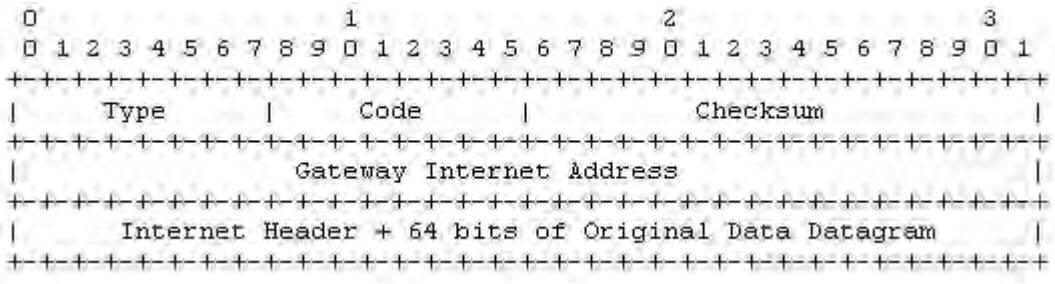
حقول IP:

Destination Address: يمثل عنوان IP الجهاز المستقبل والمأخوذ من حزمة البيانات الأصلية

حقول ICMP:

Type = 4
Code = 0
Checksum
Internet Header + 64 bits of Data Datagram

Redirect Message إعادة التوجيه- رسالة 5 4 4 7



الشكل 73

حقول IP:

Destination Address: يمثل عنوان IP الجهاز المستقبل والمأخوذ من حزمة البيانات الأصلية

حقول ICMP:

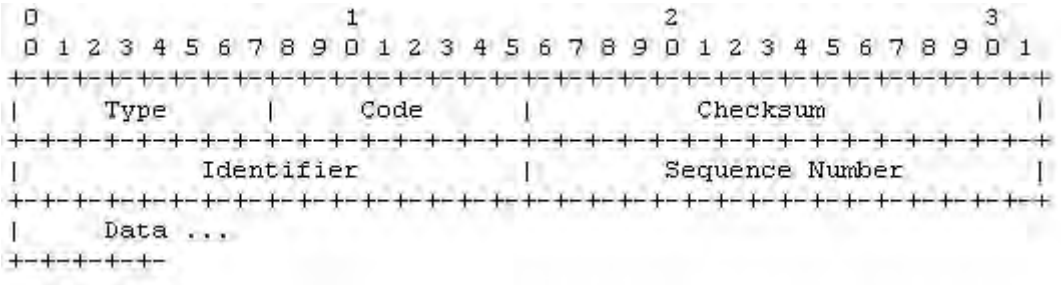
Type = 5

:Code
0 = Redirect datagrams for the Network
1 = Redirect datagrams for the Host
2 = Redirect datagrams for the Type of Service and Network
3 = Redirect datagrams for the Type of Service and Host
Checksum
Gateway Internet Address: عنوان البوابة gateway التي يتم إرسال حزم البيانات إليها.
Internet Header + 64 bits of Data Datagram

6 4 4 7 -رسالة طباعة الجواب Echo or Echo Reply Message

حقول IP:

Addresses: عنوان المصدر في رسالة الصدى echo message سيكون عنوان الوجهة لرسالة الرد echo reply message. ولتشكيل رسالة echo reply message يتم تبادل عناوين المصدر والهدف ويتم تغيير Code إلى 0 ويتم إعادة حساب checksum.

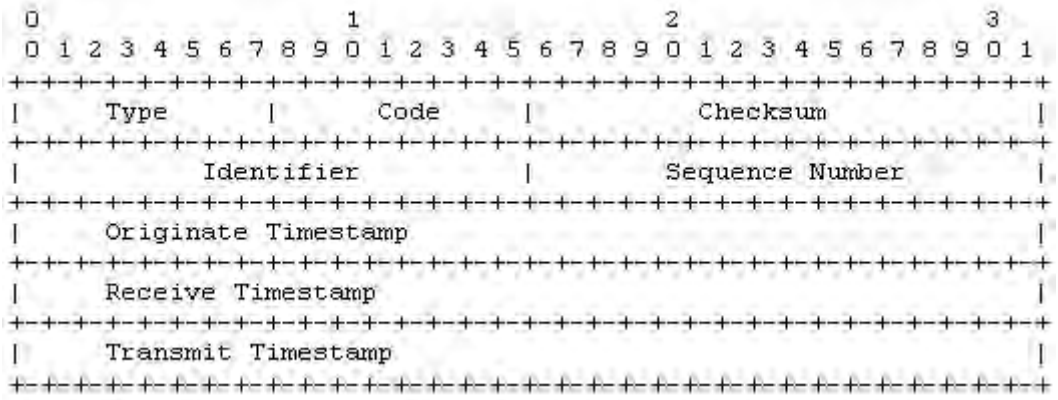


الشكل 74

حقول ICMP:

:Type
8 for echo message
0 for echo reply message
Code = 0
Checksum
Identifier: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل من رسالة الصدى و الرد.
Sequence Number: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل من رسالة الصدى و الرد.

Timestamp or Timestamp Reply Message- 7 4 4 7



الشكل 75

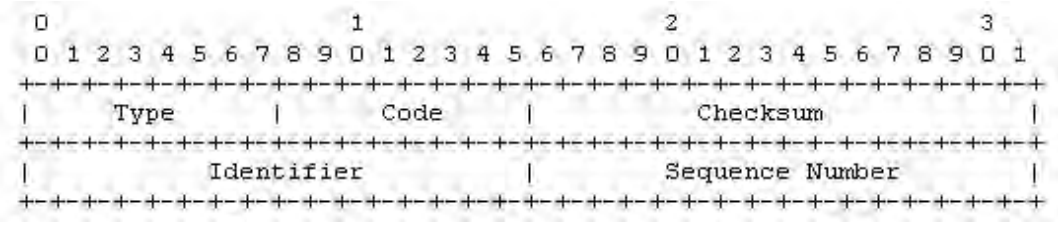
حقول IP:

Addresses: عنوان المصدر في رسالة الوقت timestamp سيكون عنوان الوجهة لرسالة الرد timestamp reply message. ولتشكيل رسالة timestamp reply message يتم تبادل عناوين المصدر والهدف ويتم تغيير Code إلى 14 ويتم إعادة حساب checksum.

حقول ICMP:

Type:
13 for timestamp message
14 for timestamp reply message
Code = 0
Checksum
Identifier: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل من الوقت والرد.
Sequence Number: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل الوقت والرد.

7 4 4 8 -رسالة معلومات الإجابة
Information Request or Information Reply Message



الشكل76

حقول IP:

Addresses: عنوان المصدر في رسالة طلب المعلومات information request message سيكون عنوان الوجهة لرسالة الرد information reply message . ولتشكيل رسالة رد للمعلومات يتم تبادل عناوين المصدر والهدف ويتم تغيير Code إلى 16 ويتم إعادة حساب checksum.

حقول ICMP:

Type:
15 for information request message
16 for information reply message
Code = 0
Checksum
Identifier: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل من الطلب والرد.
Sequence Number: إذا كان code = 0 يساعد الحقل Identifier على مطابقة كل الطلب والرد.

7 4 4 9 -خلاصة أنواع رسائل ICMP

0 Echo Reply
3Destination Unreachable
4Source Quench
5Redirect
8Echo
11Time Exceeded
12Parameter Problem
13Timestamp
14Timestamp Reply
15Information Request

16Information Reply

7 4 5 -Ethernet Address Resolution Protocol

عندما نستخدم برتوكول ما P لإرسال معلومات من جهاز معين بطريقة معينة إلى جهاز آخر T موجود في مكان ما على الشبكة باستخدام تقنية Ethernet فيجب توليد عنوان 48bit لإرسال حزم بيانات Ethernet. عنوان الجهاز S ضمن البرتوكول P ليس دائماً متوافق مع تنسيق عنوان Ethernet (والذي يكون بطول مختلف و قيمة مختلفة). البرتوكول P يسمح بتوزيع ديناميكي للمعلومات التي نحتاجها لبناء جداول التي نستخدمها لترجمة عنوان ما A من صيغة عنوان برتوكول P إلى صيغة عنوان 48bit Ethernet.بمعنى آخر يستخدم هذا البرتوكول Ethernet ARP لتحويل عناوين برتوكول الشبكة إلى عنوان Ethernet 48bit من اجل النقل باستخدام تقنية الـEthernet.عمليا هنالك وظيفتين للبرتوكول ARP:

إنشاء خريطة Map لعناوين IP و العناوين الفيزيائية لإرسال حزم البيانات.

الإجابة على طلبات الأجهزة الأخرى على الشبكة للحصول على العنوان الفيزيائي للجهاز المحلي.

7 4 5 1- تعار يف الحقل TYPE

فيما يلي تعريف لبعض القيم التي يأخدها الحقل TYPE ضمن حزمة بيانات Ethernet:

ether_type$XEROX_PUP
ether_type$DOD_INTERNET
ether_type$CHAOS
ether_type$ADDRESS_RESOLUTION
تنسيق حزمة Ethernet ARP:

لإنشاء خريطة لترجمة زوج <protocol, address> إلى عنوان Ethernet 48bit نحتاج إلى إنشاء حزمة ARP، تنسيق آو صيغة هذه الحزمة تكون كالتالي:

```
48.bit: Ethernet address of destination
48.bit: Ethernet address of sender
16 bit Protocol type = ether_type$ADDRESS_RESOLUTION
16.bit: Hardware address space (ar$hrd)
16.bit: Protocol address space (ar$pro)
bit: byte length of each hardware address (ar$hln)
bit: byte length of each protocol address (ar$pln)
16.bit: opcode (ares_op$REQUEST | ares_op$REPLY) (ar$op)
nbytes: Hardware address of sender of this packet (ar$sha)
mbytes: Protocol address of sender of this packet (ar$spa)
nbytes: Hardware address of target of this packet (ar$tha)
mbytes: Protocol address of target (ar$tpa)
```

7 4 5 2 -توليد الحزمة

عندما يتم إرسال حزمة البيانات عبر طبقات الشبكة يتم تحديد عنوان البرتوكول للمحطة التالية التي سيتم إرسال الحزمة لها. في حالة Ethernet نحتاج إلى ترجمة العنوان لتحويل زوج <protocol type, target protocol address> إلى عنوان Ethernet 48bit حيث يحاول ARP إيجاد هذا الزوج ضمن الجدول فإذا وجده فانه يقوم بإعادة عنوان Ethernet 48bit الموافق، وإذا لم يجده فانه يقوم بحذف حزمة البيانات (بافتراض أن الطبقات العليا ستعيد إرسالها) و يقوم بتوليد حزمة Ethernet مع توليد قيم للحقول كالتالي:

```
typ = ether_type$ADDRESS_RESOLUTION
ar$hrd = ares_hrd$Ethernet
ar$pro = protocol type
ar$hln = 6
ar$pln = length of an address in that protocol
ar$op = ares_op$REQUEST
ar$sha = 48.bit ethernet address of local host
ar$spa = protocol address of local host
ar$tpa = protocol address of the remote host
```

لا يقوم بإسناد أي شيء لـ ar$tha لأنه يجب ملىء هذا الحقل بالقيمة التي أحاول معرفتها ومن الممكن إسناد قيمة عنوان التعميم broadcast address إلى ar$tha لإرسال الطلب غالى كل محطات العمل التي تعمل بتقنية Ethernet.

7 4 5 3 -إجابة الطلب

عندما يتم استقبال حزمة ARP تقوم الجهة المستقبلة بارسال الحزمة إلى ARP التي تستخدم خوارزمية الشروط السلبية و إنهاء معالجة الحزمة كالتالي:

```
هل املك نوع العتاد hardware type المذكور ضمن الحقل ar$hrd؟
نعم:
[فحص الحقل ar$hln يكون اختياريا]
هل أتعامل مع البرتوكول المذكور في ar$pro؟
نعم:
[فحص الحقل ar$pln يكون اختياريا]
Merage_flage: = false
إذا كان الزوج <protocol type, sender protocol address> موجود ضمن
جدول الترجمة قم بتحديث الحقل address hardware sender بالمعلومات الجديدة وإسناد القيمة true للـ
Merage_flag
هل أنا target protocol address ؟
نعم:
إذا كان Merge flag = false قم بإضافة الثلاثة < protocol type,
<sender protocol address, sender hardware address إلى جدول الترجمة.
هل opcode = ares_op$REQUEST؟
نعم:
قم بتبديل الحقلين Hardware و Protocol ضع عناوين Hardware و
Protocol المحلية ضمن حقول Sender
وضع Ar$op = ares_op$REPLY
إرسال الحزمة إلى نفس العنوان الذي استقبل منه الطلب.
```

لا حظ أن

الثلاثية < protocol type, sender protocol address, senderhardware address >

قد حفظت في الجدول قبل فحص قيمة الحقل opcode. لأننا افترضنا إن الاتصال هو بالاتجاهين بمعنى إذا كان لـA سبب للحديث مع B إذا B يملك أيضا سبب للحديث مع B ولاحظ انه إذا وجد الزوج <protocol type, sender protocol address> فان hardware address الجديد يحل محل القديم.

4 5 4 7 -مثال

بفرض أن لدينا جهازين X,Y موجودين على نفس الشبكة والتي هي من نوع Ethernet. كل الجهازين يملكان عنوان Ethernet EA(Y) , EA(X) وكلاهما يمتلكان عنوان IP IPA(X), IPA(Y) وبافتراض أن Ethernet type هو ET(IP). الجهاز X بدء بالعمل للتو و عاجلاً أو آجلا سيقوم الجهاز X بإرسال حزم بيانات إلى الجهاز Y. الجهاز X يعلم أن سيرسل إلى IPA(Y) ويطلب من ARP تحويل الزوج <ET(IP), IPA(Y)> إلى عنوان Ethernet 48bit ولكن بما أن X بدء العمل للتو فانه لا يعرف هذه المعلومات. فيقوم بحذف الحزمة وإنشاء حزمة ADDRESS RESOLUTION بالمعلومات التالية:

```
(ar$hrd) = ares_hrd$Ethernet
)ar$pro) = ET(IP(
)ar$hln) = length(EA(X((
)ar$pln) = length(IPA(X((
(ar$op)  = ares_op$REQUEST
)ar$sha) = EA(X(
)ar$spa) = IPA(X(
(ar$tha) = don't care
)ar$tpa) = IPA(Y(
```

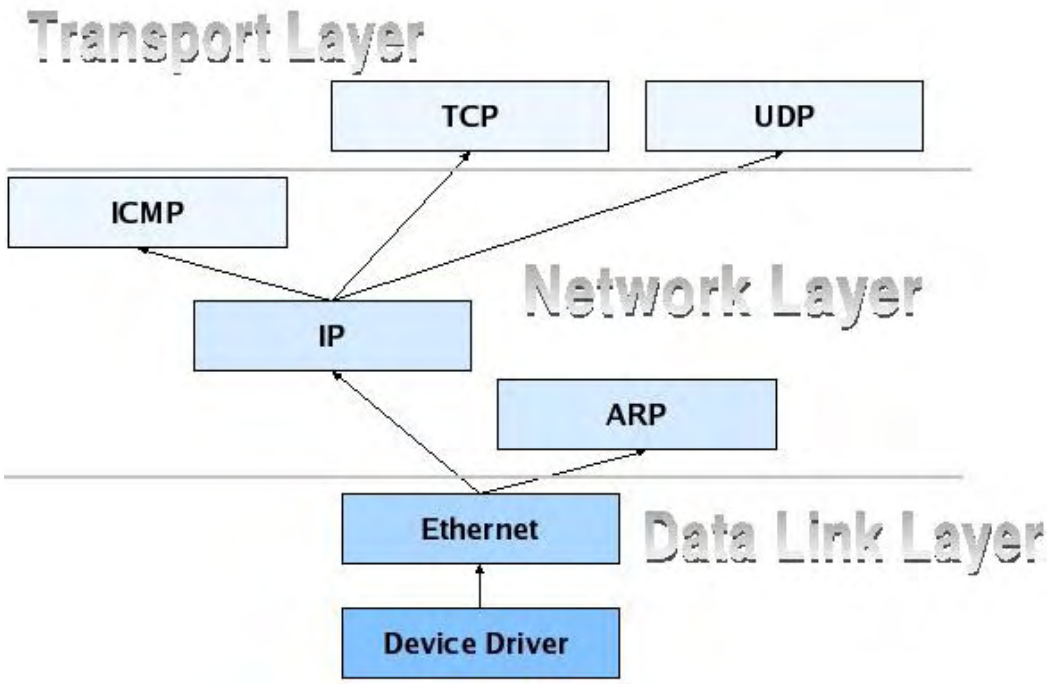
ويتم تعميم الحزمة على جميع الأجهزة في الشبكة. يتلقى الجهاز Y هذه الحزمة ويحدد فيما إذا كان يستطيع فهم الحقل hardware type (Ethernet) وان هذه الحزمة مرسلة له ((ar$tpa)=IPA(Y)) ويقوم بإدخال المعلومات <ET(IP), IPA(X)> للخريطة بعدها يلاحظ أن هذا هو طلب request لذا يقوم بتبديل الحقول ويضع EA(Y) ضمن الحقل (ar$sha) ويسند القيمة reply للحقل opcode ويسر هذه الحزمة مباشرة إلى EA(X). عند هذه اللحظة Y يعلم كيف يرسل إلى X لكن X مازال لا يعرف كيف يرسل إلى Y.

يقوم X باستقبال حزمة الرد من Y و يدخل المعلومات إلى الجدول <ET(IP), IPA(Y)> و EA(Y). في المرة القادمة عندما يحاول برتوكول الإرسال في X إرسال حزمة بيانات إلى Y فان عملية ترجمة العنوان سوف تنجح كما أن برتوكول الإرسال في Y سوف ينجح في عملية ترجمة العنوان عندما يريد إرسال معلومات إلى Y.

7 5 -قسم التنفيذ Implementation

7 5 1 -مقدمة

بعد هذه المقدمة النظرية التي استعرضنا فيها أهمية الشبكة في نظم التشغيل كما استعرضنا بنية بطاقة الشبكة. و تحدثنا عن النموذج المعياري OSI الذي تعتمده جميع نظم التشغيل الشبكية كما تحدثنا عن مجموعة برتوكولات TCP/IP بشيء من التفصيل.الآن كيف سنستفيد من المعلومات النظرية السابقة في التنفيذ، ربما الشكل التالي سيعطيك فكرة عامة عن طريقة التنفيذ.



نستنتج من الشكل الخطوات التالية للتنفيذ:

كتابة Device Driver: ويقسم إلى قسمين:
PCI Device Driver
Ethernet Relate RTL8139 Device Driver
تنفيذ برتوكول ARP
تنفيذ برتوكول IP
تنفيذ برتوكول ICMP
تنفيذ برتوكولي التحكم بالنقل TCP & UDP.

7 5 2 -بنية الملفات في قسم تنفيذ الشبكة

اسم الملف	مهمة الملف
pick	عبارة عن مجموعة التوابع اللازمة لتنفيذ مشغل PCI.
rtl8139.c	مشغل الشبكة Network Device Driver ويعمل على الشريحة RTL8139
native's	عبارة عن مكتبة تغلف بطاقة الشبكة و تتملك عدد من التوابع لإدارة بطاقات الشبكة من إضافة، وحذف،...الخ
eatharp.c	يحوي هذا الملف عملية التنفيذ implementation للبرتوكول Ether ARP.
inlet's	مكتبة تحوي على عدد من التوابع للتحقق من صحة عناوين IP المدخلة كنص و تحويلها إلى الشكل الثنائي.
app.	يحوي هذا الملف علة عملية التنفيذ implementation للبرتوكول IP.
ip_addr.c	مكتبة تحوي على عدد من التوابع للتعامل مع عناوين IP من تشكيل، مقارنة، وطباعة....الخ.
ip_frag.c	يحوي هذا الملف على توابع التجزيء و التجميع في برتوكول IP.
icmp.c	يحوي هذا الملف على عملية تنفيذ implementation للبرتوكول ICMP.
tcp.c	يحوي هذا الملف على عملية تنفيذ implementation للبرتوكول TCP.
tcp_in.c	مجموع توابع الاستقبال في برتوكول TCP.
tcp_out.c	مجموعة توابع الإرسال في TCP.
dupl.	يحوي هذا الملف على عملية تنفيذ implementation للبرتوكول UDP.
pouf's	مكتبة تحوي على بنية معطيات الحزم Packet و اجرائيات التعامل معها.
mimic	مكتبة تحوي على عدد من الاجرائيات لإدارة الذاكرة ضمن TCP/IP Stack
hemp's	مكتبة تحوي على عدد من الاجرائيات لإدارة ذاكرة الحزم Packets.

7 5 3 -تنفيذ Network Device Driver

7 5 3 1- PCI Device Driver

لانجاز هذا المشغل استخدمنا بنية المعطيات التالية التي تعبر عن أي بطاقة PCI.

```
typeset struct pci_cfg {
        uint16_t vendor_id;
        uint16_t device_id;
        uint16_t command;
        uint16_t status;
        uint8_t revision_id;
        uint8_t interface;
        uint8_t sub_class;
        uint8_t base_class;
```

```
        uint8_t cache_line_size;
        uint8_t latency_timer;
        uint8_t header_type;
        uint8_t bist;
        uint8_t bus;
        uint8_t dev;
        uint8_t func;
        uint8_t irq;
        uint32_t base[6];
        uint32_t size[6];
        uint8_t type[6];
        uint32_t rom_base;
        uint32_t rom_size;
        uint16_t subsys_vendor;
        uint16_t subsys_device;
        uint8_t current_state;
} pci_cfg_t;
```

كما استخدمنا البنية التالية لتمييز بين أصناف PCI:

```
classes_t classes[] = {

{ 0x00, 0x00, 0x00, "Undefined" },
{ 0x00, 0x01, 0x00, "VGA" },

{ 0x01, 0x00, 0x00, "SCSI" },
{ 0x01, 0x01, 0x00, "IDE" },
{ 0x01, 0x01, 0x8A, },
{ 0x01, 0x02, 0x00, "Floppy" },
{ 0x01, 0x03, 0x00, "IPI" },
{ 0x01, 0x04, 0x00, "RAID" },
{ 0x01, 0x05, 0x20, "ATA (Single DMA)" },
{ 0x01, 0x05, 0x30, "ATA (Chained DMA)" },
{ 0x01, 0x80, 0x00, "Other" },

{ 0x02, 0x00, 0x00, "Ethernet" },
{ 0x02, 0x01, 0x00, "Token Ring" },
{ 0x02, 0x02, 0x00, "FDDI" },
{ 0x02, 0x03, 0x00, "ATM" },
{ 0x02, 0x04, 0x00, "ISDN" },
{ 0x02, 0x05, 0x00, "WordFip" },
{ 0x02, 0x06, 0x00, "PICMG 2.14" },
{ 0x02, 0x80, 0x00, "Other" },
....
....
....
```

إن أهم تابع ضمن المشغل هو:

pci_find_cfg: الذي يقوم بالبحث عن بطاقة معينة في ممرات PCI وذلك بتمرير رقم الصنف الذي نريد البحث عنه وفي حال وجودها يقوم بتفعيلها. كما يقوم بإرجاع بنية معطيات تحوي على مجموعة من البيانات التي سنستخدمها للتعامل مع الجهاز مثل رقم المقاطعة و عنوان الجهاز.....الخ.

Ethernet Relate RTL8139 Device Driver- 2 3 5 7

لانجاز أي مشغل لبطاقة الشبكة يجب تحديد الشريحة التي سنعمل عليها لذا قمنا باعتماد الشريحة RTL8139. أول خطوة قمنا بها هي إنشاء بنية معطيات تمثل هذه الشريحة كما قمنا بإنشاء أنماط تعدادية و التي تمثل خيارات هذه الشريحة.أهم المعلومات التي نريدها معرفتها عن الشريحة هي Mac Address وعناوين كل من حلقة الإرسال و الاستقبال.

```
typedef struct rtl8139
{
        struct eth_addr *ethaddr;
        uint16_t iobase;
        uint8_t irq;
        uint8_t station_address[6];
        bool speed10;
        bool fullduplex;
        bool promisc;
        unsigned cur_tx;
        unsigned cur_rx;
        addr_t rx_phys, tx_phys;
        uint8_t         *rx_ring, *tx_ring;
        semaphore_t mutex;
} rtl8139_t;
```

أهم التوابع التي تشكل المشغل هي:

rtl8139_init: يقوم هذا التابع بتهيئة الشريحة لتصبح جاهزة لعمليات الارسال و الاستقبال و إسناد برامج خدمة المقاطعة للإرسال و الاستقبال.ففي البداية نقوم بالبحث عن البطاقة باستخدام التابع pci_find_cfg و بعد الحصول على المعلومات المطلوبة نقوم بإسنادها إلى البنية rtl8139_t لاستخدامها لاحقاً.

rtl8139_handler: هذا التابع هو برنامج خدمة المقاطعة الذي يتم استدعائه عن حدوث مقاطعة، يقوم هذا التابع بفحص نوع المقاطعة هل مقاطعة إرسال أم مقاطعة إرسال. ففي حال كانت مقاطعة إرسال يتم استدعاء التابع rtl8139_handle_tx أما إذا كانت مقاطعة استقبال فيتم استدعاء التابع rtl8139_handle_rx الذي يقوم بعملية الإرسال الفعلية.

rtl8139_close: يقوم هذا التابع بإغلاق بطاقة الشبكة و تعطيلها عن عمليات الإرسال و الاستقبال.

rtl8139_dump_info: يقوم هذا التابع بطباعة معلومات البطاقة على شاشة الإظهار.

get_eth_mac_addr: يقوم هذا التابع بإرجاع العنوان الفيزيائي Mac Address لبطاقة الشبكة.

low_level_output: يقوم هذا التابع بعملية الارسال الفعلية للبيانات أو الحزم Packets ويتم استدعاء هذا التابع من قبل برتوكول ARP عندما يريد إرسال البيانات.

Netif- 3 3 5 7

نعلم من خلال استخدامنا لأنظمة التشغيل الشبكية انه يمكن إسناد أكثر من عنوان IP لنفس الجهاز أو بشكل أدق إسناد أكثر من عنوان IP لنفس بطاقة الشبكة، كما يمكننا التعامل مع أكثر من بطاقة شبكة واحدة وإسناد عناوين مختلفة لكل منها. قمنا بتحقيق ذلك من خلال المكتبة Netif التي تحوي على بنية معطيات هي netif وهي عبارة عن لائحة مترابطة تمثل بطاقات الشبكة. من خلال البنية التي تظهر في الجدول نلاحظ انه علينا إسناد توابع الإدخال و الإخراج و التي مهمتها نقل البيانات من و إلى الطبقة العليا. حيث من اجل الحقل input نقوم بإسناد التابع ip_input الذي يقوم بنقل البيانات من بطاقة الشبكة إلى IP للمعالجة و منه إلى TCP أو UDP. و من اجل الحقل output نقوم بإسناد التابع etharp_output و الذي يقوم نقل البيانات من IP إلى بطاقة الشبكة و منها إلى العالم الخارجي.

```
struct netif {
  struct netif *next;

  struct ip_addr ip_addr;
  struct ip_addr netmask;
  struct ip_addr gw;

  err_t (* input)(struct pbuf *p, struct netif *inp);
  err_t (* output)(struct netif *netif, struct pbuf *p,
      struct ip_addr *ipaddr);
  err_t (* linkoutput)(struct netif *netif, struct pbuf *p);

  void *state;

  unsigned char hwaddr_len;
  unsigned char hwaddr[NETIF_MAX_HWADDR_LEN];
  u16_t mtu;
  u8_t flags;
  u8_t link_type;
  char name[2];
  u8_t num;
};
```

من أهم توابع هذه المكتبة:

netif_init: يقوم هذا التابع بتهيئة اللائحة المترابطة.
netif_add: يتم من خلال هذا التابع إضافة عنوان IP إلى بطاقة الشبكة ويمكن استدعاء هذا التابع أكثر من مرة لإضافة عناوين إضافية أو التعامل مع بطاقات شبكة إضافية.
netif_set_addr: يتم من خلال هذا التابع إسناد عنوان IP و قناع شبكة Network Mask و Gateway إلى بطاقة شبكة محددة.
netif_remove: حذف بطاقة شبكة محددة من اللائحة المترابطة.
netif_find: البحث عن بطاقة شبكة معينة ضمن اللائحة.
netif_set_default: جعل بطاقة شبكة معينة هي الافتراضية لاستخدامها في عمليات الارسال و الاستقبال.
netif_set_up: تفعيل بطاقة شبكة معينة.
netif_set_down: تعطيل بطاقة شبكة معينة.

netif_is_up: التأكد من أن بطاقة شبكة معينة هل هي مفعّلة أم لا.

7 5 4 -تنفيذ برتوكول ARP

نعلم أن مهمة ARP هي ترجمة عناوين IP إلى عناوين فيزيائية Mac Address للحزم قبل إرسالها لذا يتم استخدام مصفوفة من البنية التالية والتي تبقى في الذاكرة طالما أن بطاقة الشبكة مفعّلة:

```
struct etharp_entry {
  struct ip_addr ipaddr;
  struct eth_addr ethaddr;
  enum etharp_state state;
  u8_t ctime;
};

static struct etharp_entry arp_table[ARP_TABLE_SIZE];
```

تحوي الحزمة المرسلة من قبل ARP على العنوان الفيزيائي لكل من الوجهة و المصدر لذلك تم استخدام البنية eth_hdr و التي تم تضمينها في الحزم المرسلة.

```
struct eth_hdr {
  PACK_STRUCT_FIELD(u8_t padding[ETH_PAD_SIZE]);
  PACK_STRUCT_FIELD(struct eth_addr dest);
  PACK_STRUCT_FIELD(struct eth_addr src);
  PACK_STRUCT_FIELD(u16_t type);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

أما شكل الحزمة المرسلة أو المستقبلة فيتم تمثيلها بالبنية ethip_hdr وتتألف من جزئين هما IP Header و ARP Header.

```
PACK_STRUCT_BEGIN
struct ethip_hdr {
  PACK_STRUCT_FIELD(struct eth_hdr eth);
  PACK_STRUCT_FIELD(struct ip_hdr ip);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

أما بنية رسالة ARP من اجل طلب ترجمة عنوان فيتم تمثيلها باستخدام البنية etharp_hdr.

```
/** the ARP message */
struct etharp_hdr {
  PACK_STRUCT_FIELD(struct eth_hdr ethhdr);
  PACK_STRUCT_FIELD(u16_t hwtype);
  PACK_STRUCT_FIELD(u16_t proto);
  PACK_STRUCT_FIELD(u16_t _hwlen_protolen);
  PACK_STRUCT_FIELD(u16_t opcode);
  PACK_STRUCT_FIELD(struct eth_addr shwaddr);
  PACK_STRUCT_FIELD(struct ip_addr2 sipaddr);
  PACK_STRUCT_FIELD(struct eth_addr dhwaddr);
  PACK_STRUCT_FIELD(struct ip_addr2 dipaddr);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

أهم توابع المشكلة للبرتوكول ARPهي:

etharp_init: يقوم هذا التابع بتهيئة البرتوكول وإنشاء جدول الترجمة وتهيئته.
etharp_tmr: هذا التابع يتم تنفيذه بشكل متكرر وخلال فترة زمنية معرفة بالمتحول ETHARP_TMR_INTERVAL و يقوم بحذف المداخل المنتهية.
etharp_ip_input: يقوم هذا التابع بتحديث جدول ARP مستخدما عنوان IP المصدر للحزمة حيث يقوم بتحديث العنوان الفيزيائي Mac Address المقابل.
etharp_arp_input: يتم استدعاء هذا التابع لاجابة رسالة ARP حيث يقوم بإنشاء مدخل للعنوان IP المرسل ثم بعدها يقوم بالإجابة على الرسالة بإرسال عنواننا الفيزيائي المحلي.
etharp_output: يستدعى هذا التابع عند إرسال حزم البيانات حيث يقوم بإسناد العنوان الفيزيائي Mac Address للحزم الخارجة فإذا كان عنوان تعميم Broadcastأو UniCast يتم إرسال الحزمة مباشرة إلى بطاقة وإلا يقوم باستدعاء التابع etharp_query الذي يتولى عملية إرسال الحزم.
etharp_query: يقوم هذا التابع بالبحث ضمن جدول عناوين ARP عن عنوان IP معين فإذا لم يجده يقوم بإنشاء مدخل ip/mac ويقوم بإرسال رسالة طلب عنوان ARP للحصول على العنوان الفيزيائي للوجهة وإلا يقوم بإرسال الحزم إلى بطاقة الشبكة ومنها إلى العالم الخارجي.
etharp_request: يقوم هذا التابع بإرسال رسالة ARP لترجمة عنوان IP معين و الحصول على العنوان الفيزيائي Mac Address.
Ifconfig: سيتم شرح هذا التابع في الفقرة Configuration.

7 5 5 -تنفيذ برتوكول IP

إن عملية انجاز البرتوكول IP تقسم إلى ثلاثة مراحل:

7 5 5 1- المرحلة الأولى (التعامل مع العناوين)

تم تعريف البنية التالية للتعامل مع عناوين IP:

```
PACK_STRUCT_BEGIN
struct ip_addr {
  PACK_STRUCT_FIELD(u32_t addr);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

كما تم تعريف بعض الماكروات للتعامل مع العناوين:

```
#define IP4_ADDR(ipaddr, a,b,c,d) (ipaddr)->addr = htonl(((u32_t)(a & 0xff)
<< 24) | ((u32_t)(b & 0xff) << 16) | \ ((u32_t)(c & 0xff) << 8) | (u32_t)(d
                                                                & 0xff))
```

IP4_ADDR: يقوم هذا الماكرو بتشكيل هذا عنوان IP بالشكل الثنائي.

أما أهم التوابع للتعامل مع العناوين فهي:

ip_addr_isbroadcast: يقوم هذا التابع بتحديد هل عنوان IP الممرر إليه هو عنوان تعميم Broadcast أو لا.
inet_ntoa: يقوم هذا التابع بالتأكد من أن العنوان IP الممر إليه كنص ASCII هل هو صحيح.
inet_addr: يستخدم هذا التابع السابق للتأكد من صحة العنوان IP الممر كنص ويقوم بتحويله إلى التنسيق الثنائي.

2 5 5 7 -المرحلة الثانية (عمليات الإرسال و التوجيه)

تم تعريف البنية التالية ip_hdr التي تمثل IP Header:

```
PACK_STRUCT_BEGIN
struct ip_hdr {
  PACK_STRUCT_FIELD(u16_t _v_hl_tos);
  PACK_STRUCT_FIELD(u16_t _len);
  PACK_STRUCT_FIELD(u16_t _id);
  PACK_STRUCT_FIELD(u16_t _offset);

#define IP_RF 0x8000        /* reserved fragment flag */
#define IP_DF 0x4000        /* dont fragment flag */
#define IP_MF 0x2000        /* more fragments flag */
#define IP_OFFMASK 0x1fff  /*mask for fragmenting bits */

  PACK_STRUCT_FIELD(u16_t _ttl_proto);
  PACK_STRUCT_FIELD(u16_t _chksum);
  PACK_STRUCT_FIELD(struct ip_addr src);
  PACK_STRUCT_FIELD(struct ip_addr dest);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

اما بالنسبة لعمليات التوجيه و الارسال فانه تم استخدام لائحة netif كجدول للتوجيه.

أهم التوابع المشكلة للبرتوكول IP:

ip_init: يقوم هذا التابع بتهيئة البرتوكول IP.
ip_route: يقوم هذا التابع في البحث ضمن اللائحة المترابطة netif لإيجاد بطاقة الشبكة المناسبة لعملية الارسال. حيث يتم التطابق إذا كان قناع بطاقة الشبكة Network Mask مساوي للقناع العنوان الذي نريد الارسال إليه.
ip_forword: يقوم هذا التابع بالبحث عن بطاقة الشبكة المناسبة لعملية الارسال ثم يقوم بتمرير الحزم إليها.
ip_input: يتم استدعاء هذا التابع من قبل مشغل الشبكة Network Device Driver عند وصول حزم IP. يقوم التابع بفحص الحزم فإذا كانت غير موجهة لنا يقوم بتمريرها باستخدام عنوان Gateway واذا كانت الحزم موجهة لنا يتم إرسالها إلى الطبقة الأعلى (TCP or UDP).
ip_output_if: يقوم هذا التابع بإنشاء حزمة IP ويقوم بإرسالها.

3 5 5 7 -المرحلة الثالثة (عمليات التقسيم و التجميع)

إن التوابع المستخدمة في التجميع و التقسيم:

ip_frag: يقوم هذا التابع بتجزئة الحزم إذا كانت كبيرة جداً.
ip_reass: يقوم هذا التابع بتجميع الحزم و ذلك بمقارنة الحزم القادمة هل تنتمي إلى ذاكرة التجميع Buffer.

7 5 6 -تنفيذ برتوكول ICMP

إن عملية تنفيذ البرتوكول ICMP هي سهلة بعض الشيء فكل ما علينا فعله هو إنشاء بني معطيات تمثل رسائل ICMP بالإضافة إلى كتابة التابع المناسب لتمرير هذه الرسائل إلى الطبقة العليا. قمنا بإنشاء بني لثلاثة رسائل هي:

بنية المعطيات icmp_echo_hdr و تمثل الرسالة Echo or Echo Reply Message.
بنية المعطيات icmp_dur_hdr و تمثل الرسالة Destination Unreachable Message.
بنية المعطيات icmp_te_hdr و تمثل الرسالة Time Exceeded Message.

```
PACK_STRUCT_BEGIN
struct icmp_echo_hdr {
  PACK_STRUCT_FIELD(u16_t _type_code);
  PACK_STRUCT_FIELD(u16_t chksum);
  PACK_STRUCT_FIELD(u16_t id);
  PACK_STRUCT_FIELD(u16_t seqno);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END

PACK_STRUCT_BEGIN
struct icmp_dur_hdr {
  PACK_STRUCT_FIELD(u16_t _type_code);
  PACK_STRUCT_FIELD(u16_t chksum);
  PACK_STRUCT_FIELD(u32_t unused);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END

PACK_STRUCT_BEGIN
struct icmp_te_hdr {
  PACK_STRUCT_FIELD(u16_t _type_code);
  PACK_STRUCT_FIELD(u16_t chksum);
  PACK_STRUCT_FIELD(u32_t unused);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

فيما يلي أهم التوابع المشكلة للبرتوكول ICMP:

icmp_input
icmp_dest_unreach
icmp_time_exceeded

7.5.7- تنفيذ البرتوكولين TCP & UDP

7.5.7.1- البرتوكول TCP

تم تمثيل TCP Header باستخدام البنية tcp_hdr:

```
PACK_STRUCT_BEGIN
struct tcp_hdr {
  PACK_STRUCT_FIELD(u16_t src);
  PACK_STRUCT_FIELD(u16_t dest);
  PACK_STRUCT_FIELD(u32_t seqno);
  PACK_STRUCT_FIELD(u32_t ackno);
  PACK_STRUCT_FIELD(u16_t _hdrlen_rsvd_flags);
  PACK_STRUCT_FIELD(u16_t wnd);
  PACK_STRUCT_FIELD(u16_t chksum);
  PACK_STRUCT_FIELD(u16_t urgp);
} PACK_STRUCT_STRUCT;
PACK_STRUCT_END
```

كما تم تعريف البنية tcp_state لتمثيل حالات الاتصال في TCP:

```
enum tcp_state {
  CLOSED      = 0,
  LISTEN      = 1,
  SYN_SENT    = 2,
  SYN_RCVD    = 3,
  ESTABLISHED = 4,
  FIN_WAIT_1  = 5,
  FIN_WAIT_2  = 6,
  CLOSE_WAIT  = 7,
  CLOSING     = 8,
  LAST_ACK    = 9,
  TIME_WAIT   = 10
};
```

تمثل بنية المعطيات tcp_pcb (كتلة التحكم بالبرتوكول) الشكل العام للبرتوكول TCP، حيث نقوم من خلال هذه البنية إدارة اتصالات TCP وعمليات الاستقبال و الارسال من و إلى المنافذ المناسبة. بالإضافة إلى القيام بالاستدعاء المناسبة للتوابع عند حدوث أفعال معينة، كما نستطيع باستخدام هذه البنية الحصول على معلومات عن منطقة الذاكرة المخصصة للإرسال و الاستقبال.

```
/* the TCP protocol control block */
struct tcp_pcb {
/** common PCB members */
  IP_PCB;

  struct tcp_pcb *next; /* for the linked list */
  enum tcp_state state; /* TCP state */
  u8_t prio;
  void *callback_arg;

  u16_t local_port;
  u16_t remote_port;

  u8_t flags;
#define TF_ACK_DELAY (u8_t)0x01U
#define TF_ACK_NOW   (u8_t)0x02U
#define TF_INFR      (u8_t)0x04U
#define TF_RESET     (u8_t)0x08U
#define TF_CLOSED    (u8_t)0x10U
#define TF_GOT_FIN   (u8_t)0x20U
#define TF_NODELAY   (u8_t)0x40U

  /* receiver variables */
  u32_t rcv_nxt;
  u16_t rcv_wnd;

  /* Timers */
  u32_t tmr;
  u8_t polltmr, pollinterval;

  /* Retransmission timer. */
  u16_t rtime;

  u16_t mss;

  /* RTT (round trip time) estimation variables */
  u32_t rttest;
  u32_t rtseq;
  s16_t sa, sv;
  u16_t rto;
  u8_t nrtx;

  /* fast retransmit/recovery */
  u32_t lastack;
  u8_t dupacks;


  /* idle time before KEEPALIVE is sent */
  u32_t keepalive;

  /* KEEPALIVE counter */
  u8_t keep_cnt;
};
```

```
  /* congestion avoidance/control variables */
  u16_t cwnd;
  u16_t ssthresh;

  /* sender variables */
  u32_t snd_nxt,
    snd_max,
    snd_wnd,
    snd_wl1, snd_wl2,
    snd_lbb;

  u16_t acked;

  u16_t snd_buf;
  u8_t snd_queuelen;


  /* These are ordered by sequence number: */
  struct tcp_seg *unsent;
  struct tcp_seg *unacked;

  struct tcp_seg *ooseq;

/* Function to be called when more send buffer space is available.*/
  err_t (* sent)(void *arg, struct tcp_pcb *pcb, u16_t space);
/* Function to be called when (in-sequence) data has arrived. */
  err_t (* recv)(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err);
/* Function to be called when a connection has been set up. */
  err_t (* connected)(void *arg, struct tcp_pcb *pcb, err_t err);
/* Function to call when a listener has been connected. */
  err_t (* accept)(void *arg, struct tcp_pcb *newpcb, err_t err);
/* Function which is called periodically. */
  err_t (* poll)(void *arg, struct tcp_pcb *pcb);
  /* Function to be called whenever a fatal error occurs. */
  void (* errf)(void *arg, err_t err);
```

فيما يلي شرح لأهم حقول هذه البنية:

next: مؤشر إلى كتلة تحكم بالبرتوكول يستخدم لإنشاء لائحة مترابطة من الاتصالات الفعّالة.
state: يمثل هذا الحقل حالة البرتوكول.
callback_arg: مؤشر إلى وسيط خارجي لاستخدامه داخل البرتوكول.
local_port: المنفذ المحلي.
remote_port: المنفذ البعيد.
rcv_nxt: الرقم التسلسلي للحزمة المتوقع استقبالها.
rcv_wnd: نافذة المستقبل.
rto: الزمن الذي يجب أن يمر قبل تنفيذ عملية إعادة الارسال.
nrtx: الرقم التسلسلي للحزمة التي يجب إعادة إرسالها.

snd_nxt: الرقم التسلسلي للحزمة التي يجب إرسالها.
snd_wnd: نافذة المرسل.
snd_buf: حجم الذاكرة بالبايت المتوفر من اجل الارسال.
sent: مؤشر إلى تابع يتم استدعاءه عند توفر مساحة فارغة ضمن ذاكرة buffer الارسال.
recv: مؤشر إلى تابع يتم استدعاءه عند استقبال حزمة بيانات.
connected: مؤشر إلى تابع يتم استدعاءه عند تأسيس اتصال.
accept: مؤشر إلى تابع يتم استدعاءه عند قبول اتصالات جديدة.
poll: مؤشر إلى تابع يتم استدعاءه بشكل متكرر بتأخير زمني معين.
errf: مؤشر إلى تابع يتم استدعاءه عند حدوث خطأ قاتل.

عندما نقوم بتهيئه TCP ليقوم بالإصغاء لقبول اتصالات قادمة تصبح العديد من حقول البنية السابقة بلا فائدة لذلك قمنا بإنشاء البنية tcp_pcb_listen التي تحوي على البيانات التي نحتاجها فقط خلال عملية انتظار قدوم اتصالات جديدة وذلك بهدف توفير الذاكرة بالإضافة إلى السرعة في التنفيذ.

```
struct tcp_pcb_listen {
  IP_PCB;
  struct tcp_pcb_listen *next
  enum tcp_state state;   /* TCP state */
  u8_t prio;
  void *callback_arg;
  u16_t local_port;
  err_t (* accept)(void *arg, struct tcp_pcb *newpcb, err_t err);
};
```

فيما يلي شرح لتوابع واجهة TCP:

tcp_init: يقوم هذا التابع بتهيئة البرتوكول TCP.
tcp_tmr: يستدعى هذا التابع بشكل متكرر بزمن قدره 250 ميلي ثانية وذلك لقدح مؤقتات البرتوكول TCP.
tcp_new: يقوم هذا التابع بإنشاء كتلة تحكم بالبرتوكول PCB وإضافتها إلى اللائحة المترابطة.
tcp_arg: يتم استخدام هذا التابع لتمرير وسطاء إلى التوابع Callback.
tcp_accept: يتم استخدام هذا التابع لتحدد تابع Callback يتم استدعاءه عند حدوث اتصال مع المضيف.
tcp_recv: يتم استخدام هذا التابع لتحديد تابع Callback يتم استدعاء عند وصول البيانات ويقوم تابع Callback بإعادة القيمة NULL ليشير إلى إن الاتصال قد انتهي مع المضيف البعيد.
tcp_sent: يتم استخدام هذا التابع لتحديد تابع Callback يتم استدعاء عند وصول البيانات بنجاح إلى المضيف البعيد.
tcp_recved: يجب استدعاء هذا التابع عند استقبال البيانات من الوجهة البعيدة.
tcp_bind: ربط كتلة تحكم بالبرتوكول PCB بعنوان IP محلي و منفذ معين. يمكن إن يكون عنوان IP مساوي لـIP_ADDR_ANY وذلك لربط الاتصال بجميع عناوين IP المحلية

tcp_connect: يقوم هذا التابع بإعداد كتلة تحكم بالبرتوكول PCB للاتصال مع وجهة بعيدة ويقوم بإرسال أشارة SYN لفتح الاتصال.
tcp_listen: يقوم هذا التابع بأمر كتلة التحكم بالبرتوكول PCB بالإصغاء لقول الاتصالات الواردة. عند قبول أي اتصال باستخدام التابع tcp_accept ينتهي تنفيذ هذا التابع.
tcp_abort: يقوم هذا التابع بقطع الاتصال بإرسال إشارة RST إلى الوجهة البعيدة ويقوم بتحرير كتلة التحكم بالبرتوكول PCB من الذاكرة. إذا تم قطع الاتصال بسبب حدوث خطأ ما فيتم تنبيه التطبيق باستخدام تابع Callback لمعالجة الأخطاء حيث يتم تحديد هذا التابع باستخدام التابع tcp_err.
tcp_close: يقوم هذا التابع بإغلاق الاتصال مع الوجهة البعيدة.
tcp_write: يقوم هذا التابع بعملية إرسال البيانات إلى الوجهة البعيدة و يجب استدعاء التابع tcp_sndbuf للتأكد من وجود ذاكرة كافية للإرسال.
tcp_setprio: باستخدام هذا التابع نستطيع تحديد أولوية الاتصال.

7 5 2 7- البرتوكول UDP

إن عملية تنفيذ البرتوكول UDP مشابهه إلى حد كبير لـTCP. فيما يلي بنية المعطيات الممثلة لـ UDP Header.

```
struct udp_hdr {
  PACK_STRUCT_FIELD(u16_t src);
  PACK_STRUCT_FIELD(u16_t dest);  /* src/dest UDP ports */
  PACK_STRUCT_FIELD(u16_t len);
  PACK_STRUCT_FIELD(u16_t chksum);
} PACK_STRUCT_STRUCT;
```

و كتلة التحكم بالبرتوكول الخاصة بـUDP ممثلة ببنية المعطيات udp_pcb وهي مشابهة إلى تلك الخاصة بـ TCP عدا أنها ابسط بكثير.

```
struct udp_pcb {
/* Common members of all PCB types */
  IP_PCB;
/* Protocol specific PCB members */
  struct udp_pcb *next;
  u8_t flags;
  u16_t local_port, remote_port;
  u16_t chksum_len;

  void (* recv)(void *arg, struct udp_pcb *pcb, struct pbuf *p,     struct
ip_addr *addr, u16_t port);

  void *recv_arg;
};
```

فيما يلي شرح لتوابع واجهة UDP:

udp_new: يتم إنشاء كتلة تحكم بالبروتوكول PBC تستخدم البروتوكول UPB في عملية الاتصال.

udp_remove: يقوم هذا التابع بحذف جميع كتل التحكم بالبروتوكول PCBs المنتهية.
udp_bind: ربط كتلة التحكم بالبرتوكول PCB إلى عنوان محلي. يمكن أن يكون عنوان IP مساوي لـIP_ADDR_ANY وذلك لربط الاتصال بجميع عناوين IP المحلية
udp_connect: يقوم هذا التابع بإسناد عنوان الوجهة البعيدة إلى كتلة التحكم بالبرتوكول PCB، لا يقوم هذا التابع بأي عملية نقل لتأسيس الاتصال.
udp_disconnect: يقوم هذا التابع بحذف عنوان الوجهة البعيدة من كتلة التحكم بالبرتوكول PCB، لا يقوم هذا التابع بأي عملية نقل لقطع الاتصال.
udp_recv: يستخدم هذا التابع لتحديد تابع Callback يتم استدعاءه عند وصول حزم UDP.
udp_sendto: يستخدم هذا التابع لإرسال حزم بيانات إلى وجهة محددة.
udp_send: يقوم هذا التابع بإرسال حزم البيانات إلى الوجهة البعيدة.
udp_input: يقوم هذا التابع بمعالجة البيانات المستقبلة.

7 5 8 -نظام حزم البيانات

نقوم ببناء حزم البيانات باستخدام بنية المعطيات pbuf. تدعم هذه البنية الحجز الديناميكي للذاكرة من اجل حفظ محتوى الحزم آو يمكن أن تقوم بالتأشير على محتوى موجود ضمن الذاكرة RAM. يتم حجز سريع للذاكرة من اجل الحزم المستقبلة من خلال النوع PBUF_POOL والذي يكون ذو حجم ثابت. يمكن أن تكون حزمة معينة مقسمة على سلسلة من pbuf أي كلائحة مترابطة تسمى pbuf chain.كما يمكن آن نسلسل عدد من الحزم ضمن لائحة مترابطة تسمى packet queue، لذا تتألف سلسلة حزم packet queue من واحدة أو اكثر من pbuf chain. سلسلة الحزم لا تستخدم سوى ضمن البرتوكول ARP.

هنالك عدة أنواع من pbuf لكل منها استخداماتها المعينة:

PBUF_RAM: يتم استخدام هذا النوع لحجز قطعة كبيرة واحدة في الذاكرة لاستخدامها في إنشاء الـHeaders للبرتوكولات.
PBUF_ROM: لا يتم حجز إي منطقة في الذاكرة لهذا النوع من pbuf. حيث يتم حجز pbuf أخري وإضافتها إلى مقدمة pbuf من هذا النوع.
PBUF_REF: أيضا في هذا النوع لا يتم حجز أي منطقة في الذاكرة.
PBUF_POOL: يتم في هذا النوع حجز pbuf كـpbuf chain.

فيما يلي تعريف أنواع الـ pbuf:

```
typedef enum {
  PBUF_RAM,
  PBUF_ROM,
  PBUF_REF,
  PBUF_POOL
} pbuf_flag;
```

أيضا تعريف الـ pbuf لاحظ أن الحقل payload هو الذي يحوي البيانات الفعلية للحزم.

```
struct pbuf {
  struct pbuf *next;
  void *payload;
  u16_t tot_len;
  u16_t len;
  u16_t flags;
  u16_t ref;
};
```

أهم التوابع المشكلة للنظام حزم البيانات:

pbuf_init: يقوم هذا التابع بتهيئة نظام الحزم حيث يتم حجز كمية كبيرة من الذاكرة لحفظ الحزم.
pbuf_alloc: يقوم بحجز pbuf في الذاكرة بنوع معين حيث يتم تحديد الحجم من قبل الطبقة التي تقوم بالحجز.
pbuf_realloc: يقوم هذا التابع بتقليص حجم pbuf الممرة إليه لتناسب طول معين.
pbuf_header: يقوم هذا التابع بتكيّف المؤشر payload لإظهار أو إخفاء الـHeader ضمن payload.
pbuf_free: يقوم هذا التابع بتحرير جميع الحزم الغير مستخدمة ضمن سلسلة من الحزم وماذا كان حجم السلسلة هو 0 فانه يقوم بتحريها أيضا.
pbuf_clen: يقوم هذا التابع بإرجاع عدد الحزم ضمن سلسلة من الحزم.
pbuf_take: يقوم بإنشاء نسخة عن pbuf من نوع PBUF_REF ووضعها ضمن pbuf من نوع PBUF_POOL.

إعداد الشبكة ضمن نظام التشغيل

لإعداد الكتل البرمجية المشكلة للبرتوكول TCP/IP قمنا بإنشاء الأمر ipconfig. لهذا الأمر خيارين up, down.إذا اخترنا الخيار up سيقوم محرر الأوامر باستدعاء التابع ifconfig و الذي يقوم بتنفيذ سلسلة الأوامر التالية:

إنشاء عنوان IP و قناع الشبكة Network Mask و Gateway.
تهيئة نظام الذاكرة و نظام حزم البيانات ونظام الإحصائيات.
استدعاء تابع تهيئة بطاقة الشبكة و الذي يقوم بالبحث عنها ضمن الممرات و تفعيلها و إسناد Netif لها كما يقوم بتهيئة البرتوكول ARP.
عند نجاح العملية السابقة يتم تهيئة برتوكولات النقل TCP و UDP.
. إسناد عنوان IP و قناع الشبكة Network Mask و Gateway لـ Netif وجعلها واجهة الشبكة الافتراضية.
إظهار بعض المعلومات للمستخدم لدلالة على نجاح عملية التهيئة أو فشلها، وفي حال النجاح يتم عرض العنوان الفيزيائي و عنوان IP و قناع الشبكة Network Mask.

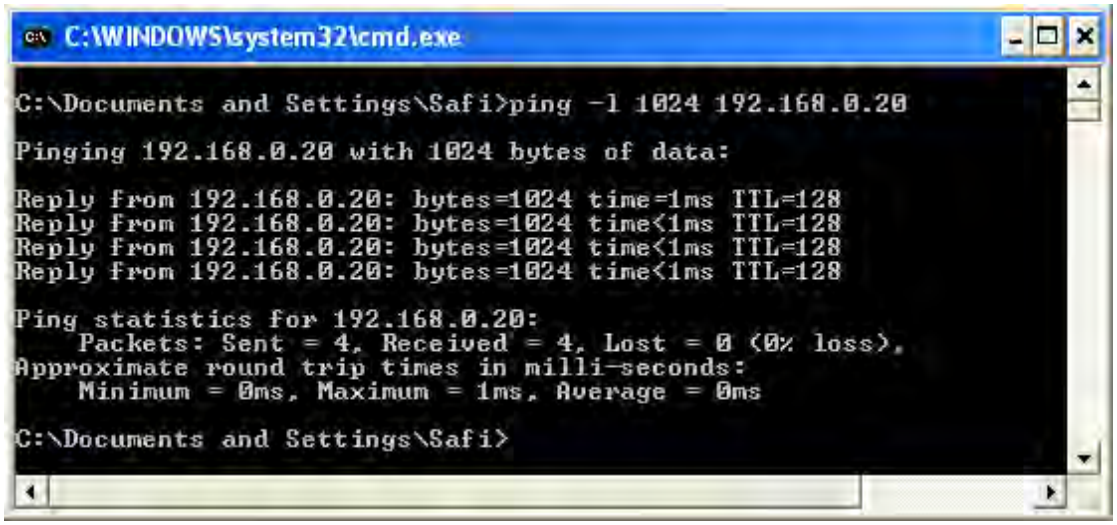
يبين الجدول التالي تسلسل العمليات (التابع ifconfig موجود ضمن الملف etharp.c)

```
void ifconfig(char *cmd)
{
        rtl8139_t *rtl = netif->state;
        struct ip_addr ip, netmask, gw;
.........
        // Get the host IP and netmask    //
        IP4_ADDR(&ip, 192, 168, 0, 20 );
        IP4_ADDR(&netmask, 255, 255, 255, 0);

        mem_init();
        memp_init();
        stats_init();
        pbuf_init();

        rtl8139_init(strstr(cmd, "promisc");
        rtl = netif->state;
        udp_init();
        tcp_init();
        netif_set_addr(netif, &ip, &netmask, NULL);
        netif_set_default(netif);
        // Dump the ethernet card informations//
        rtl8139_dump_info(rtl);
.....................
}
```

التطبيقات

1- 8 5 7 (PING) Packet Internet Groper

PING هي أداة شائعة تقوم بتحديد فيما إذا كان عنوان IP محدد قابل للوصول أو لا. تعمل هذه الأداة بإرسال حزم بيانات إلى عنوان محدد و تنتظر الإجابة، حيث تستخدم رسائل البرتوكول ICMP (ECHO_REQUEST و ECHO_REPLY) للوصول إلى الوجهة البعيدة. تستخدم هذه الأداة لحل مشاكل الاتصالات ضمن الشبكة.قمنا بنجاح بعمل PING بين جهازين الأول يحوي نظام التشغيل WINDOWS و الآخر يحوي نظام تشغيلنا حيث يتم العملية بنجاح.



الشكل77

7 6 -المراجع

PCI System Architecture (4th Edition) by Inc. MindShare, Tom Shanley, Don Anderson

REALTEK 3.3V SINGLE CHIP FAST ETHERNET CONTROLLER WITH POWER MANAGEMENT RTL8139C(L) by Realtek Corp. 2002/01/10

Network Buffers And Memory Management by Alan Cox 29, September 1996. http://www.tldp.org/LDP/khg/HyperNews/get/net/net-intro.html

Design and Implementation of the lwIP TCP/IP Stack by Adam Dunkels February 20, 2001

Understanding the Network by Michael J. Martin 2000

RFC 791 - Internet Protocol

RFC 792 - Internet Control Message Protocol

RFC 793 - Transmission Control Protocol

RFC 826 - Ethernet Address Resolution Protocol

RFC 768 (rfc768) - User Datagram Protocol

الباب □ . الدخل والخرج

8 -مقدمة إلى الدخل / الخرج [1]

8 .1 -الدخل / الخرج المعنون كذاكرة

كل متحكم فيه عدة مسجلات تستخدم للاتصال مع المعالج. يستطيع نظام التشغيل من خلال الكتابة عليها أن يأمر الجهاز بإرسال البيانات أو استقبال البيانات أو تشغيل أو إطفاء نفسه أو القيام بأي فعل آخر. وبالقراءة من هذه المسجلات، يستطيع نظام التشغيل معرفة حالة الجهاز، هل هو جاهز لاستقبال أمر جديد أو غير ذلك.

بالإضافة إلى مسجلات التحكم هذه، يوجد في العديد من الأجهزة مخازن بيانات مؤقتة يستطيع نظام التشغيل القراءة منها أو الكتابة عليها. مثلاً، من الطرق الشائعة لإظهار نقاط على الشاشة استخدام ذاكرة إظهار، وهي عبارة عن مخزن بيانات مؤقت Buffers متاح للبرامج ونظام التشغيل للكتابة عليه.

المسألة التي تواجهنا هنا هي كيف يستطيع المعالج الاتصال مع مسجلات التحكم ومخازن البيانات المؤقتة. هناك طريقتان: في الطريقة الأولى، يعطى لكل مسجل تحكم رقم منفذ دخل / خرج ويكون عبارة عن رقم صحيح بطول 8 أو 16 بت. وتستخدم تعليمات خاصة للدخل / الخرج مثل:

```
In      reg, port
```

يستطيع المعالج بهذه التعليمة قراءة مسجل التحكم ذي الرقم وتخزين الناتج في مسجل المعالج. وبشكل مشابه، يمكن استخدام التعليمة:

```
Out     reg, port
```

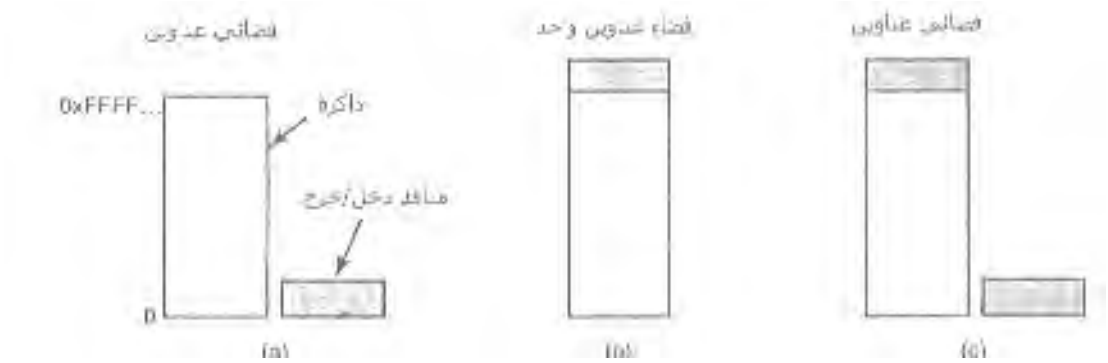
لكتابة محتويات المسجل إلى مسجل تحكم. معظم الحواسب القديمة، بما فيها جميع الأجهزة الضخمة تقريباً، مثل وأخلافه، كانت تعمل بهذه الطريقة.

في هذه الطريقة، تكون فضاءلت العنونة لكل من الذاكرة والدخل / الخرج منفصلة عن بعضها، كما في الشكل. فالتعليمتان:

```
In          r0, 4
Mov         r0, 4
```

مختلفتان تماماً في هذا التصميم. تقرأ الأولى محتويات منفذ الدخل / الخرج رقم 4 وتضعها في المسجل. بينما تقرأ الثانية محتويات الذاكرة 4 وتضعها في المسجل. أي أن الرقمين 4 في هذين المثالين يشيران إلى فضائي عناوين مختلفين وغير مترابطين.

[1] الفصل الخامس من كتاب تصميم وتنفيذ نظم التشغيل الحديثة لدار شعاع.



الشكل 78

ظهرت الطريقة الثانية مع جهاز PDP-11، وتتمثل بوضع جميع مسجلات التحكم ضمن فضاء الذاكرة كما في الشكل. يعطى لكل مسجل تحكم عنوان ذاكرة فريد غير مرتبط بحجرة ذاكرة. يدعى هذا النظام بالدخل / الخرج المعنون كذاكرة. عادة، تكون العناوين المخصصة للدخل / الخرج في قمة فضاء العناوين. هناك طريقة هجينة فيها مخازن مؤقتة معنونة كذاكرة memory maped-I/O ومنافذ دخل / خرج منفصلة لمسجلات التحكم كما في الشكل.

يستخدم معالج بنتيوم هذه الطريقة حيث تكون العناوين بين و محجوزة لمخازن البيانات المؤقتة للأجهزة في الحواسب المتوافقة مع IBM PC، بالإضافة إلى منافذ دخل / خرج من 0 حتى 64KB.

.كيف تعمل هذه التركيبات ؟ في جميع الحالات، عندما يرغب المعالج بقراءة كلمة، إما من الذاكرة أو من منفذ دخل / خرج، فإنه يضع العنوان الذي يريده على خطوط العنونة ثم يعطي إشارة Read على ممر التحكم. توجد إشارة أخرى تستخدم لتدل إذا ما كانت العملية عملية ذاكرة أم دخل / خرج. إذا كانت عملية ذاكرة تستجيب الذاكرة للأمر، أما إذا كانت عملية دخل / خرج فإن جهاز الدخل / الخرج يستجيب. إذا كان هناك فضاء عنونة واحد _(كما في الشكل) فإن جميع وحدات الذاكرة والدخل / الخرج تقوم بمقارنة العنوان الموجود على خطوط العنونة مع مجال العناوين التي تستجيب لها. إذا كان العنوان ضمن مجال العناوين المخصص لهذه الوحدة، فإنها تستجيب الأمر. وبما أنه لا يوجد عنوان مخصص لكل من الدخل / الخرج والذاكرة بنفس الوقت، فإنه لن يحدث أي تعارض.

لكل من طريقتي عنونة المتحكمات نقاط ضعف ونقاط قوة مختلفة. لنبدأ أولاً بمزايا الدخل / الخرج المعنون كذاكرة. أولاً، إذا كانت عملية القراءة والكتابة من وإلى مسجلات التحكم بالجهاز تتطلب تعليمات دخل / خرج خاصة، فإن الوصول إليهم يتطلب استخدام شفرة بلغة التجميع لأنه لا يمكن تنفيذ تعليمات الدخل والخرج و من لغة C أو C++. إن استدعاء إجراءات كهذه يضيف عبئاً آخر على عملية الدخل / الخرج. أما في حالة الدخل / الخرج المعنون كذاكرة، فإن مسجلات التحكم بالأجهزة تصبح كأنها متحولات في الذاكرة ويمكن عونتها في C بنفس طريقة عنونة أي متحول. وبالتالي، يمكن كتابة برامج تشغيل أجهزة الدخل / الخرج في حالة الدخل / الخرج المعنون كذاكرة، بشكل كامل بلغة C. أما بدون العنونة كذاكرة، فإننا نحتاج لبعض شفرة التجميع.

ثانياً، لا حاجة، في حالة الدخل / الخرج المعنون كذاكرة، إلى آلية خاصة للحماية من أجل منع عمليات المستخدم من تنفيذ تعليمات دخل / خرج. كل ما يحتاج نظام التشغيل إلى عمله

هو إزالة الجزء من فضاء العناوين الذي يحوي مسجلات التحكم من فضاء العناوين الظاهري الخاص بعمليات المستخدم. بالإضافة إلى ذلك. إذا كانت مسجلات التحكم لكل جهاز موجودة في صفحة مستقلة من فضاء العناوين، يستطيع نظام التشغيل إعطاء إمكانية التحكم بجهاز معين لمستخدم معين دون الآخرين، وذلك ببساطة عن طريق وضع الصفحات المرغوبة في جدول الصفحات الخاص بعملية المستخدم. تسمح هذه الطريقة بوضع برامج تشغيل أجهزة مختلفة في فضاءات عناوين مختلفة، وهذا لا يؤدي فقط لتصغير حجم النواة، لكنه أيضاً يمنع برامج التشغيل من التأثير على بعضها البعض.

ثالثاً، في حالة الدخل / الخرج المعنون كذاكرة، كل تعليمة تستطيع الإشارة إلى الذاكرة تستطيع أيضاً الإشارة إلى مسجلات التحكم. مثلاً، إذا كان لدينا التعليمة التي تفحص كلمة ذاكرة إذا ما كانت 0، تستطيع هذه التعليمة أيضاً فحص مسجل التحكم إذا كان 0 أم لا، وربما يكون ذلك إشارة إلى أن الجهاز خامل ويستطيع استقبال أمر جديد. وتكون شفرة التجميع المناسبة كما يلي:

```
loop:   test port_4
                BEQ   ready
                Branch        loop
Ready
```

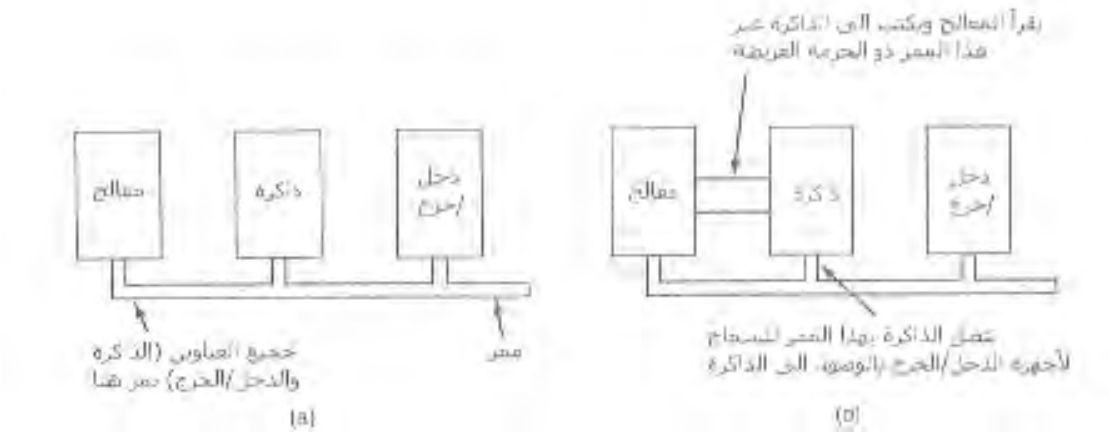
تفحص التعليمة الأولى منفذ الدخل / الخرج 4 إذا كان 0 أم لا، إذا كان 0 فإن التعليمة الثانية تقفز على اللافتة ready وألا فإن التعليمة الثالثة ستنفذ والتي تقفز إلى اللافتة loop للقيام بالفحص مجدداً. إذا لم تكن العنونة كذاكرة موجودة، يجب أولاً قراءة مسجل التحكم إلى المعالج ثم فحصه. وهذا يتطلب تعليمتين عوضاً عن واحدة. وفي حالة الحلقة المبينة أعلاه، نحتاج إلى تعليمة رابعة، مما يبطئ الاستجابة عند اكتشاف جهاز خامل.

في مجال تصميم الحواسب، لكل شيء إيجابيات وسلبيات، وكذلك الحال بالنسبة للدخل / الخرج المعنون كذاكرة. أولاً، معظم الحواسب الجديدة تحوي شكلاً من أشكال الذاكرة المخبئية. إن تخزين مسجلات التحكم لجهاز ما في الذاكرة المخبئة قد يؤدي إلى أخطاء جسيمة. لندرس مثلاً الحلقة السابقة المكتوبة بلغة التجميع في حالة وجود ذاكرة مخبئية.

الإشارة الأولى للمنفذ port_4 ستؤدي إلى تخزين قيمته في الذاكرة المخبئية. والإشارات التالية ستأخذ القيمة المخزنة في الذاكرة المخبئة ولن تعود إلى قراءتها من الجهاز نفسه. وبالتالي عندما يصبح الجهاز جاهزاً، لن يستطيع البرنامج اكتشاف ذلك، وسيدور ضمن الحلقة إلى ما لا نهاية.

لمنع هذه الحالة في الدخل / الخرج المعنون كذاكرة، يجب تجهيز العتاد بإمكانية إزالة تفعيل الذاكرة المخبئية انتقائياً حسب الصفحات. تضيف هذه الخاصية تعقيداً إضافياً إلى كل من العتاد ونظام التشغيل، الذي يجب أن يدير عملية التخزين المخبئي الانتقائي.

ثانياً، بما أنه يوجد فضاء عناوين واحد فقط، فإن جميع وحدات الذاكرة وأجهزة الدخل / الخرج يجب أن تفحص جميع إشارات الذاكرة لاكتشاف الإشارات التي يجب أن تستجيب لها. إذا كان الحاسب فيه ممر واحد، كما في الشكل فإن ذلك لا يمثل أي مشكلة.



الشكل 79

إلا أن الحواسب الشخصية الحديثة تتجه الآن إلى وجود ممر عالي السرعة مخصص للذاكرة كما في الشكل، كما أن الحواسب الكبيرة تحوي منذ زمن على ممرات مزدوجة. يتميز هذا الممر بأنه مصمم من أجل أداء أمثلي للذاكرة، دون التضحية من أجل أجهزة الدخل / الخرج البطيئة. حتى أن أنظمة بنتيوم فيها ثلاثة ممرات خارجية (الذاكرة و PCI ISA) كما في الشكل.

المشكلة الأساسية في وجود ممر ذاكرة منفصل في الأجهزة التي تعتمد على عنونة الدخل / الخرج كذاكرة، أن أجهزة الدخل / الخرج لا تستطيع رؤية عناوين الذاكرة لدى وجودها على ممر الذاكرة، لذلك لا تستطيع هذه الأجهزة أن تستجيب للعناوين.

طبعاً، هناك إجراءات خاصة لجعل الدخل / الخرج المعنون كذاكرة يعمل على الأنظمة التي تحوي ممرات متعددة. يتمثل أحد الحلول في إرسال جميع إشارات الذاكرة إلى الذاكرة أولاً. إذا فشلت الذاكرة في الاستجابة، يحاول المعالج إرسالها إلى الممرات الأخرى. يمكن تشغيل مثل هذا النظام لكنه يتضمن تعقيداً كبيراً في العتاد.

حل آخر يتمثل بوضع جهاز متنصت على ممر الذاكرة لتمرير جميع العناوين المارة فيه إلى أجهزة الدخل التي قد تهتم بها. المشكلة هنا أن أجهزة الدخل / الخرج قد لا تستطيع معالجة الطلبات بنفس سرعة الذاكرة.

الحل الممكن الثالث، والمستخدم في نظام بنتيوم المبين في الشكل، هو تصفية العناوين في شريحة جسرPCI. تحوي هذه الشريحة مسجلات مجال تحمّل مسبقاً عند الإقلاع. مثلاً يمكن تحديد المجال من 640KB حتى 1MB على أنه مجال غير ذاكري. تحوّل العناوين التي تقع في المجالات غير الذاكرية إلى ممر PCI عوضاً عن الذاكرة. سيئة هذه الطريقة هي الحاجة لتحديد المجالات التي تمثل ذاكرة والمجالات التي لا تمثل ذاكرة عند الإقلاع. إذاً، لكل طريقة إيجابياتها وسلبياتها، لذلك لا يمكن تجنب استخدام الحلول الوسطى والتضحية بأداء أحد أجزاء الحاسب لحساب الآخر.

8 2 -مبادئ برمجيات الدخل / الخرج

لنبتعد الآن عن عتاد الدخل / الخرج وندرس برمجيات الدخل / الخرج. سنتعرف أولاً غاية برمجيات الدخل / الخرج ثم نتكلم عن الطرق المختلفة لإنجاز الدخل / الخرج من وجهة نظر نظام التشغيل.

8 2 1- الغاية من برمجيات الدخل / الخرج

نعتبر الاستقلالية عن الأجهزة من المفاهيم الأساسية في تصميم برمجيات الدخل / الخرج. وتعني أنه من الممكن كتابة برامج تستطيع الوصول إلى أي جهاز دخل / خرج دون تحديد نوع الجهاز بشكل مسبق. مثلاً، البرنامج المكتوب لقراءة ملف كدخل يجب أن يتمكن من قراءة ملف من القرص المرن أو القرص الصلب أو القرص المدمج، دون تعديل البرنامج من أجل كل جهاز مختلف. وبشكل مشابه يجب أن تتوفر إمكانية كتابة أمر كالتالي:

sort imput > output

ويجب أن يعمل سواءً كان imput أو ملفاً موجوداً في قرص IDE أو قرص SCSI أو قرص مرن أو من لوحة المفاتيح، وسواء كان output عبارة عن ملف موجود على أي نوع من الأقراص أو الشاشة. يتوجب على نظام التشغيل الاهتمام بالمشاكل الناشئة عن اختلاف هذه الأجهزة اختلافاً كبيراً وحاجة كل منها إلى تسلسل مختلف من الأوامر للقراءة أو للكتابة.

يرتبط مفهوم الاستقلالية عن الجهاز بشكل وثيق بمفهوم آخر هو التسمية الموحدة يجب أن يكون اسم الملف أو الجهاز عبارة عن سلسلة نصية أو رقم صحيح ولا يتعلق بنوع الجهاز بأي شكل كان.

يمكن في يونيكس ضم جميع الأقراص في هرمية نظام الملفات بطرق مختلفة بحيث لا يحتاج المستخدم أن يحفظ أي اسم يتعلق بأي جهاز، يمكن مثلاً تثبيت قرص مرن فوق الفهرس usr/mnt/fd بحيث يؤدي نسخ ملف إلى الفهرس إلى نسخ الملف إلى القرص المرن. بهذه الطريقة، أصبحت جميع الملفات والأقراص تسمى بنفس الطريقة وهي اسم المسار.

من المواضيع الهامة أيضاً في برمجيات الدخل / الخرج معالجة الأخطاء. بشكل عام، يجب معالجة الأخطاء قريباً من العتاد قدر الإمكان. إذا اكتشف المتحكم خطأ أثناء القراءة، يجب عليه أن يحاول إصلاح الخطأ بنفسه إن استطاع. أما إذا لم يستطع، عندها يجب على برنامج التشغيل معالجة الوضع، ربما بمحاولة إعادة قراءة الكتلة مرة أخرى. العديد من الأخطاء تكون عابرة وتزول عند إعادة تنفيذ العملية مثل أخطاء القراءة الناتجة عن توضع الغبار على رأس القراءة. يجب عدم إخبار الطبقات العليا بحدوث خطأ إلا عندما تعجز الطبقات الدنيا عن معالجته. في العديد من الحالات، يمكن إنجاز إصلاح الأخطاء بشفافية في المستويات السفلى دون أن يكون للمستويات العليا فكرة عن وجود خطأ.

هناك أيضاً موضوع هام آخر هو عمليات النقل المتزامنة (أو المعيقة) مقابل غير المتزامنة (أو المقادة بالمقاطعات). إن معظم الدخل / الخرج الفيزيائي غير متزامن - يبدأ المعالج عملية النقل ثم يتجه لتنفيذ شيء آخر حتى تصل المقاطعة. إن كتابة برامج مستخدم تصبح أسهل بكثير إذا كانت عمليات الدخل / الخرج معيقة (أو متزامنة) ـ عند تنفيذ استدعاء النظام، يتم إرجاء البرنامج تلقائياً ريثما تتوفر البيانات في المخزن المؤقت. ويقع على عاتق نظام التشغيل جعل العمليات التي تكون في الواقع مقادة بالمقاطعات تبدو كأنها متزامنة بالنسبة لبرامج المستخدم.

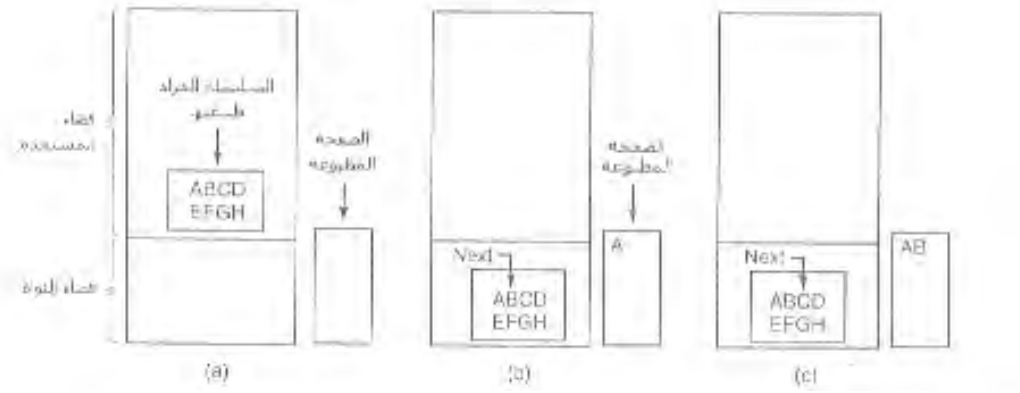
من المفاهيم الأخرى في برمجيات الدخل / الخرج هو التخزين المؤقت. لا يمكن غالباً تخزين البيانات القادمة من جهاز دخل / خرج مباشرة في وجهتها النهائية. مثلاً، عندما تصل حزمة من الشبكة، فإن نظام التشغيل لا يستطيع أن يعرف أين سيضعها حتى يخزن الحزمة مؤقتاً في مكان ما ويفحصها. كما أن بعض الأجهزة تخضع لشروط زمن حقيقي قاسية (مثل أجهزة الصوت الرقمية)، لذلك يجب وضع البيانات في مخزن إخراج مؤقت بشكل مسبق لفصل معدّل ملء المخزن عن معدّل تفريغه، وذلك لتجنب نضوب المخزن. يتضمن التخزين المؤقت كمية لا بأس بها من النسخ وغالباً ما يؤثر بشكل كبير على أداء الدخل / الخرج.

المفهوم الأخير الذي سنتكلم عنه هنا هو الأجهزة المشتركة مقابل الأجهزة المخصصة. بعض أجهزة الدخل / الخرج، مثل الأقراص، يمكن استخدامها من قبل عدة مستخدمين في نفس الوقت. ليس هناك أية مشكلة إذا قام عدة مستخدمين بفتح عدة ملفات على نفس القرص بنفس الوقت. هناك أجهزة أخرى، مثل سواقات الأشرطة، يجب أن تخصص لمستخدم معين حتى انتهائه من استخدامها. عندها، يستطيع مستخدم آخر استخدام سواقة الشريط. إن وجود مستخدمين أو أكثر يكتبان كتلاً مختلطة بشكل عشوائي على نفس الشريط سيؤدي إلى الفوضى. إن وجود أجهزة مخصصة (غير مشتركة) يؤدي أيضاً إلى ظهور عدة مشاكل مثل الاستعصاءات. مرة أخرى، يجب على نظام التشغيل أن يعالج كلاً من الأجهزة المخصصة والمشتركة بطريقة يتجنب فيها المشاكل.

8 2 2 -الدخل / الخرج المبرمج

هناك ثلاث طرق أساسية مختلفة لإنجاز الدخل / الخرج. سنتكلم في هذا القسم عن الطريقة الأولى وهي الدخل / الخرج. وسنتكلم في القسمين التاليين عن الطرقتين الأخريين وهما الدخل/ الخرج المقاد بالمقاطعات والدخل / الخرج باستخدام DMA، تتمثل الطريقة الأبسط في إنجاز الدخل / الخرج في جعل المعالج يقوم بالعمل كله. تسمى هذه الطرقة بالدخل / الخرج المبرمج.

من الأسهل توضيح الدخل / الخرج المبرمج بواسطة مثال. افترض وجود عملية مستخدم تريد طباعة سلسلة من ثمانية محارف هي ABCDEFG على الطابعة. تقوم العملية أولاً بتجميع السلسلة في مخزن مؤقت في فضاء المستخدم، كما في الشكل



الشكل80

تقوم عملية المستخدم بعد ذلك بالاستحواذ على الطابعة للكتابة من خلال القيام باستدعاء نظام لفتحها. إذا كانت الطابعة مستخدمة من قبل عملية أخرى، فإن هذا الاستدعاء سيفشل ويعيد شفرة خطأ، أو سيؤدي إلى إعاقة العملية ريثما تصبح الطابعة متاحة، وذلك حسب نظام التشغيل وال بارامترات الممررة للاستدعاء. حالما تحصل العملية على الطابعة، تقوم بإجراء استدعاء نظام آخر لجعل نظام التشغيل يطبع السلسلة على الطابعة.

يقوم نظام التشغيل عند ذلك بنسخ المخزن المؤقت الذي يحوي السلسلة إلى مصفوفة، ولتكن، في فضاء النواة، حيث يمكن الوصول إليها (لأن النواة تحتاج لتغيير خريطة الذاكرة للوصول إلى فضاء المستخدم). ثم يفحص إذا ما كانت الطابعة متاحة حالياً. إذا لم تكن متاحة، فإنه ينتظر حتى تصبح متاحة. حالما تصبح الطابعة حرة، ينسخ نظام التشغيل المحرف الأول إلى مسجل البيانات الخاص بالطابعة، وذلك باستخدام الدخل / الخرج المعنون كذاكرة في هذا المثال، يؤدي ذلك إلى تفعيل الطابعة. قد لا يظهر المحرف مباشرة في هذه المرحلة لأن بعض الطابعات تقوم بتخزين سطر كامل أو صفحة كاملة مؤقتاً قبل طباعة أي شيء. على كل، نرى في الشكل أن المحرف الأول قد طبع وأن نظام التشغيل أصبح يشير إلى الحرف على أنه الحرف التالي الذي سيطبع.

بعد طباعة الحرف الأول، يفحص نظام التشغيل إذا ما كانت الطابعة جاهزة لاستقبال حرف آخر. عموماً، تملك الطابعة مسجلاً آخر يبين حالتها. إن الكتابة إلى مسجل البيانات يؤدي إلى جعل الحالة غير جاهزة. عندما يعالج متحكم الطابعة المحرف الحالي، يقوم بالإشارة إلى جاهزة الطابعة من خلال تأهيل بت ما في سجل الحالة أو بوضع قيمة ما فيه.

ينتظر نظام التشغيل في هذه المرحلة الطابعة كي تصبح جاهزة مرة أخرى. وعندما يحدث ذلك، يقوم النظام بطباعة المحرف التالي كما في الشكل . تستمر هذه الحلقة حتى طباعة السلسلة بأكملها. يعود التنفيذ بعد ذلك إلى عملية المستخدم.

يلخص الشكل الأفعال التي يقوم بها نظام التشغيل. تنسخ البيانات أولاً إلى النواة. ثم يدخل النظام في حلقة محكمة تقوم بإخراج المحارف واحداً تلو الآخر. يوضح هذا الشكل الطبيعة الأساسية للدخل / الخرج المبرمج. وهي أنه بعد إخراج محرف، يقوم المعالج باستجواب الجهاز بشكل مستمر لمعرفة إذا ما كان جاهزاً لاستقبال محرف آخر. يسمى هذا التصرف غالباً بالاستجواب Pooling أو الانتظار المشغول busy waiting.

الدخل / الخرج المبرمج بسيط جداً، لكنه يعاني من عيب أساسي هو أنه يستهلك وقت المعالج بأكمله حتى انتهاء الدخل / الخرج بأكمله. إذا كان وقت طباعة محرف واحد قصيراً جداً (لأن كل ما تفعله الطابعة هو نسخ المحرف الجديد إلى مخزن مؤقت داخلي). فإن الانتظار المشغول ليس بالمشكلة الكبيرة. كما أنه يعتبر معقولاً في الأنظمة المضمنة التي لا يكون للمعالج فيها أي وظيفة أخرى. لكن، في الأنظمة الأكثر تعقيداً، والتي يكون للمعالج فيها أعمال أخرى يجب أن ينجزها، فإن الانتظار المشغول ليس مجدياً ونحتاج لذلك طريقة دخل / خرج أفضل.

8 2 3 -الدخل / الخرج المقاد بالمقاطعات

لندرس الآن حالة الطباعة على طابعة لا تخزن المحارف مؤقتاً بل تطبع كل واحد منها حال وصوله. إذا كانت الطابعة تستطيع طباعة 100 محرف / ثانية مثلاً، فإن كل محرف يستغرق 10 ميلي ثانية لطباعته. وهذا يعني أنه بعد كتابة كل محرف إلى مسجل بيانات الطابعة،

سيبقى المعالج خاملاً لمدة 10 ميلي ثانية منتظراً أن يسمح له بإرسال المحرف التالي. هذا الوقت كاف تماماً لإجراء تبديل سياق وتشغيل عملية أخرى أثناء فترة 10 ميلي ثانية عوضاً عن هدرها.

استخدام المقاطعات هي الطريقة التي تسمح للمعالج بالقيام بأشياء أخرى أثناء انتظاره الطابعة كي تصبح جاهزة. عند إجراء استدعاء النظام الذي يطبع السلسلة، ينسخ المخزن المؤقت إلى فضاء النواة، كما أوضحنا سابقاً، وينسخ المحرف الأول إلى الطابعة حالما تصبح جاهزة لذلك. في هذه اللحظة يستدعي المعالج المجدول ويتم تشغيل عملية أخرى. تتوقف العملية التي طلبت طباعة السلسلة حتى انتهاء طباعة السلسلة بأكملها. يبين الشكل العمل المنجز أثناء استدعاء النظام.

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

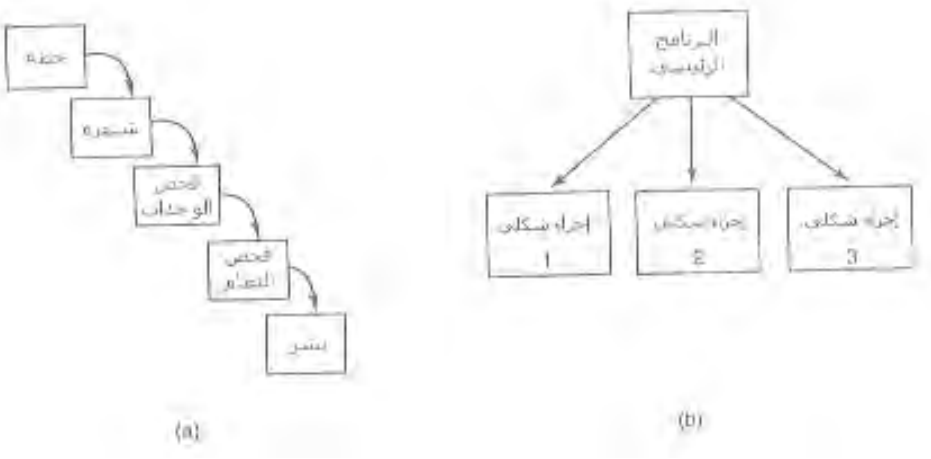
الشكل 81

عندما تطبع الطابعة المحرف وتصبح جاهزة لاستقبال المحرف التالي، تقوم بتوليد مقاطعة. تؤدي هذه المقاطعة إلى إيقاف العملية الحالية وتخزن حالتها. ثم يعمل إجراء خدمة مقاطعة الطابعة. يبين الشكل نسخة مبسطة من شفرة هذا الإجراء.

إذا لم يكن هناك أي محارف أرى للطباعة، يقوم معالج المقاطعة باستئناف عمل العملية المتوقفة. وإلا فإنه يقوم بطباعة المحرف التالي، ثم يرسل إشعاراً للمقاطعة ثم يعود إلى العملية التي كانت تعمل قبل حدوث المقاطعة، والتي تتابع التنفيذ من النقطة التي توقفت عندها.

8 2 4 -الدخل / الخرج باستخدام DMA

هناك مشكلة واضحة في الدخل / الخرج المقاد بالمقاطعات، وهي أن المقاطعة تحدث عن كل محرف. المقاطعات تستهلك وقتاً، لذلك تعتبر هذه الطريقة مضيعة لوقت المعالج. الحل هو استخدام. تتلخص الفكرة هنا في جعل متحكم يغذي الطابعة بالمحارف واحداً تلو ا لآخر دون إزعاج المعالج. في الواقع. عبارة عن دخل/خرج مبرمج، لكن متحكم هو الذي يقوم بالعمل عوضاً عن المعالج الرئيسي. يبين الشكل مخططاً عاماً للشفرة المستخدمة.



الشكل 82

الفائدة الكبرى عند استخدام DMA هي تقليل عدد المقاطعات من مقاطعة لكل محرف إلى مقاطعة واحدة لكل مخزن مؤقت يطبع. إذا كانت المحارف المراد طباعتها كثيرة، وكانت المقاطعات بطيئة، يؤدي استخدام DMA إلى تحسين كبير في الأداء. من ناحية أخرى، إذا كان متحكم غير قادر على قيادة الجهاز بسرعته القصوى، أو إذا لم يكن للمعالج عمل آخر عادة أثناء انتظاره لمقاطعة، فإن الدخل / الخرج المقاد بالمقاطعات أو حتى الدخل / الخرج المبرمج قد يكونان أفضل من استخدام DMA.

8 3 -طبقات برمجيات الدخل/الخرج

تنظم برمجيات الدخل / الخرج نموذجياً ضمن أربع طبقات كما هو موضح في الشكل. لكل طبقة وظيفة معرّفة بها وواجهة معرّفة بشكل جيد بينها وبين الطبقات المجاورة. تختلف الوظيفة والواجهة حسب نظام التشغيل، لذلك فإن الدراسة التالية التي تغطي جميع الطبقات ابتداءً من الأسفل، ليست مخصصة لأي نظام معين.

8 3 1- معالجات المقاطعات

مع أن الدخل / الخرج المبرمج مفيد أحياناً، إلا أن المقاطعات في معظم الحالات تعتبر واقعاً مفروضاً ولا يمكن تجنبها. يجب دائماً إخفاء المقاطعات بعيداً في أعماق نظام التشغيل بحيث لا يدري بها إلا جزء صغير من نظام التشغيل. الطريقة الأفضل لاخفائها هي جعل برنامج التشغيل الذي يبدأ عملية الدخل / الخرج يتوقف ريثما ينتهي الدخل / الخرج وتحدث المقاطعة. يستطيع برنامج التشغيل إعاقة نفسه بتنفيذ Down على قفل semaphore ما، أو wait على متحول شرط أو recieve على رسالة أو أي شيء آخر مشابه.

عند حدوث مقاطعة، يقوم الإجراء بما يجب عليه فعله لمعالجتها. يستطيع بعد ذلك إزالة إعاقة برنامج التشغيل الذي بدأ العملية. في بعض الحالات، يكون ذلك بتنفيذ upعلى قفل ما. وفي حالات أخرى، يقوم بتنفيذ signal على متحول شرط ضمن مراقب، أو يرسل رسالة إلى برنامج التشغيل المعاق. في جميع الحالات، تكون النتيجة أن برنامج التشغيل الذي كان معاقاً

سابقاً يستطيع الآن متابعة التنفيذ. يعمل هذا النموذج أفضل ما يمكن إذا كانت برامج التشغيل مصممة كي تعمل كعمليات نواة، مع وجود مكدسات وحالات وعدادات برامج خاصة بها.

طبعاً، الواقع ليس بهذه السهولة. إن معالجة المقاطعة ليست مجرد أخذ المقاطعة وإجراء عملية على مسير ما ثم تنفيذ تعليمة IRET للعودة من المقاطعة إلى العملية السابقة. هناك الكثير من العمل الذي يجب أن يقوم به نظام التشغيل ضمن المقاطعة. سنعطي الآن مخططاً لهذا العمل على شكل سلسلة من الخطوات التي يجب إنجازها في البرمجيات بعد اكتمال مقاطعة العتاد. لاحظ أن التفاصيل تعتمد بشكل كبير على نوع النظام، لذلك قد تكون بعض الخطوات غير لازمة في بعض الأنظمة، وقد تحتاج بعض الأنظمة إلى خطوات غير مذكورة. كما أن هذه الخطوات قد تكون بترتيب مختلف في بعض الأنظمة.

1- خزن جميع المسجلات (بما فيها PSW) التي لم تخزن من قبل عتاد المقاطعة.

2- هيء سياقاً خاصاً من أجل تنفيذ إجراء خدمة المقاطعة. قد يتضمن ذلك إعداد TLP و MMU وجدول الصفحات.

3- هيئ مكدساً لإجراء خدمة المقاطعة

4- أرسل إشعاراً باستلام المقاطعة إلى متحكم المقاطعات. إذا لم يكن هناك متحكم مقاطعات مركزي، أعد تفعيل المقاطعات.

5- انسخ المسجلات من مكان تخزينها (من المكدس مثلاً) إلى جدول العمليات.

6- نفذ إجراء خدمة المقاطعة، الذي سيستخلص المعلومات من مسجلات متحكم الجهاز الذي طلب المقاطعة.

7- اختر العملية التي يجب أن تعمل تالياً. إذا تسببت المقاطعة بجعل إحدى العمليات ذات الأفضلية العالية التي كانت معاقة جاهزة، يمكن اختيار هذه العملية كي تعمل الآن.

8- هيئ سياق MMU للعملية التي ستعمل تالياً. ربما تحتاج أيضاً لإعداد TLB

9- حمّل مسجلات العملية الجديدة، بما فيها

10- ابدأ تشغيل العملية الجديدة.

كما هو واضح، فإن معالجة المقاطعة ليست سهلة أبداً. كما أنها تستغرق قدراً لا بأس به من تعليمات المعالج، خاصة في الأنظمة التي تحوي ذاكرة ظاهرية ويجب فيها إعداد الصفحات وتهيئة حالة MMU (أي البتات R و M). في بعض الأنظمة يجب أيضاً إدارة وذاكرة المعالج المخبئية عند التبديل بين نمطي المستخدم والنواة. وهذا ما يستغرق دورات آلة إضافية.

8 3 2 -برامج تشغيل الأجهزة

تكلمنا سابقاً في هذا الفصل عن عمل متحكمات الأجهزة. ورأينا أن كل متحكم فيه بعض المسجلات التي تستخدم لإعطائه أوامر أو بعض المسجلات التي يمكن عبرها قراءة حالة

الجهاز أو مسجلات من كلا النوعين. يختلف عدد مسجلات الجهاز وطبيعة الأوامر الممكنة بشكل كبير من جهاز إلى آخر. مثلاً، يحتاج برنامج تشغيل الفأرة إلى قبول معلومات عن الفأرة تحدد المسافة التي تحركت بها والأزرار المضغوطة حالياً. على العكس من ذلك، يحتاج برنامج تشغيل الأقراص إلى معرفة معلومات عن القطاعات والمسارات والرؤوس وحركة الذراع والمحرك وزمن استقرار الرأس وجميع المعلومات الميكانيكية اللازمة لتشغيل القرص بشكل صحيح. من الواضح أن برنامجي التشغيل هذين سيكونان مختلفين جداً.

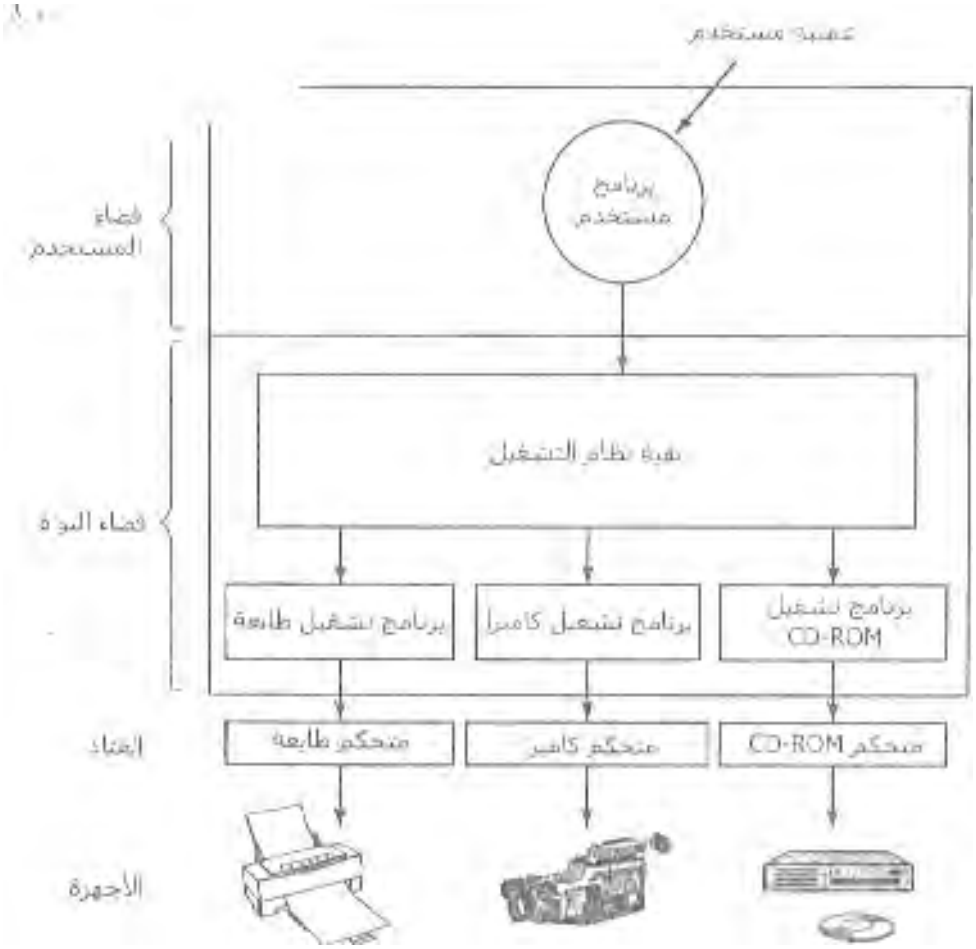
نتيجة لذلك يحتاج كل جهاز دخل / خرج متصل بحاسب إلى شفرة خاصة بالجهاز للتحكم به.

تدعى هذه الشفرة ببرنامج تشغيل الجهاز، وتكتب عادة من قبل مصنع الجهاز وتوزع معه. وبما أن كل نظام تشغيل يحتاج برامج تشغيل خاصة به، فإن مصنعي الأجهزة يوزعون برامج تشغيل لعدة أنظمة تشغيل مشهورة.

كل برنامج تشغيل يعالج عادة نوعاً واحداً من الأجهزة، أو صنفاً واحداً على الأكثر من الأجهزة المتشابهة. مثلاً، يستطيع برنامج تشغيل أقراص SCSI عادة قيادة عدة أقراص SCSI بأحجام وسرعات مختلفة وربما يستطيع قيادة سواقة أيضاً SCSI CD-ROM من ناحية أخرى، تختلف الفأرة عن عصا الألعاب اختلافاً كبيراً، لذلك تحتاج إلى برامج تشغيل مختلفة. إلا أنه لا يوجد أي قيد تقني يمنع وجود برنامج تشغيل واحد يتحكم بعدة أجهزة مختلفة. لكن ذلك لا يعتبر فكرة جيدة.

للوصول إلى عتاد الجهاز، أي مسجلات التحكم الخاصة بالجهاز، يجب على برنامج التشغيل عادة أن يكون جزءاً من نواة نظام التشغيل، على الأقل مع الأنظمة المستخدمة حالياً. من الممكن إنشاء برامج تشغيل تعمل في فضاء المستخدم بوجود استدعاءات نظام لقراءة وكتابة مسجلات الجهاز. في الواقع، يعتبر هذا التصميم فكرة جيدة، لأنه يعزل النواة عن برامج التشغيل وبرامج التشغيل عن بعضها البعض. كما أنه يخلصنا من مصدر أساسي من مصادر انهيار النظام التي تنشأ عن برامج تشغيل غير متقنة تتعارض مع النواة بشكل أو بآخر. إلا أنه باعتبار أن نظم التشغيل الحالية تتوقع من برامج التشغيل أن تعمل في النواة، فإننا سندرس هنا هذا النموذج فقط.

بما أن مصممي كل نظام تشغيل يعرفون أن بعض أجزاء الشفرة (برامج التشغيل) المكتوبة من قبل أشخاص خارجين ستثبت ضمن النظام، يجب أن يكون لنظام التشغيل بنية معمارية تسمح بإجراء هذا التثبيت. يعني ذلك وجود نموذج معرف جيداً يحدد ما يفعله برنامج التشغيل و كيفية تفاعله مع بقية أجزاء النظام. تتوضع برامج التشغيل عادة تحت بقية نظام التشغيل كما في الشكل.



الشكل83

تصنّف أنظمة التشغيل برامج التشغيل عادة ضمن فئة معينة من أصل بضع فئات قليلة. الفئات الأكثر شهرة هي الأجهزة الكتلية مثل الأقراص، والتي تحوي كتل ببيانات متعددة يمكن عنونتها بشكل مستقل. والأجهزة المحرفية، مثل لوحات المفاتيح والطابعات، والتي تولّد أو تستقبل دفقاً من المحارف.

تعرّف معظم أنظمة التشغيل واجهة قياسية يجب أن تدعمها جميع الأجهزة الكتلية، وواجهة قياسية أخرى يجب أن تدعمها جميع الأجهزة المحرفية. تتألف هاتان الواجهتان من عدد من الإجراءات التي تستطيع باقي أجزاء النظام استدعاءها لجعل برنامج التشغيل يقوم بعمل من أجلها. من الإجراءات النموذجية إجراء لقراءة كتلة (في الأجهزة الكتلية) وإجراء لكتابة محرف (في الأجهزة المحرفية).

في بعض الأنظمة، يكون نظام التشغيل عبارة عن برنامج ثنائي وحيد يحوي جميع برامج تشغيل الأجهزة التي يحتاجها مترجمة ضمنه. كان هذا النموذج الحالة الطبيعية لأنظمة يونيكس لعدة سنوات لأنها تعمل في مراكز الحاسب. حيث كانت أجهزة الدخل / الخرج قلما تتغير. إذا أضيف جهاز جديد إلى النظام، يقوم مدير النظام ببساطة بإعادة ترجمة النواة مع الجهاز الجديد لإنشاء ملف ثنائي جديد.

مع ظهور الحواسب الشخصية وأجهزة الدخل / الخرج المتنوعة، لم يعد هذا النموذج صالحاً للعمل. قلة من المستخدمين يستطيعون إعادة ترجمة أو إعادة ربط النواة، حتى عندما

يتوفر لديهم الشفرة المصدرية أو الوحدات الغرضية، وهذا نادراً ما يحدث. عوضاً عن ذلك، انتقلت أنظمة التشغيل، ابتداءً من دوس إلى نموذج يتم فيه تحميل برامج التشغيل بشكل ديناميكي إلى النظام أثناء عمله. تتعامل الأنظمة المختلفة مع تحميل برامج التشغيل بطرق مختلفة.

لبرامج التشغيل عدة وظائف. الوظيفة الأكثر وضوحاً هي قبول طلبات القراءة والكتابة المجردة من البرنامج المستقل عن الجهاز المتوضع فوقه والتأكد من تحقيق هذه الطلبات. لكن هناك بضع وظائف أخرى يجب عليه إنجازها. مثلاً، يجب على برامج التشغيل تهيئة الجهاز إذا تطلب ذلك. وقد يحتاج أيضاً لإدارة متطلبات الطاقة لديه وسجل الأحداث.

تملك العديد من برامج التشغيل نفس البنية العامة. يبدأ برنامج التشغيل النموذجي باختبار بارامترات الدخل ليرى هل هي صالحة. إذا لم تكن صالحة، يعاد خطأ. وإذا كانت صالحة، ربما تحتاج للترجمة من المصطلحات المجردة إلى الواقعية. ويعني ذلك بالنسبة للقرص تحويل رقم الكتلة الخطي إلى أرقام الرأس والمسار والقطاع والاسطوانة الخاصة بهندسة القرص.

بعد ذلك، يقوم القرص باختبار إذا ما كان الجهاز مستخدماً حالياً. إذا كان كذلك، يتم وضع الطلب ضمن الرتل من أجل معالجته لاحقاً. أما إذا كان الجهاز خاملاً، فيتم فحص حالة العتاد الصلب لمعرفة إذا ما كان يمكن معالجة الطلب الآن. ربما يلزم تشغيل الجهاز أو تشغيل محرك قبل البدء بنقل البيانات. حالما يصبح الجهاز شغالاً وجاهزاً، يبدأ التحكم الفعلي.

يعني التحكم بالجهاز إصدار سلسلة من الأوامر إليه. برنامج التشغيل هو الذي يحدد سلسلة الأوامر هذه، وذلك حسب العمل المطلوب منه. بعد أن يعرف برنامج التشغيل الأوامر التي سيصدرها، يبدأ بكتابتها إلى مسجلات متحكم الجهاز. بعد كتابة كل أمر إلى الجهاز، قد يحتاج الأمر لاختبار إذا ما كان المتحكم قد قبل الأمر وأصبح جاهزاً لاستقبال الأمر التالي. يستمر هذا التسلسل حتى إصدار جميع الأوامر. بعض المتحكمات يمكن إعطاؤها لائحة مترابطة من الأوامر في الذاكرة وجعلها تقرؤها وتعالجها بنفسها دون أية مساعدة من نظام التشغيل.

بعد إصدار جميع الأوامر، نصل إلى إحدى حالتين. يجب على برنامج التشغيل في معظم الحالات الانتظار حتى يقوم المتحكم بالعمل الذي طلبه منه، لذلك يقوم بإيقاف نفسه ريثما تأتي المقاطعة التي تعيده للعمل. في الحالات الأخرى، تنتهي العملية مباشرة ولا حاجة لإيقاف برنامج التشغيل. كمثال عن الحالة الأخيرة، يحتاج تحريك الشاشة بضعة أسطر في نمط المحارف فقط لكتابة بعض البايتات في مسجلات المتحكم. لا حاجة لأي حركة ميكانيكية، لذلك تتم العملية بأكملها خلال بضعة نانو ثانية.

في الحالة الأولى، يتم إيقاظ برنامج التشغيل المتوقف بواسطة المقاطعة. أما في الحالة الثانية، فإنه لا ينام أبداً. في كلا الطريقتين، يجب بعد انتهاء العملية اختبار الأخطاء. إذا كان شيء على ما يرام، يصبح لدى برنامج التشغيل بعض البيانات التي يجب تمريرها إلى البرنامج المستقل عن الجهاز (وهي عبارة عن كتلة البيانات التي تمت قراءتها). أخيراً، يعيد برنامج التشغيل معلومات حالة للإعلان عن الأخطاء إلى المستدعي. إذا كان هناك طلبات أخرى في الرتل، يمكن اختيار وتنفيذ أحدها. إذا لم تكن هناك أي طلبات، يتوقف برنامج التشغيل منتظراً ورود الطلب التالي.

هذا النموذج البسيط عبارة عن تقريب غير دقيق لما يحدث في الواقع. هناك العديد من العوامل التي تجعل الشفرة أكثر تعقيداً من ذلك. نذكر منها أن جهاز الدخل / الخرج قد يكمل عمله

أثناء عمل برنامج التشغيل مما يؤدي إلى مقاطعة برنامج التشغيل نفسه. وتؤدي هذه المقاطعة عادة إلى تشغيل برنامج التشغيل الحالي نفسه الذي تمت مقاطعته. مثلاً، بينما يكون برنامج تشغيل بطاقة الشبكة يعالج حزمة واردة، قد ترد حزمة أخرى. وبالتالي، يجب أن تكون برامج التشغيل قابلة لإعادة الدخول، ويعني ذلك أن برنامج التشغيل يجب أن يتوقع احتمال استدعائه مرة ثانية قبل انتهائه من الاستدعاء الأول.

في الأنظمة ذات التوصيل الحار، يمكن إضافة أو إزالة الأجهزة بينما يكون الحاسب شغالاً. نتيجة لذلك، قد يحدث أحياناً أنه أثناء قراءة برنامج التشغيل لكتلة بيانات من الجهاز، يخبره النظام بأن هذا الجهاز قد تمت إزالته من النظام. ينبغي إنهاء عملية الدخل / الخرج الحالية دون إحداث أي ضرر يبنى المعطيات التابعة للنواة، وليس ذلك فحسب، بل يجب أيضاً إزالة أي طلبات موجودة في الرتل من أجل هذا الجهاز المزال بطريقة ودية ونقل الأخبار السيئة للبرامج التي طلبتها. علاوة على ذلك، قد تؤدي إضافة جهاز جديد إلى جعل النواة تعيد توزيع الموارد (وبخاصة خطوط طلب المقاطعة). حيث تأخذ الخطوط القديمة من برنامج التشغيل وتعطيه خطوطاً جديدة بدلاً عنها.

لا يسمح لبرامج التشغيل بالقيام باستدعاءات نظام، لكنها تحتاج عادة للتواصل مع بقية أجزاء النواة. يسمح عادة باستدعاء إجراءات خاصة في النواة. مثلاً، هناك عادة إجراءات لحجز وتحرير بعض الصفحات المثبتة في الذاكرة لاستخدامها كمخازن مؤقتة. وهناك استدعاءات أخرى مفيدة لازمة لإدارة MMU والمؤقتات ومتحكم DMA ومتحكم المقاطعات وغيرها.

8 3 3 -برمجيات الدخل / الخرج المستقلة عن الجهاز

على الرغم من أن بعض برمجيات الدخل / الخرج متعلقة بنوع الجهاز، فإن أجزاءً أخرى منها مستقلة عن الجهاز. يختلف الحد الفعلي بين برامج التشغيل والبرمجيات المستقلة عن الجهاز باختلاف النظام (والجهاز، لأن بعض الوظائف التي يمكن القيام بها بطريقة مستقلة عن الجهاز يمكن أن تتم فعلياً ضمن برنامج التشغيل من أجل الكفاءة أو لأسباب أخرى. الوظائف المبينة في الشكل تنفذ نموذجياً في البرمجيات المستقلة عن الجهاز.

الشكل84

الوظيفة الأساسية للبرمجيات المستقلة عن الأجهزة هي إنجاز وظائف الدخل / الخرج المشتركة بين جميع الأجهزة لتقديم واجهة موحدة لبرمجيات مستوى المستخدم. سنتكلم تالياً عن المواضيع المذكورة أعلاه بمزيد من التفصيل.

8 3 4 -الواجهة الموحدة مع برامج التشغيل

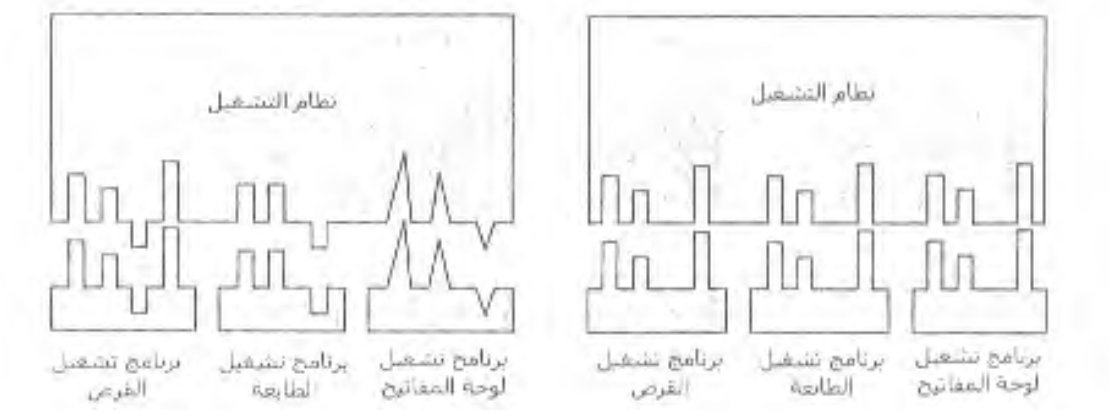
من المواضيع الهامة في أنظمة التشغيل كيفية جعل جميع أجهزة الدخل / الخرج وبرامج تشغيلها تبدو متشابهة إلى حد ما. إذا كانت الأقراص والطابعات ولوحات المفاتيح وغيرها تتصل جميعاً بطرق وواجهات مختلفة، فإن كل جهاز جديد يضاف إلى النظام سيتطلب تغيير نظام التشغيل من أجل الجهاز الجديد. إن تعديل نظام التشغيل من أجل كل جهاز جديد ليس بالفكرة الجيدة.

أحد الجوانب الهامة لهذا الموضوع الواجهة بين برنامج التشغيل وبقية أجزاء نظام التشغيل. يوضح الشكل الحالة التي يكون فيها لكل برنامج تشغيل واجهة مختلفة مع نظام التشغيل. يعني ذلك أن وظائف برنامج التشغيل المتاحة للاستدعاء من قبل النظام تختلف من برنامج تشغيل إلى آخر. وقد يعني ذلك أيضاً أن وظائف النواة التي يحتاجها برنامج تشغيل تختلف من برنامج تشغيل إلى آخر. ويعني ذلك بشكل عام، أن اتصال كل برنامج تشغيل جديد بالنظام يتطلب مزيداً من الجهد البرمجي.

على العكس من ذلك، نرى في الشكل تصميماً مختلفاً تكون لجميع برامج التشغيل فيه نفس الواجهة. يصبح وصل برنامج تشغيل جديد الآن أكثر سهولة بكثير، وذلك بفرض أنه متوافق مع واجهة التشغيل الموحدة. ويعني ذلك أيضاً أن مبرمجي برامج التشغيل يعرفون الآن ما المطلوب منهم. (أي، ما هي الوظائف التي يجب أن يقدموها وما هي وظائف النواة التي يمكنهم أن يستدعوها). عملياً، ليست جميع برامج التشغيل متطابقة، لكن يوجد عادة عدد قليل من أنواع الأجهزة، وحتى هذه الأنواع تكون واجهتها متشابهة بشكل عام. مثلاً، حتى الأجهزة الكتلية والأجهزة المحرفية لها العديد من الوظائف المشتركة.

من الجواب الأخرى للواجهة الموحدة كيفية تسمية أجهزة الدخل / الخرج. تهتم البرمجيات المستقلة عن الأجهزة بموضوع ربط أسماء الأجهزة الرمزية ببرامج التشغيل المناسبة. مثلاً، يحدد الاسم /dev/disk0 في UNIX بشكل فريد عقدة من أجل ملف خاص. وتحوي هذه العقدة رقم الجهاز الأساسي، والذي يستخدم لتحديد موقع برنامج التشغيل المناسب. تحوي أيضاً رقم الجهاز الفرعي، والذي يمرر كبارامتر لبرنامج التشغيل لتحديد الوحدة التي سيقرأ منها أو يكتب عليها. جميع الأجهزة لها أرقام أساسية وفرعية، وجميع برامج التشغيل تختار باستخدام رقم الجهاز الأساسي.

ترتبط التسمية بشكل وثيق مع الحماية. كيف يمنع النظام المستخدمين من الوصول إلى أجهزة غير مرخص لهم بالوصول إليها ؟ تظهر الأجهزة في كل من UNIX و Windows 2000 في نظام الملفات على شكل كائنات مسماة، ويعني ذلك أن قواعد الحماية الاعتيادية المطبقة على الملفات تطبق أيضاً على أجهزة الدخل / الخرج حيث يستطيع مدير النظام إعداد السماحيات المناسبة لكل جهاز

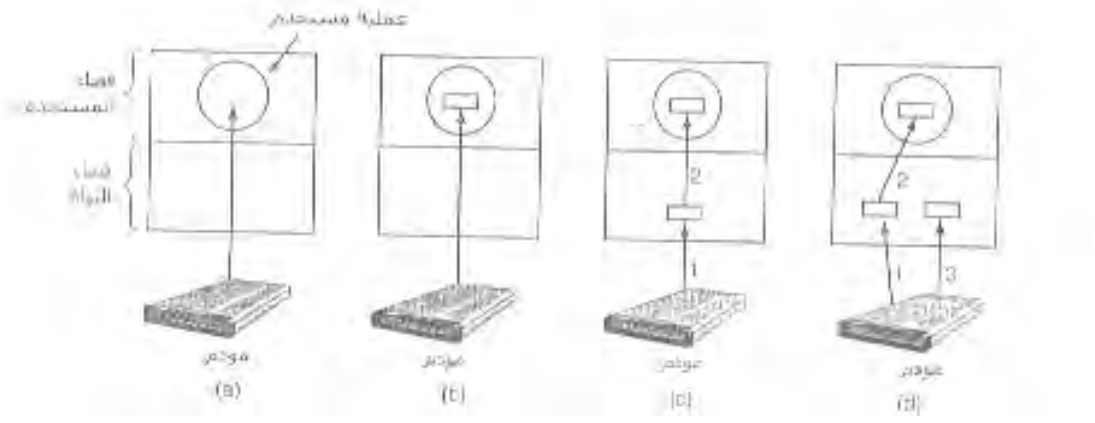


الشكل85

8 3 5 -التخزين المؤقت

يعتبر التخزين المؤقت أيضاً من المواضيع الهامة بالنسبة لكل من الأجهزة الكتلية والأجهزة المحرفية لأسباب متنوعة. لشرح أحدها سندرس عملية تريد قراءة بيانات من مودم. إحدى الإستراتيجيات الممكنة للتعامل مع المحارف الواردة هي جعل عملية المستخدم تقوم باستدعاء النظام Read والتوقف منتظرة من أجل محرف واحد. يسبب كل محرف وارد مقاطعة. يسلّم إجراء خدمة المقاطعة المحرف إلى عملية المستخدم ويستأنف عملها. بعد أن تضع العملية هذا المحرف في مكان ما، تعود إلى قراءة محرف آخر مما يجعلها تتوقف مرة أخرى. هذا النموذج مبين في الشكل.

تتمثل مشكلة هذه الطريقة في إنجاز الأعمال في أي عملية المستخدم يجب أن تستيقظ من أجل كل محرف وارد. إن السماح لعملية بأن تعمل عدة مرات لفترات قصيرة يؤثر على الأداء بشكل سيء، لذلك يعتبر هذا التصميم سيئاً.



الشكل86

يبين الشكل تحسيناً لهذا النموذج. تقدم عملية المستخدم هنا مخزناً مؤقتاً يتسع لـ N محرف في فضاء المستخدم وتقرأ Nمحرف دفعة واحدة. يقوم إجراء خدمة المقاطعة بوضع المحارف الواردة في هذا المخزن حتى يمتلئ. عندها يقوم بإيقاظ عملية المستخدم. تتميز هذه

الطريقة بكفاءة أكبر بكثير من السابقة، لكنها أيضاً تعاني من مشكلة: ماذا يحدث إذا أزيلت الصفحة التي تحوي المخزن المؤقت إلى خارج الذاكرة عند وصول محرف جديد ؟ يمكن قفل المخزن المؤقت ضمن الذاكرة، لكن إذا قامت العديد من العمليات بقفل الصفحات في الذاكرة، فإن مجموعة الصفحات المتوفرة ستتقلص وسينخفض الأداء.

هناك طريقة أخرى تتمثل بإنشاء مخزن مؤقت داخل النواة وجعل إجراء خدمة المقاطعة يضع المحارف فيه، كما في الشكل. عندما يمتلئ هذا المخزن، يتم جلب الصفحة التي تحوي المخزن المؤقت الخاص بالمستخدم إذا تطلب الأمر. وينسخ المخزن من النواة إلى مخزن المستخدم بعملية واحدة. هذه الطريقة أكثر كفاءة بكثير مما سبق.

إلا أنه حتى هذه الطريقة تعاني من مشكلة: ماذا يحدث للمحارف التي تصل أثناء جلب صفحة مخزن المستخدم من القرص ؟ بما أن المخزن ممتلئ، فإنه لا يوجد مكان لوضع هذه المحارف. يكمن الحل في استخدام مخزن مؤقت ثان في النواة. بعد أن يمتلئ المخزن الأول، ولكن قبل تفريغه، يستخدم المخزن الثاني. كما في الشكل

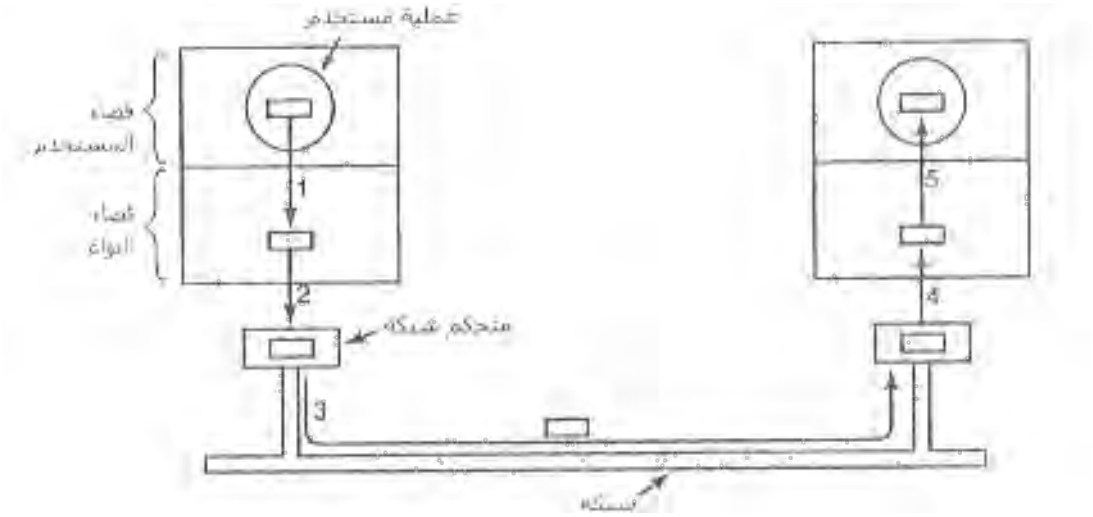
عندما يمتلئ المخزن الثاني، يأتي دوره للنسخ إلى فضاء المستخدم (بفرض أن عملية المستخدم قد طلبت ذلك). أثناء نسخ المخزن الثاني إلى فضاء المستخدم، يمكن استخدام المخزن الأول من أجل المحارف الجديدة. بهذه الطريقة، تتبادل المخازن المؤقتة الأدوار: عندما يكون أحدها ينسخ إلى فضاء المستخدم، يقوم الآخر بتجميع الدخل الجديد. تسمى طريقة التخزين المؤقت هذه بالتخزين المؤقت المزدوج.

التخزين المؤقت هام أيضاً بالنسبة للخرج. لندرس مثلاً كيفية إنجاز الخرج إلى المودم دون تخزين مؤقت باستخدام النموذج المعروض في الشكل. تنفذ عملية المستخدم استدعاء النظام لإخراج محرف. في هذه النقطة يواجه نظام التشغيل خيارين. يستطيع إيقاف المستخدم ريثما تكتب جميع المحارف، لكن هذه الطريقة تستغرق وقتاً طويلاً جداً عند استخدام خط الهاتف البطيء. أو يمكنه تحرير عملية المستخدم مباشرة وينفذ الدخل / الخرج بينما تقوم عملية المستخدم ببعض الحسابات الأخرى. لكن ذلك يؤدي إلى مشكلة أسوأ: كيف ستستطيع عملية المستخدم أن تعرف أن الإخراج قد انتهى وأنها تستطيع إعادة استخدام المخزن المؤقت ؟ يستطيع النظام توليد إشارة أو مقاطعة برمجية، لكن هذا النمط من البرمجة صعب جداً وعرضة لحالات السياق. الحل الأفضل بالنسبة للنواة أن تنسخ البيانات إلى مخزن مؤقت في النواة، وهذا يشبه الشكل لكن بالاتجاه المعاكس. ثم يستأنف عمل العملية المستدعية فوراً. لا يهم الآن متى يحدث الدخل / الخرج الفعلي، لأن عملية المستخدم تستطيع إعادة استخدام مخزنها المؤقت مباشرة بعد استئناف عملها.

التخزين المؤقت من التقنيات الواسعة الاستخدام لكنها لها أيضاً جانب سلبي. إذا خزنت المعلومات عدداً كبيراً من المرات، فإن الأداء سينخفض بشكل كبير. لندرس مثلاً الشبكة المبينة في الشكل. تنفذ عملية المستخدم هنا استدعاء نظام للكتابة إلى الشبكة. تنسخ النواة الرزمة إلى مخزن مؤقت في النواة للسماح للمستخدم بمتابعة العمل مباشرة (الخطوة 1).

عندما يستدعي برنامج التشغيل، يقوم بنسخ الرزمة إلى المتحكم من أجل إخراجها (الخطوة 2). السبب في عدم إخراج البيانات مباشرة إلى السلك من ذاكرة النواة أنه حالما يبدأ إرسال الرزمة، فإنه يجب أن يتابع بسرعة ثابتة. لا يستطيع برنامج التشغيل ضمان الوصول إلى الذاكرة بسرعة ثابتة، لأن قنوات DMA وأجهزة الدخل / الخرج الأخرى قد تسرق العديد من

دورات الممر. إن الفشل في الحصول على كلمة بيانات في الوقت المناسب سيدمر الرزمة. يتم تجنب هذه المشكلة بنسخ الرزمة داخل المتحكم.



الشكل87

بعد نسخ الرزمة إلى مخزن المتحكم الداخلي، يتم نسخها إلى الشبكة (الخطوة 3). بعد فترة قصيرة، تصل البتات إلى المستقبل، أي بعد إرسال البت الأخير بقليل، يصل إلى المستقبل، حيث تخزن الرزمة بأكملها ضمن مخزن المتحكم. تنسخ الرزمة بعد ذلك إلى المخزن المؤقت الموجود في النواة في جانب المستقبل (الخطوة 4). أخيراً، تنسخ الرزمة إلى مخزن العملية المستقبلة (الخطوة 5). عادة، يقوم المستقبل بإرسال إشعار. عندما يحصل المرسل على الإشعار، يستطيع البدء بإرسال الرزمة التالية. من الواضح أن كل هذا النسخ يؤدي إلى إبطاء معدل النقل بشكل ملموس لأن جميع هذه الخطوات يجب تنفيذها بشكل تسلسلي.

8 3 6 -الإعلان عن الأخطاء

إن كثافة الأخطاء في سياق الدخل / الخرج أكبر بكثير منها في أي سياق آخر. عندما تحدث الأخطاء، يجب أن يعالجها نظام التشغيل بأفضل شكل ممكن. العديد من الأخطاء تتعلق بالجهاز ويجب معالجتها في برنامج التشغيل الموافق، لكن إطار العمل الخاص بمعالجة الأخطاء مستقل عن الأجهزة.

من أصناف أخطاء الدخل /الخرج البرمجية. تحدث هذه الأخطاء عندما تطلب عملية مستخدم شيئاً مستحيلاً، مثل الكتابة إلى جهاز دخل (لوحة المفاتيح أو الفأرة أو الماسحة.. الخ) أو القراءة من جهاز خرج (الطابعة، الراسمة.. الخ). تتضمن الأخطاء الأخرى إعطاء عنوان غير صالح للمخزن المؤقت أو تزويد بارامترات أخرى غير صالحة، وتحديد جهاز غير صالح (مثلاً القرص 3 عندما يكون النظام يحوي قرصين فقط). الإجراء المناسب لهذه الأخطاء بسيط جداً: أعد ببساطة شفرة خطأ إلى المستدعي.

صنف آخر من الأخطاء يتمثل بأخطاء الدخل / الخرج الفعلية. مثال على ذلك محاولة الكتابة على كتلة قرص تالفة أو محاولة القراءة من كاميرا مطفأة. في هذه الظروف، يعود لبرنامج التشغيل مهمة تحديد ماذا سيفعل. إذا لم يكن برنامج التشغيل يعرف ماذا سيفعل، يمكنه تمرير المشكلة إلى الأعلى إلى البرمجيات المستقلة عن الجهاز.

يعتمد ما ستفعله هذه البرمجيات على البيئة وطبيعة الخطأ. إذا كان الخطأ عبارة عن خطأ قراءة بسيط وكان يوجد مستخدم تفاعلي، قد يعرض مربع حوار يسأل المستخدم ماذا يفعل. تتضمن الخيارات المتاحة إعادة المحاولة عدداً من المرات أو تجاهل الخطأ أو قتل العملية المستدعية. إذا لم يكن هناك مستخدم، يكون الخيار الوحيد المتوفر هو إعادة شفرة خطأ مع إنهاء استدعاء النظام.

إلا أن بعض الأخطاء لا يمكن معالجتها بهذه الطريقة. مثلاً، إذا تم تدمير بنية بيانات حساسة مثل الفهرس الرئيسي أو لائحة الكتل الحرة، فإن النظام في هذه الحالة يجب أن يعر رسالة خطأ وينتهي.

8 3 7 -حجز وتحرير الأجهزة المخصصة

يمكن استخدام بعض الأجهزة مثل مسجلات الأقراص من قبل عملية واحدة فقط في نفس الوقت. تعود إلى نظام التشغيل مهمة اختبار طلبات استخدام مثل هذه الأجهزة وقبولها أو رفضها، اعتماداً على توفر الجهاز المطلوب. الطريقة السهلة لمعالجة هذه الطلبات إجبار العمليات على إجراء عملية open على الملفات الخاصة لهذه الأجهزة مباشرة. إذا لم يكن الجهاز متوفراً، تفشل عملية open. يؤدي إغلاق مثل هذا الجهاز المخصص إلى تحريره.

هناك طريقة أخرى تتلخص بوجود آليات خاصة لطلب الأجهزة المخصصة وتحريرها. تؤدي محاولة طلب جهاز غير متوفر إلى إعاقة المستدعي عوضاً عن الفشل. توضع العمليات المعاقة ضمن رتل. عاجلاً أم آجلاً، سيصبح الجهاز متوفراً وعندها يسمح للعملية الأولى في الرتل بالحصول على الجهاز وتستأنف تنفيذها.

8 3 8 -أحجام الكتل المستقلة عن الجهاز

قد يكون للأقراص المختلفة أحجام قطاعات مختلفة. يجب على البرمجيات المستقلة عن الأجهزة إخفاء هذه الحقيقة وتقديم حجم كتلة موحد للطبقات العليا، وذلك مثلاً باعتبار كل مجموعة قطاعات ككتلة منطقية واحدة. بهذه الطريقة تتعامل الطبقات العليا فقط مع أجهزة مجردة تستخدم كلها نفس حجم الكتلة المنطقي، بشكل مستقل عن حجم القطاع الفيزيائي. بشكل مشابه، تقدم بعض الأجهزة المحرفية بياناتها بايتاً تلو الآخر (مثل المودمات)، بينما تقدم أجهزة أخرى بياناتها على شكل وحدات كبيرة (مثل بطاقات الشبكات). يمكن أيضاً إخفاء هذه الاختلافات.

8 3 9 -برمجيات الدخل / الخرج في فضاء المستخدم

على الرغم من أن معظم برمجيات الدخل / الخرج موجودة ضمن نظام التشغيل، فإن جزءاً صغيراً منها مؤلف من مكتبات ترتبط مع برامج المستخدم، وحتى برامج كاملة تعمل خارج النواة. تتكون استدعاءات النظام، بما فيها استدعاءات نظام الدخل / الخرج عادة من إجراءات المكتبات. عندما يحوي برنامج C الاستدعاء التالي:

```
Count=write(fd,buffer,nbytes);
```

فإن إجراء المكتبة سيربط مع البرنامج ويوضع ضمن البرنامج الثنائي الموجود في الذاكرة عند التنفيذ. إن مجموعة جميع إجراءات المكتبة هذه هي بكل تأكيد جزء من نظام الدخل / الخرج.

مع أن معظم هذه الإجراءات لا تقوم إلا بوضع البارامترات في المكان المناسب من أجل استدعاء النظام، إلا أن بعضها تقوم بعمل حقيقي. تحديداً، يتم تنسيق الدخل والخرج من قبل إجراءات المكتبة. من أمثلة ذلك الإجراء printf في لغة C، والذي يأخذ سلسلة تنسيق وبعض المتحولات كدخل، ثم يبني سلسلة أسكي ويستدعي Write لإخراج السلسلة. كمثال عنprintf لنأخذ العبارة:

```
Printf("the square of %3d is %6d/n",I,i*I);
```

تقوم العبارة بتنسيق سلسلة مؤلفة من سلسلة فيها 14 محرفاً "the square of" متبوعة بقيمة i على شكل سلسلة من 3 محارف، ثم أربعة محارف "is"، ثم مربع i كستة محارف، وأخيراً رمز تغذية سطر جديد.

هناك إجراء دخل مشابه هو الذي يقرأ الدخل ويخزنه في متحولات موصوفة بسلسلة تنسيق باستخدام نفس قواعد تحوي مكتبة الدخل / الخرج القياسي عدداً من الإجراءات التي تتضمن عمليات دخل / خرج وتعمل كلها كجزء من برنامج المستخدم.

لا تتألف جميع برمجيات الدخل / الخرج في مستوى المستخدم من إجراءات مكتبة. هناك صنف آخر منها هو نظام التوزيع spooling. التوزيع spooling عبارة عن طريقة للتعامل مع الأجهزة المخصصة في الأنظمة ذات البرمجة المتعددة. لندرس جهازاً موزعاً نموذجياً وهو الطابعة. صحيح من السهل تقنياً السماح لأي عملية مستخدم بفتح الملف الخاص المحرفي للطابعة، ولكن ماذا سيحصل إذا فتحت إحدى العمليات الطابعة ولم تفعل شيئاً لساعات. لن تستطيع أي عملية أخرى طباعة أي شيء.

عوضاً عن ذلك، يتم إنشاء عملية خاصة تسمى بالبرنامج الخفي أو العفريت Deamonوفهرس خاص اسمه فهرس التوزيع. لطباعة ملف، تقوم العملية أولاً بتوليد الملف المراد طباعته كاملاً وتضعه في فهرس التوزيع. يقوم البرنامج الخفي الآن بطباعة الملفات الموجودة في الفهرس إلى الطابعة، وهي العملية الوحيدة التي تستطيع الوصول إلى الملف الخاص بالطابعة. إن حماية الملف الخاص من فتحه من قبل المستخدمين مباشرة ينهي مشكلة فتحه لفترات طويلة دون فائدة.

لا يستخدم التوزيع فقط في الطابعات، بل يستخدم أيضاً في حالات أخرى. مثلاً، يتم نقل الملفات عبر شبكة عادة باستخدام برنامج خفي للشبكة. لإرسال ملف إلى مكان ما، يضع المستخدم الملف في فهرس توزيع الشبكة. بعد قليل، يقوم برنامج الشبكة الخفي بأخذه وإرساله عبر الشبكة. أحد الاستخدامات العملية لإرسال الملفات باستخدام التوزيع نظام أخبارUSENET. تتألف هذه الشبكة من ملايين الحواسب الموزعة حول العالم والتي تتصل بالإنترنت. توجد آلاف من مجموعات الأخبار في العديد من المواضيع. لإرسال رسالة أخبار، يقوم المستخدم بتشغيل برنامج أخبار، والذي يقبل الرسالة المراد إرسالها ويضعها في فهرس التوزيع من أجل إرسالها لاحقاً إلى الحواسب الأخرى. يعمل نظام الأخبار بأكمله خارج نظام التشغيل.

يلخص الشكل نظام الدخل / الخرج، ويبين جميع الطبقات والوظائف الأساسية لكل طبقة. ابتداءً من الأسفل، الطبقات هي العتاد الصلب ومعالجات المقاطعات وبرامج التشغيل والبرمجيات المستقلة عن الأجهزة وأخيراً عمليات المستخدم.

تبين الأسهم في الشكل جريان التحكم. عندما يحاول برنامج مستخدم أن يقرأ كتلة من ملف مثلاً، يطلب من نظام التشغيل القيام بالاستدعاء. تبحث البرمجيات المستقلة عن الأجهزة عن الكتلة المطلوبة في الذاكرة المخبئية الخاصة بالمخازن المؤقتة. إذا لم تكن الكتلة المطلوبة موجودة هناك يستدعى برنامج التشغيل لإصدار طلب إلى العتاد للحصول على البيانات من القرص. تتوقف العملية حتى تنتهي عملية القرص. عندما ينتهي القرص، يولّد العتاد مقاطعة.

يتمثل العمل الأساسي لبرنامج تشغيل لوحة المفاتيح بجمع الدخل من لوحة المفاتيح وتمريره إلى برامج المستخدم عندما تقرأ من الطرفية. هذه الفلسفة ملائمة جداً لحاجات برامج تحرير النصوص المعقدة مثل، والتي تسمح للمستخدم بربط عمل معين بأي محرف أو تسلسل من المحارف. لكن ذلك يعني أن المستخدم إذا كتب dste عوضاً date عن ثم صحح الخطأ بالضغط على السهم الخلفي ثلاث مرات ثم كتب ate متبوعة برمز العودة إلى أول السطر. فإن برنامج المستخدم سيحصل على 11 شفرة هي:

Dste <- <- <- ate CR

لا تحتاج جميع البرامج إلى مثل هذه التفاصيل. وعادة تحتاج البرامج للدخل المصحح فقط، وليس تسلسل المحارف الذي أدى إلى توليده. أدت هذه الملاحظة إلى فلسفة ثانية: يعالج برنامج التشغيل جميع عمليات التحرير ضمن السطر الواحد ويرسل إلى برنامج المستخدم فقط الأسطر المصححة. تسمى الفلسفة الأولى بالمحرفية والثانية بالسطرية . وكانتا تسميان في الأصل النمط النيئ والنمط المطبوخ. يستخدم معيار تسميتين أكثر غرابة وهما النمط القانوني من أجل النمط السطري والنمط غير القانوني للنمط المحرفي. مع أن هذه الأنماط مختلفة في الكثير من التفاصيل عن الأنماط السابقة الذكر. تقدم الأنظمة المتوافقة مع عدة مكتبات تدعم اختيار أحد النمطين وتغيير العديد من صفات الطرفية.

المهمة الأولى لبرنامج تشغيل لوحة المفاتيح هي جمع الدخل. إذا كانت كل ضغطة مفتاح تولد مقاطعة، يستطيع برنامج التشغيل الحصول على المحرف أثناء معالجة المقاطعة. وإذا حوّلت المقاطعات إلى رسائل من قبل البرمجيات المنخفضة المستوى، فإنه من الممكن وضع المحارف الواردة حديثاً ضمن الرسالة. ويمكن بطريقة أخرى وضع المحارف في مخزن مؤقت صغير واستخدام الرسالة لإخبار برنامج التشغيل أن شيئاً قد وصل. الطريقة الأخيرة أسلم في الواقع إذا كانت الرسائل لا يمكن إرسالها إلا إلى عمليات منتظرة وهناك احتمال أن يكون برنامج التشغيل مشغولاً بمعالجة المحرف السابق.

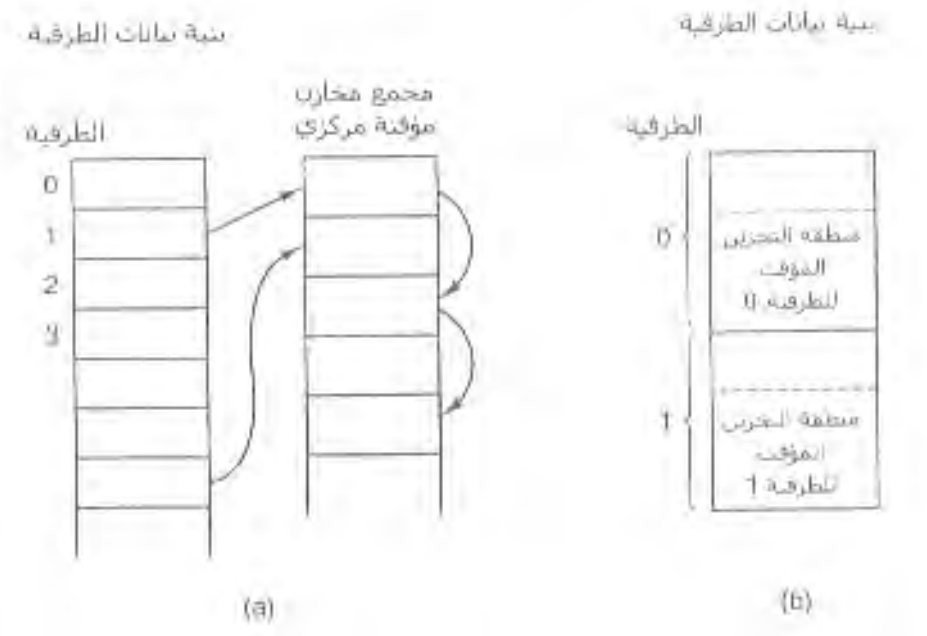
إذا كانت الطرفية في النمط السطري ، فإن المحارف يجب أن تخزن حتى تجميع السطر بأكمله، لأن المستخدم قد يقرر فيما بعد أن يسمح قسماً منه. وحتى إذا كانت الطرفية في النمط المحرفي، فإن البرنامج قد لا يكون قد طلب إدخال شيء. عندها يجب أن تخزن هذه المحارف في مخزن مؤقت للسماح بالمزيد من الكتابة.

هناك طريقتان شائعتان للتخزين المؤقت للمحارف. يحوي برنامج التشغيل في الطريقة الأولى مجمعاً مركزياً للمخازن المؤقتة، كل مخزن يستطيع تخزين10 محارف مثلاً. ويخصص لكل طرفية بنية بيانات تحوي فيما تحويه مؤشراً يشير إلى سلسلة من المخازن المؤقتة للدخل

القادم من هذه الطرفية. كلما كتبت محارف أكثر، تخصص مخازن مؤقتة أكثر وتعلّق على السلسلة. عندما تمرر المحارف إلى برنامج المستخدم، تحرر المخازن المؤقتة وتعاد إلى المجمع المركزي.

تتمثل الطريقة الثانية في إنجاز التخزين المؤقت مباشرة في بنية بيانات الطرفية نفسها، دون وجود مجمع مركزي للمخازن المؤقتة. بما أنه من الشائع أن يكتب المستخدمون أمراً يستغرق تنفيذه بعض الوقت (مثل إعادة ترجمة وربط برنامج كبير) ثم كتابة بضعة أسطر مسبقاً قبل انتهاء الأمر الأول، فإنه يجب على برنامج التشغيل أن يحتاط لذلك ويخصص حوالي 200 محرف لكل طرفية. لكن عند العمل في أنظمة المشاركة الزمنية الكبيرة التي يصل فيها عدد الطرفيات إلى 100 طرفية، فإن حجز 20 كيلو بايت بشكل دائم من أجل الأوامر المكتوبة مسبقاً يعتبر مضيعة للذاكرة، لذلك يعتبر مجمع تخزين مؤقت مركزي بحجم 5 كيلوبايت كافياً حتماً. من ناحية أخرى، إن وجود مخزن مؤقت مستقل لكل طرفية يسهل بشكل كبير تصميم برنامج التشغيل (لا حاجة لإدارة اللوائح المترابطة) ويفضل استخدامه في الحواسب الشخصية التي لها لوحة مفاتيح واحدة. يبين الشكل الفرق بين هاتين الطريقتين.

مع أن لوحة المفاتيح وجهاز العرض هما جهازان منفصلان منطقياً، إلا أن العديد من المستخدمين اعتادوا على رؤية المحارف التي يكتبونها تطبع على الشاشة. كانت بعض الطرفيات القديمة تعرض كل شيء يكتب تلقائياً (بواسطة العتاد). لم يكن ذلك مزعجاً عند كتابة كلمات المرور فحسب، بل كان أيضاً يحد من مرونة محررات النصوص المعقدة وغيرها من البرامج. لحسن الحظ، معظم الطرفيات لا تطبع أي شيء تلقائياً عند الضغط على أي مفتاح. وتعود مهمة عرض المحارف المكتوبة على الشاشة بأكملها إلى البرمجيات الموجودة في الحاسب. وتسمى هذه العملية بالارتداد أو الصدى



الشكل 88

تعتبر عملية الارتداد معقدة لأن البرامج قد تكتب على الشاشة بينما يكون المستخدم يكتب بعض المحارف. يجب على برنامج تشغيل لوحة المفاتيح على الأقل أن يكتشف أين يستطيع كتابة الدخل الجديد دون أن يكتب فوق الخرج الصادر من البرنامج.

تتعقد العملية أكثر عندما نحتاج لعرض أكثر من 80 محرفاً على الشاشة فيها أسطر تحوي 80 محرفاً فقط (أو أي رقم آخر). حسب نوع التطبيق، قد يكون الالتفاف إلى السطر التالي ملائماً. بعض برامج التشغيل تقص الأسطر عند المحرف 80 وتهمل جميع المحارف التي تليه.

مشكلة أخرى هي معالجة الجدولة . يجب على برنامج التشغيل هنا أن يحسب المكان الحالي للمشيرة، آخذاً بعين الاعتبار كلاً من الخرج الصادر عن البرنامج والخرج الناتج عن الارتداد، ثم يحسب عدد الفراغات التي يجب إظهارها.

المشكلة الأهم هي التكافؤ بين الأجهزة المختلفة. منطقياً، نحتاج عند انتهاء كل سطر من النص إلى أن ننتقل إلى بداية السطر التالي. وهذه العملية مؤلفة من الانتقال إلى العمود 1 (إرجاع الحامل)، والانتقال إلى السطر الثاني (تغذية سطر). إن الطلب من المستخدم أن يكتب محرفين عند نهاية كل سطر ليس فكرة محبذة (مع أن بعض الطرفيات فيها مفتاح يولد كلا المحرفين عند نهاية السطر). يجب على برنامج التشغيل أن يحوّل المحارف الواردة من لوحة المفاتيح إلى التنسيق الداخلي القياسي المعتمد في نظام التشغيل.

إذا كان المعيار المعتمد هو تخزين محارف تغذية السطر فقط (كما في)، فإنه يجب تحويل كل محارف إرجاع الحامل إلى تغذية سطر. أما إذا كان التنسيق الداخلي يتطلب تخزين كلا المحرفين (كما في)، يجب على برنامج التشغيل توليد محرف تغذية سطر عندما يحصل على محرف إرجاع حامل وتوليد محرف إرجاع حامل عندما يحصل على محرف تغذية سطر. وبغض النظر عن التنسيق الداخلي، فإن الطرفية تحتاج إلى ارتداد كلا المحرفين كي يكون الإظهار صحيحاً على الشاشة. وبما أن الحواسب الكبيرة قد تتصل بطرفيات من أنواع مختلفة، يجب على برنامج تشغيل لوحة المفاتيح أن يحول جميع تشكيلات إرجاع الحامل / تغذية سطر إلى التنسيق الداخلي القياسي المعتمد وترتيب عملية الارتداد بالشكل الصحيح.

عند العمل في النمط القانوني، يكون لبعض محارف الدخل معنى خاص، يبين الشكل جميع المحارف الخاصة المطلوبة من قبل معيار. جميع التشكيلات الافتراضية عبارة عن أحرف تحكم لا تتعارض مع إدخال النص أو الشفرة المستخدمة من قبل البرامج، لكن يمكن تغييرها بطلب من البرنامج ما عدا المحرفين الأخيرين.

جدول10

المحرف	اسمه في POSIX	المعنى
Ctrl+H	ERASE	مسح محرف واحد إلى الخلف
Ctrl+U	KILL	مسح السطر الذي يكتب بأكمله
Ctrl+V	LNEXT	تفسير المحرف التالي حرفياً
Ctrl+S	STOP	إيقاف الخرج (يقصد بالخرج خرج لوحة المفاتيح)
Ctrl+Q	START	بدء الخرج
DEL	INTR	مقاطعة العملية (SIGINT)
Ctrl+\	QUIT	خروج مع كتابة صورة الذاكرة (SIGQUIT)
Ctrl+D	EOF	نهاية الملف
Ctrl+M	CR	إرجاع الحامل (لا يمكن تغييره)
Ctrl+J	NL	تغذية سطر (لا يمكن تغييره)

يسمح المحرف للمستخدم بمسح المحرف الذي كتبه تواً. وعادة يكون مفتاح الحذف الخلفي backspace. لا يضاف المحرف إلى رتل الإدخال، لكنه يقوم عوضاً عن ذلك بإزالة المحرف السابق من الرتل. يجب إرساله إلى الشاشة على شكل ثلاثة محارف متتالية: حذف خلفي ثم حذف خلفي وذلك لإزالة المحرف السابق من الشاشة. إذا كان المحرف السابق هو، فإن مسحه يعتمد على طريقة معالجته عند كتابته. إذا تم توسيعه مباشرة إلى فراغات، يجب معرفة عدد هذه الفراغات لإزالتها. أما إذا كان مخزناً في رتل الإدخال كما هو، يمكن إزالته ثم إعادة إخراج السطر بأكمله. في معظم الأنظمة، يمسح محرف الحذف الخلفي المحارف الموجودة على السطر الحالي فقط. ولا يسمح محرف إرجاع الحامل ويعود إلى السطر السابق.

عندما يلاحظ المستخدم خطأً في بداية السطر الذي يكتبه، فإن من الأنسب غالباً أن يسمح السطر بأكمله ويعيد كتابته من جديد. يمسح المحرف KILL السطر بأكمله. معظم الأنظمة تخفي السطر الممسوح من الشاشة، لكن بعضها تظهره بالإضافة إلى محرفي إرجاع الحامل والتغذية السطرية لأن بعض المستخدمين يحبون أن يروا السطر القديم. وبالتالي، فإن طريقة عرض المحرف هي مسألة ذوق لا أكثر. وكما هي الحال مع ERASE، فإنه عادة لا يمكن العودة إلى السطر السابق. عند مسح كتلة من المحارف، قد يكون من الأنسب (أو قد لا يكون) أن يعاد المخزن المؤقت إلى المجمع المركزي.

تقدم بعض برامج تشغيل الطرفيات إمكانيات تحرير سطرية أفضل مما تكلمنا عنه هنا. حيث تقدم محارف تحكم خاصة لمسح كلمة وتجاوز كلمة إلى الوراء أو إلى الأمام و الذهاب إلى بداية السطر أو نهاية السطر الحالي أو إدخال نص في وسط السطر، وغيرها. إن إضافة كل هذه الوظائف إلى برنامج التشغيل يجعله أكبر كثيراً، كما أنه يضيع عند استخدام محررات النصوص المعقدة التي تعمل في النمط غير المطبوخ.

8 3 10 -برمجيات الخرج

الخرج أبسط من الدخل. خلاصة الموضوع أن الحاسب يرسل محارف إلى الطرفية حيث يتم إظهارها هناك. عادة، تكتب كتلة من المحارف، سطر مثلاً، إلى الطرفية باستخدام

استدعاء نظام واحد. الطريقة الشائعة الاستخدام بالنسبة لطرفيات RS-232 تتمثل بوجود مخازن مؤقتة للخرج من أجل كل طرفية. يمكن أن تأتي هذه المخازن من نفس المجمع المركزي مثل مخازن الدخل، أو قد تكون مخازن مخصصة كما هي الحال في الدخل. عندما يكتب برنامج إلى الطرفية، يتم نسخ الخرج أولاً إلى مخزن الخرج المؤقت. وكذلك ينسخ الخرج الناتج عن الارتداد إلى المخزن المؤقت. بعد نسخ كل الخرج إلى المخزن المؤقت، يتم إخراج المحرف الأول ثم ينام برنامج التشغيل. عندما تأتي المقاطعة، يرسل المحرف التالي وهكذا.

تحتاج محررات الشاشة والعديد من البرامج المعقدة الأخرى إلى التمكن من تحديث الشاشة بطرق معقدة مثل استبدال سطر في منتصف الشاشة. لتلبية هذه الحاجات، تدعم معظم الطرفيات سلسلة من الأوامر لتحريك المشيرة وحشر وحذف محارف أو أسطر عند المشيرة..الخ. تسمى هذه الأوامر عادة بتسلسلات الهروب. في العصر الذهبي لطرفيان RS 232 -، كان هناك مئات الأنواع من الطرفيات، وكان كل منها يدعم مجموعة خاصة به من تسلسلات الهروب، وبالتالي، كان من الصعب كتابة برمجيات تعمل على أكثر من نوع طرفيات واحد.

قدم نظام باركلي يونيكس حلاً لهذه المشكلة يتمثل بقاعدة بيانات طرفيات سميت Termcap. عرفت رزمة البرمجيات هذه عدداً من الأفعال الأساسية، مثل تحريك المشيرة إلى (سطر، عمود). لتحريك المشيرة، يستخدم برنامج ما، ليكن محرر نصوص، تسلسل هروب عام والذي يحّول إلى تسلسل الهروب الفعلي الملائم للطرفية المراد الكتابة إليها. بهذه الطريقة، يمكن تشغيل المحرر على أية طرفية لها سجل في قاعدة بيانات Termcap.شيئاً فشيئاً برزت الحاجة لوجود معيار لتسلسلات الهروب، لذلك تم تطوير معيار يبين الشكل بعض القيم المعيارية.

لندرس كيف يمكن استخدام تسلسلات الهروب هذه من قبل محرر نصوص. افترض أن المستخدم كتب أمراً يطلب من المحرر أن يحذف كامل السطر 3 ويزيل الفراغ بين السطرين 2 و 4. يرسل المحرر تسلسل الهروب التالي على الخط التسلسلي إلى الطرفية:

ESC [3;1 H ESC [0 K ESC [1M

(استخدمنا الفراغات هنا فقط لفصل الرموز، لكنها لا ترسل). ينقل هذا التسلسل المشيرة إلى بداية السطر 3، ثم يمسح السطر بأكمله ثم يحذف السطر الذي أصبح الآن فارغاً مما يؤدي إلى نقل جميع الأسطر ابتداءً من 5 سطراً واحداً إلى الأعلى. عندها يصبح ما كان السطر 4 السطر 3 وما كان السطر 5 يصبح السطر 4 وهكذا.يمكن استخدام تسلسل هروب مشابه لإضافة نص إلى منتصف الشاشة. يمكن أيضاً إضافة الكلمات أو إزالتها بطريقة مشابهة.

جدول11

تسلسل الهروب	المعنى
ESC [n A	الذهاب إلى الأعلى n سطراً.
ESC [n B	الذهاب إلى الأسفل n سطراً.
ESC [n C	الذهاب إلى اليمين n فراغاً.
ESC [n D	الذهاب إلى اليسار n فراغاً
ESC [m ; n H	نقل المشيرة إلى الموقع (m,n).
ESC [s J	مسح الشاشة من موقع المشيرة (حسب قيمة s: 0 إلى النهاية، 1 من البداية، 2 الكل).
ESC [s K	مسح سطر من موقع المشيرة (حسب قيمة s: 0 إلى النهاية، 1 من البداية، 2 الكل).
ESC [n L	حشر n سطراً عند المشيرة.
ESC [n M	حذف n سطراً من عند المشيرة.
ESC [n P	حذف n محرف من عند المشيرة.
ESC [n @	إدخال n محرف عند المشيرة.
ESC [n m	تفعيل المميز (حسب قيمة n: 0 طبيعي، 1 عريض، 5 وميض، 7 معكوس).
ESC M	دحرجة الشاشة إلى الخلف إذا كانت المشيرة في السطر العلوي.

8 3 11- واجهات المستخدم الرسومية

تستطيع الحواسب الشخصية استخدام واجهات محرفية، وفي الواقع، سيطر نظام التشغيل، وهو نظام محرفي، على سوق الحواسب الشخصية لسنوات عديدة، لكن معظم الحواسب الشخصية هذه الأيام تستخدم واجهة مستخدم رسومية.

تعتمد على أربعة عناصر أساسية يشار إليها بالأحرف تدل هذه الأحرف على النوافذ والرموز والقوائم وجهاز التأشير على الترتيب. النوافذ عبارة عن كتل مستطيلة من مساحة الشاشة تستخدم لتشغيل البرامج. الرموز عبارة عن رسوم صغيرة يمكن النقر عليها لتنفيذ عمل ما. القوائم عبارة عن لوائح من الأوامر التي يمكن الاختيار منها. أخيراً، جهاز التأشير هو الفأرة أو كرة التتبع أو أي جهاز عتادي آخر يستخدم لتحريك المؤشر عبر الشاشة لاختيار الأشياء.

يمكن إنجاز برمجيات GUI إما في شفرة مستوى المستخدم، كما في أنظمة يونكس أو ضمن النظام نفسه كما هي الحالة في وندوز. سنتكلم في الأقسام اللاحقة عن العتاد وبرمجيات الإدخال وبرمجيات الإخراج المرتبطة بواجهات GUI الخاصة بالحواسب الشخصية مع التركيز على، لكن الأفكار العامة تنطبق أيضاً على GUIالأخرى.

8 3 12- عتاد لوحة المفاتيح والفأرة وجهاز العرض في الحاسب الشخصي

جميع الحواسب الشخصية الحديثة فيها لوحة مفاتيح وجهاز عرض معنون كذاكرة يعمل على مستوى البتات. هذان المكونان عبارة عن جزأين أساسيين من الحاسب نفسه. إلا أن لوحة المفاتيح والشاشة منفصلان تماماً عن بعضهما، ولكل منهما برنامج التشغيل الخاص به.

يمكن وصل لوحة المفاتيح عبر المنفذ التسلسلي أو التفرعي أو USB. عند كل ضربة مفتاح، تتم مقاطعة المعالج، ويقوم برنامج تشغيل لوحة المفاتيح بالحصول على المحرف المضغوط بالقراءة من منفذ دخل / خرج. يحدث كل شيء آخر في البرمجيات ومعظمه في برنامج تشغيل لوحة المفاتيح.

يولد أيضاً المحرف A الكبير مع أن واجهة لوحة المفاتيح هذه تلقي حملاً كبيراً على البرمجيات، إلا أنها تتمتع بمرونة كبيرة. مثلاً، قد يكون أحد برامج المستخدم يريد أن يعرف إذا ما كان الرقم المكتوب آتياً من الصف العلوي من المفاتيح أم أنه من لوحة المفاتيح الرقمية الجانبية. مبدئياً، يستطيع برنامج التشغيل تقديم هذه المعلومات.

معظم الحواسب الشخصية فيها فأرة أو كرة تتبع، وهي عبارة عن مجرد فأرة مستلقية على ظهرها. يحوي النوع الأكثر انتشاراً من الفئران كرة مطاطية تبرز قليلاً من فجوة موجودة في الأسفل وتدور عند تحريك الفأرة على سطح خشن. يؤدي دوران الكرة إلى تدوير قرصين مثقبين مثبتين على محورين متعامدين. تؤدي الحركة إلى اليمين واليسار إلى تدوير المحور الموازي المحور Y وتؤدي الحركة إلى الأسفل والأعلى إلى تدوير المحور الموازي لمحورX. كلما تحركت الفأرة مسافة دنيا معينة في أي اتجاه أو عند الضغط على أحد أزرار الفأرة، يتم إرسال رسالة إلى الحاسب. تبلغ المسافة الدنيا حوالي 0.1 ملم (على الرغم من إمكانية تحديدها برمجياً). يمكن أن تحوي الفأرة زراً واحداً أو اثنين أو ثلاثة حسب تقدير المصمم لقدرات المستخدم العقلية من أجل تتبع أكثر من زر واحد.

تتألف الرسالة المرسلة إلى الحاسب من ثلاث معلومات: تغير محور x و تغير محور y والأزرار. المعلومة الأولى هي مقدار التغير في موقع x منذ الرسالة الأخيرة. والثانية هي تغير موقع y منذ الرسالة الأخيرة.

يحوي الجزء الأخير حالة أزرار الفأرة. يعتمد تنسيق الرسالة على النظام وعدد الأزرار الموجودة في الفأرة. وعادة تتكون من 3 بايت. تعطي معظم الفئران رسائل بتردد يصل إلى 40 رسالة / ثانية، لذلك قد تكون الفأرة قد تحركت أكثر من واحدة حركية منذ الرسالة الأخيرة.

لاحظ أن الفأرة لا تقدم الموقع المطلق لها، بل تعطي فقط تغيرات في الموقع. إذا رفعت الفأرة ثم وضعت في مكان آخر بلطف، لا يتم إرسال أي رسالة.

تميز بعض بين النقرة الواحدة والنقرة المزدوجة لأزرار الفأرة. إذا كانت النقرتان قريتين بشكل كاف بالمسافة وبالزمن أيضاً تعتبران كنقرة مزدوجة. يتم تحديد معيار التقارب بين النقرات برمجياً، مع إمكانية تحديد كل من البارامترين من قبل المستخدم.

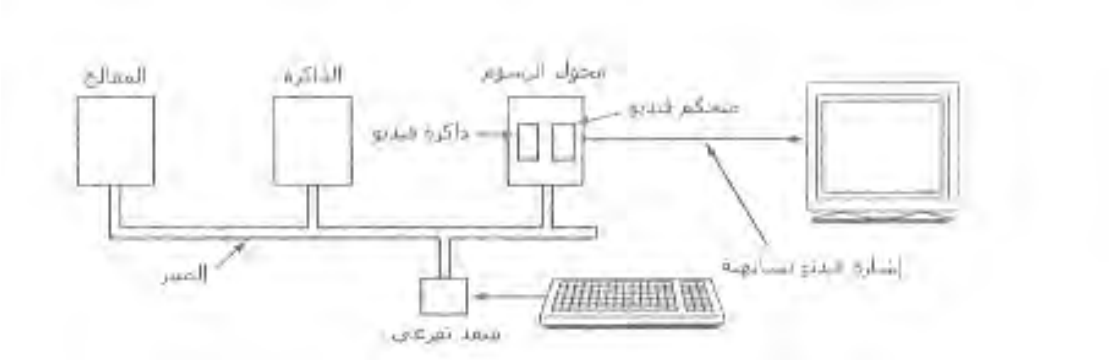
يكفي ما قلناه عن عتاد الإدخال، وسننتقل الآن إلى عتاد جهاز العرض. يمكن تقسيم أجهزة العرض إلى صنفين اثنين. يقبل الصنف الأول، أجهزة الرسوم الشعاعية، أوامر مثل رسم نقاط وخطوط وأشكال هندية ونصوص. أما الصنف الثاني، وهو أجهزة المسح، فإنه يمثل منطقة الإخراج على شكل شبكة مستطيلة من النقاط التي تدعى العنصورات والتي يملك كل منها تدرجاً

رمادياً أو لوناً معيناً. كانت أجهزة الرسوم الشعاعية في بدايات عصر الحاسب شائعة جداً، لكن الراسمات الآن هي أجهزة الرسوم الشعاعية الوحيدة المتوفرة. تستخدم جميع الأجهزة الأخرى الرسومات المسحية والتي تدعى أحياناً الرسوم النقطية .

تحقق أجهزة الرسوم الشعاعية بواسطة عتاد يسمى محول رسوم الفيديو. يحوي محول الرسوم ذاكرة خاصة تدعى ذاكرة الفيديو أو ذاكرة الإظهار، والتي تشكل جزءاً من فضاء عناوين الحاسب وتعنون من قبل المعالج بنفس طريقة عنونة بقية الذاكرة (الشكل). تخزن صورة الشاشة إما بالنمط المحرفي أو النمط النقطي. في النمط المحرفي يحوي كل بايت (أو 2 بايت) من ذاكرة الفيديو محرفاً واحداً. أما في النمط النقطي، يتم تمثيل كل بكسل على الشاشة بشكل منفصل من ذاكرة الفيديو، حيث يخصص 1 بت لكل عنصورة في حالة العرض باللونين الأبيض والأسود، وهي الحالة الأبسط، وقد يخصص 24 بتاً أو أكثر لكل عنصورة من أجل العرض بألوان عالية النوعية.

يحوي محول الرسوم أيضاً شريحة تسمى متحكم الفيديو. تقرأ هذه الشريحة المحارف أو النقاط من ذاكرة الفيديو وتحولها إلى إشارات الفيديو التي تشغل الشاشة. تولد الشاشة شعاعاً إلكترونياً يمسح الشاشة أفقياً ويرسم خطوطاً عليها. تحوي الشاشة نموذجياً من 480 حتى 1024 خط مسح يتألف كل منها من 640 حتى 1200 عنصورة. تعدّل الإشارات الصادرة عن متحكم الفيديو الشعاع الإلكتروني وتحدد إذا ما كانت عنصورة معينة فاتحة أو داكنة. تحوي الشاشات الملونة ثلاثة أشعة الكترونية من أجل الألوان الأحمر والأخضر والأزرق، حيث تعدّل بشكل منفصل. تستخدم أجهزة العرض ذات السطح المستوي أيضاً عنصورات لها ثلاثة ألوان، لكن مبدأ عمل هذه الأجهزة خارج عن موضوع هذا الكتاب.

هناك نمطان لمتحكم الفيديو: النمط المحرفي (يستخدم للنصوص البسيطة) والنمط النقطي (يستخدم للأشياء الأخرى). في النمط المحرفي، قد يضع المتحكم كل محرف في مربع عرضه 9 عنصورات وارتفاعه 14 عنصورة (بما في ذلك الفراغ بين المحارف)، ويقسم الشاشة إلى 25 سطراً في كل منها 80 محرفاً. يحوي جهاز العرض في هذه الحالة 350 خطاً مسح في كل منها 720 عنصورة. يرسم كل إطار من 60 إلى 100 مرة كل ثانية لتجنب الوميض.



الشكل 89

لعرض النص على الشاشة، يمكن أن يحضر متحكم الفيديو أول 80 محرفاً من ذاكرة الفيديو ويولد 14 خط مسح، ثم يحضر 80 محرفاً التالية ويولد 14 خط مسح التالية، وهكذا. ويمكن بطريقة أخرى إحضار كل محرف واحدة لكل خط مسح للاستغناء عن المخزن المؤقت

في المتحكم. تحفظ أشكال الأحرف ذات الحجم 9 × 14 بت في ذاكرة تستخدم من قبل متحكم الفيديو. (يمكن أيضاً استخدام الذاكرة لدعم خطوط مخصصة). تعنون بعناوين البتات الثمانية الموجودة في كل بايت من ذاكرة العنصورات الثمانية الموجودة في سطر المسح الموافق، أما العنصورة التاسعة فهي فارغة دائماً. وبالتالي يحتاج كل سطر نصي إلى الرجوع إلى ذاكرة الفيديو مرة كي يعرض. ويرجع إلى ذاكرة الخاصة بتوليد المحارف بنفس العدد من المرات.

نرى في الشكل جزءاً من ذاكرة الفيديو لجهاز عرض يعمل بالنمط المحرفي. يحتل كل محرف على الشاشة في الشكل بايتين من ذاكرة. البايت الأدنى هو شفرة أسكي للمحرف المراد عرضه. أما البايت الأعلى فهو بايت السمة، والذي يستخدم لتحديد اللون والوميض..الخ تحتاج شاشة فيها محرف إلى 4000 بايت من ذاكرة الفيديو في هذا النمط.

يستخدم نفس المبدأ في حالة العرض بالنمط النقطي، ما عدا أن كل عنصورة في الشاشة يتم التحكم بها منفردة، وتمثل منفردة ببت واحد أو أكثر في ذاكرة الإظهار. التشكيل الأبسط هو حالة العرض أحادي اللون. حيث تمثل كل عنصورة في الشاشة ببت واحد في ذاكرة الإظهار. أما التشغيل الأعقد، فتمثل كل عنصورة فيه برقم طوله 24 بتاً في ذاكرة الإظهار، حيث يخصص 8 بتات لكل من الألوان الأحمر والأخضر والأزرق. يستخدم تمثيل RGB(الأحمر والأخضر والأزرق) لأن هذه الألوان هي الألوان الجمعية الأساسية، والتي يمكن اشتقاق جميع الألوان منها من خلال جمعها مع بعضها بشدائد مختلفة.

لتجنب التعامل مع هذه الأحجام الكبيرة للصور، تضحي بعض الأنظمة بدقة الألوان من أجل تقليل حجم الصورة. وتتمثل الطريقة الأبسط في تمثيل كل عنصورة برقم 8 بت. لا يمثل هذا الرقم لوناً بل دليلاً ضمن جدول مؤلف من 256 سجل يحوي كل منها قيمة بطول 24 بت (أحمر، أخضر، أزرق). يسمى هذا الجدول بلوحة الألوان ويخزن غالباً في العتاد ويسمح للشاشة باحتواء أي 256 لوناً في أي لحظة. يؤدي تغير السجل 7 مثلاً في الجدول إلى تغير لون جميع العنصورات في الصورة التي تحمل القيمة 7. إن استخدام لوحة ألوان 8 بت يقلل المساحة المطلوبة لتخزين صورة الشاشة من 3 بايت لكل عنصورة إلى 1 بايت لكل عنصورة. وثمن ذلك هو دقة ألوان أخشن. يعمل نظام ضغط الصور مع لوحة ألوان مماثلة.

يمكن أيضاً استخدام لوحة ألوان ذات 16 بتاً لكل عنصورة. في هذه الحالة، تحوي لوحة الألوان سجلاً، لذلك يمكن استخدام لوناً دفعة واحدة. إلا أن توفير المساحة أقل من الطريقة السابقة لأن كل عنصورة تحتاج هنا إلى 2 بايت في ذاكرة الإظهار. كما أنه إذا خزنت لوحة الألوان في العتاد (لتجنب البحث الكثيف في الذاكرة عند كل عنصورة)، فإننا نحتاج إلى عتاد أكثر لتخزين لوحة الألوان الكبيرة.

يمكن أيضاً معالجة ألوان ذات 16 بتاً بتخزين قيم RGB كثلاث قيم بطول 5 بت مع ترك بت واحد (أو إعطاء 6 بتات للأخضر بما أن العين البشرية أكثر حساسية للون الأخضر من الأحمر أو الأزرق). في الواقع، هذا يشبه ألوان 24 بت، إلا أنه يحوي تدرجات أقل من كل لون.

9 -مقاييس التوقيت"Timing Measurements"

مُقدِّمة

إن العديد من النشاطات الحاسبية تقاد بمقاييس التوقيت, غالبا في خلفية المستخدم.على سبيل المثال, إذا أوقفت الشاشة تلقائيا بعد أن توقفت عن استخدام حاسبك, سيكون ذلك بفضل المؤقت الذي يسمح للنواة بأن تحفظ كم من الوقت قد مر ولم تضغط على زر أو تحرك الفأرة. إذا تلقيت إنذارا من جهازك بضرورة حذف بعض الملفات غير المستخدمة , يكون ذلك بنتيجة برنامج يحدد جميع ملفات المستخدم التي لم يتم الدخول إليها منذ وقت طويل،من أجل القيام بذلك على البرامج أن تأخذ من الملفات ما يسمى بختم الوقت time stamp محددة آخر زمن تم الدخول فيه إلى الملف , لذلك لا بد للنواة أن تكتب تلقائيا الختم الزمني timestamp.

بإمكاننا أن نميز بين نوعين من مقاييس التوقيت في النواة:

- الحفاظ على الوقت والتاريخ الحاليين.
- تعيين المؤقتات , والتي هي آلات قادرة على تنبيه النواة أو برنامج المستخدم أنه قد تم مرور فترة معينة من الزمن.

تجدر الإشارة أنه تنجز مقاييس التوقيت بواسطة العديد من دارات الكيان الصلب المعتمدة على عدادات ثابتة التردد وتعتبر المؤقتات أساسية جداً لعمل أي نظام متعدد البرمجة لعدة أسباب. نذكر من هذه الأسباب أنها تحتفظ بتوقيت الساعة وتمنع العمليات من احتكار المعالج. قد يأخذ المؤقت أحياناً شكل برنامج تشغيل جهاز، مع أن المؤقت ليس جهازاً كتلياً مثل القرص ولا جهازاً محرفياً مثل الفأرة.

9 .1 -عتاد المؤقتات

هناك نوعان من المؤقتات يشيع استخدامهما في الحواسب، وكلاهما مختلفان عن المؤقتات والساعات المستخدمة من قبل الناس.يرتبط الشكل الأبسط من المؤقتات مع خط التغذية أو فولت، ويعطي مقاطعة عند كل دورة جهد بتردد أو هرتز. كانت هذه المؤقتات شائعة جداً لكنها نادرة هذه الأيام.

يتألف النوع الآخر من المؤقتات من ثلاثة عناصر: هزاز كريستالي وعداد ومسجل ماسك، عندما تقطع قطعة من الكوارتز المتبلور بشكل مناسب وتوضع تحت ضغط معين، يمكن جعلها تولد إشارة دورية بتردد دقيق جداً، ويكون عادة في مجال عدة مئات من الميغاهرتز حسب قطعة الكريستال المختارة.

يمكن باستخدام دارات الكترونية ضرب هذا التردد بعامل صحيح صغير للحصول على ترددات تصل إلى أو أكثر. توجد واحدة من هذه الدارات على الأقل في كل حاسب كي تعطي

إشارة التزامن لدارات الحاسب المختلفة. تغذى هذه النبضات إلى العداد لجعله يعد هابطاً إلى الصفر. عندما يصل العداد إلى الصفر، فإنه يولد مقاطعة للمعالج.

تملك المؤقتات المبرمجة نموذجياً عدة أنماط عمل. لنأخذ مثلاً نمط الطلقة الوحيدة فعندما يبدأ الوقت ينسخ قيمة المسجذل الماسك إلى العداد ثم ينقص العدد عند كل نبضة من الهزاز الكريستالي. عندما يصل العداد إلى الصفر، يولد مقاطعة ويتوقف حتى يتم بدؤه من جديد صراحة من قبل البرنامج. في نمط الموجة المربعة، بعد الوصول إلى الصفر وتوليد المقاطعة يتم نسخ المسجل الماسك تلقائياً إلى العداد، وتكرر العملية كلها بشكل غير نهائي. تسمى هذه المقاطعات الدورية بدقات الساعة.

يتميز المؤقت المبرمج بإمكانية التحكم بتردد المقاطعات برمجياً. إذا كان الهزاز الكريستالي المستخدم بتردد، فإن العداد ينقص كل 2 نانو ثانية. بوجود مسجل (غير مؤشر) بطول 32 بت، يمكن برمجة المقاطعات بحيث تحدث في فترات زمنية تتراوح من 2 نانو ثانية حتى ثانية. تحوي شرائح التوقيت المبرمجة عادة مؤقتين مبرمجين مستقلين أو ثلاثة مؤ عتاد المؤقتات

هناك نوعان من المؤقتات يشبع استخدامهما في الحواسب، وكلاهما مختلفان عن المؤقتات والساعات المستخدمة من قبل الناس.يرتبط الشكل الأبسط من المؤقتات مع خط التغذية أو فولت، ويعطي مقاطعة عند كل دورة جهد بتردد أو هرتز. كانت هذه المؤقتات شائعة جداً لكنها نادرة هذه الأيام.

يتألف النوع الآخر من المؤقتات من ثلاثة عناصر: هزاز كريستالي وعداد ومسجل ماسك، كما هو مبين في الشكل. عندما تقطع قطعة من الكوارتز المتبلور بشكل مناسب وتوضع تحت ضغط معين، يمكن جعلها تولد إشارة دورية بتردد دقيق جداً، ويكون عادة في مجال عدة مئات من الميغاهرتز حسب قطعة الكريستال المختارة.

يمكن باستخدام دارات الكترونية ضرب هذا التردد بعامل صحيح صغير للحصول على ترددات تصل إلى أو أكثر. توجد واحدة من هذه الدارات على الأقل في كل حاسب كي تعطي إشارة التزامن لدارات الحاسب المختلفة. تغذى هذه النبضات إلى العداد لجعله يعد هابطاً إلى الصفر. عندما يصل العداد إلى الصفر، فإنه يولد مقاطعة للمعالج.

تملك المؤقتات المبرمجة نموذجياً عدة أنماط عمل. لنأخذ مثلاً نمط الطلقة الوحيدة فعندما يبدأ الوقت ينسخ قيمة المسجل الماسك إلى العداد ثم ينقص العدد عند كل نبضة من الهزاز الكريستالي. عندما يصل العداد إلى الصفر، يولد مقاطعة ويتوقف حتى يتم بدؤه من جديد صراحة من قبل البرنامج. في نمط الموجة المربعة، بعد الوصول إلى الصفر وتوليد المقاطعة يتم نسخ المسجل الماسك تلقائياً إلى العداد، وتكرر العملية كلها بشكل غير نهائي. تسمى هذه المقاطعات الدورية بدقات الساعة.

يتميز المؤقت المبرمج بإمكانية التحكم بتردد المقاطعات برمجياً. إذا كان الهزاز الكريستالي المستخدم بتردد، فإن العداد ينقص كل 2 نانو ثانية. بوجود مسجل (غير مؤشر) بطول 32 بت، يمكن برمجة المقاطعات بحيث تحدث في فترات زمنية تتراوح من 2 نانو ثانية حتى ثانية. تحوي شرائح التوقيت المبرمجة عادة مؤقتين مبرمجين مستقلين أو ثلاثة مؤقتا وتتميز بعدة خيارات إضافية (مثل العد إلى الأعلى عوضاً عن الأسف وإزالة تفعيل المقاطعات وغيرها).

9 2 -برمجيات المؤقتات

إن كل ما يفعله عتاد المؤقت هو توليد مقاطعات عند فترات زمنية معروفة. كل شيء آخر متعلق بالتوقيت يجب إنجازه برمجياً من قبل برنامج تشغيل المؤقت. تختلف مهام برنامج تشغيل المؤقت باختلاف أنظمة التشغيل، لكنها تتضمن عادة معظم المهام التالية:

1- إدارة توقيت الساعة

2- منع العمليات من العمل أطول مما ينبغي لها.

3- حساب معدّل استخدام المعالج

4- معالجة استدعاء النظام الذي تستدعيه عمليات المستخدم

5- تقديم مؤقتات حراسة من أجل أجزاء نظام التشغيل

6- القيام بالتشخيص والمراقبة وجمع الإحصائيات.

الوظيفة الأولى للمؤقت هي إدارة توقيت الساعة اليومية (وتدعى أيضاً الوقت الحقيقي)، وهي ليست مهمة صعبة. فهي لا تحتاج إلا إلى زيادة العداد عند كل دقة ساعة كما شرحنا آنفاً. الشيء الوحيد الذي تجب مراقبته هو عدد البتات في عداد الساعة اليومية. إذا كان تردد الساعة هرتز وطول العداد بتاً. فإن هذا العداد سيطفح بعد سنتين فقط. من الواضح أن النظام لا يستطيع تخزين الوقت الحقيقي كعدد الدقات منذ في بت.

هناك ثلاثة أساليب يمكن إتباعها لحل هذه المشكلة. الطريقة الأولى هي استخدام عداد بطول بت، على الرغم من أن إدارة هذا العداد مكلفة زمنياً لأنها يجب أن تتم عدة مرات في كل ثانية. الطريقة الثانية هي تخزين الساعة بالثواني عوضاً عن دقات الساعة، وذلك باستخدام عداد فرعي بعد الدقات حتى يصل إلى ثانية واحدة. بما أن ثانية تساوي أكثر من سنة فإن هذه الطريقة ستعمل حتى القرن الثاني والعشرين.

تتمثل الطريقة الثالثة بعد الدقات، لكن نسبة إلى وقت إقلاع النظام عوضاً عن العد نسبة إلى لحظة خارجية ثابتة. عندما يقرأ النظام الساعة الاحتياطية، أو عندما يدخل المستخدم الوقت الحقيقي، فإن زمن إقلاع النظام يحسب من قيمة الساعة الحالية ويخزن في الذاكرة بأي طرقة مناسبة. وعندما تطلب الساعة لاحقاً، تضاف قيمة الساعة المخزنة إلى العداد للحصول على وقت الساعة الحالية.

الوظيفة الثانية للمؤقت هي منع العمليات من العمل لفترات أطول مما ينبغي لها. كلما بدأت عملية ما بالعمل، يهيء المجدول عداداً بقيمة الشريحة الزمنية الحقيقية المخصصة للعملية مقدرة بدقات الساعة. عند كل مقاطعة ساعة، يقوم برنامج تشغيل المؤقت بإنقاص هذا العداد بمقدار. وعندما يصل إلى الصفر، يستدعي برنامج تشغيل المؤقت المجدول من أجل إعداد عملية أخرى للعمل.

الوظيفة الثالثة للمؤقت هي حساب أزمان استخدام المعالج. الطريقة الأكثر دقة لإنجاز ذلك هي بدء مؤقت ثان منفصل عن مؤقت النظام الرئيسي كلما بدأت عملية. عندما تتوقف

العملية، يمكن قراءة المؤقت لمعرفة كم من الوقت عملت. لإنجاز الأمور بشكل صحيح يجب تخزين قيمة المؤقت الثاني عند حدوث المقاطعة واستعادة القيمة بعد معالجة المقاطعة.

هناك طريقة أخرى أقل دقة لكنها أبسط بكثير للقيام بهذه الوظيفة. وتتمثل بالاحتفاظ بمؤشر يشير إلى السجل الخاص بالعملية التي تعمل حالياً ضمن جدول العمليات ووضع هذا المؤشر في متحول عام. عند كل دقة ساعة، تتم زيادة حقل معين في سجل العملية الحالية. بهذه الطريقة تضاف كل دقة ساعة إلى حساب العملية الحالية التي كانت تعمل عند حصول دقة الساعة.

هناك مشكلة صغيرة لهذه الطريقة وهي أنه عند حصول الكثير من المقاطعات أثناء عمل العملية، فإن هذا الزمن يحسب على حساب هذه العملية مع أنها لم تقم فيه بالكثير من العمل. إن الحساب الصحيح لزمن المعالج أثناء المقاطعات مكلف جداً وقلما يستخدم.

في العديد من الأنظمة، تستطيع العمليات أن تطلب من نظام التشغيل أن يعطيها تنبيهاً بعد فاصل زمني محدد. يأخذ التنبيه عادة شكل إشارة أو مقاطعة أو رسالة أو أي شيء مشابه. من التطبيقات التي تحتاج مثل هذه الخدمات تطبيقات الشبكات، فعندما لا يصل إشعار باستلام رزمة ضمن فاصل زمني معين فإن الرزمة يجب إعادة إرسالها. من التطبيقات الأخرى أنظمة التدريس بمساعدة الحاسب، فعندما لا يستجيب الطالب ضمن زمن معين، يتم إعطاؤه الجواب الصحيح.

9 3 ـ المؤقتات البرمجية

يوجد في معظم الحواسب مؤقت مبرمج ثان يمكن إعداده لتوليد مقاطعات زمنية بأي تردد يحتاجه البرنامج. هذا المؤقت بالإضافة إلى مؤقت النظام الرئيسي هما المؤقتان اللذان تكلمنا عن وظائفهم أعلاه. طالما أن تردد المقاطعات منخفض، لا توجد أي مشكلة في استخدام هذا المؤقت الثاني من أجل الأغراض الخاصة بالتطبيقات. تظهر المشاكل عندما يكون التردد الذي يعمل فيه مؤقت خاص بالتطبيق عالياً جداً. سنشرح فيما يلي باختصار مؤقتاً برمجياً يعمل بشكل جيد تحت عدة ظروف مختلفة حتى عند الترددات العالية.

عموماً، هناك طريقتان لإدارة الدخل / الخرج: المقاطعات والاستجواب. تتميز المقاطعات بتأخيرها المنخفض، أي أنها تحدث فوراً بعد الحدث نفسه بتأخير صغير أو بدون تأخير. من ناحية أخرى، تعطي المقاطعات في معظم المعالجات الحديثة عبئاً لا يستهان به بسبب الحاجة إلى تبديل السياق وبسبب تأثيرها السيء على التنفيذ الأنبوبي والذاكرة المخبئية.

الطريقة البديلة للمقاطعات هي أن يقوم التطبيق نفسه باستجواب الحدث المتوقع. يؤدي ذلك إلى تجنب المقاطعات، لكن قد يؤدي أيضاً إلى تأخيرات كبيرة، لأن الحدث قد يقع مباشرة بعد الاستجواب مما يؤدي إلى تأخر اكتشافه حتى الاستجواب التالي. وسطياً، يكون التأخير هو نصف فترة الاستجواب.

من أجل بعض التطبيقات، لا يكون عبء المقاطعات ولا تأخير الاستجواب مقبولاً لديها. لنأخذ مثلاً شبكة عالية الأداء مثل. تستطيع هذه الشبكة إرسال أو استقبال رزمة كاملة خلال 12 ميكروثانية. ولكي تعمل بأداء كامل يجب أن ترسل رزمة واحدة كل 12 ميكروثانية.

تتمثل إحدى الطرق لتحقيق هذا المعدّل بجعل انتهاء إرسال الرزمة يؤدي إلى مقاطعة أو إعداد المؤقت الثاني لتوليد المقاطعة كل 12 ميكروثانية. هذا العبء أكبر قليلاً من العبء المقابل في أجهزة السبعينيات. ففي معظم الحواسب الصغيرة في ذلك الوقت، كانت المقاطعة تستهلك أربع دورات ممر: لتكديس عداد البرنامج وكلمة الحالة وتحميل عداد برنامج وكلمة حالة جديدين. أما في الحواسب المعاصرة، فإن التعامل مع التنفيذ الأنبوبي والذاكرة المخبئية يضيف قدراً كبيراً من الأعمال إلى العبء السابق. وربما يزداد الأمر سوءاً في الأيام المقبلة، وهذا يؤدي إلى إلغاء أثر ترددات الساعة السريعة التي تعمل بها المعالجات الحديثة.

تتجنب المؤقتات البرمجية حدوث مقاطعات. وعوضاً عن ذلك، كلما كانت النواة تعمل لأي سبب آخر، فإنها قبل أن تعود إلى نمط المستخدم تقوم بفحص عداد الوقت الحقيقي لترى إذا ما كان قد وصل إلى نهاية العد. إذا كان المؤقت منتهياً، يتم إنجاز الحدث المجدول (مثل إرسال رزمة أو فحص وصول رزمة واردة)، دون الحاجة للانتقال إلى نمط النواة لأن النظام موجود فيه حالياً. بعد إنجاز العمل المطلوب. يتم تصفير المؤقت البرمجي ويشغل مرة أخرى. كل ما يجب فعله هو نسخ قيمة الوقت الحالي إلى المؤقت وإضافة الفترة الزمنية للمقاطعة إليه.يزداد تردد المؤقتات البرمجية أو ينقص تبعاً لتردد الدخول إلى النواة من أجل أسباب أخرى.

تتضمن هذه الأسباب:

1- استدعاء النظام

2- فشل البحث في

3- أخطاء الصفحات

4- مقاطعات الدخل / الخرج

5- خمول المعالج

لمعرفة درجة تكرار هذه الأحداث، تم إجراء عدة قياسات عند عدة أحمال للمعالج تتضمن مخدم ذا حمل كامل، ومخدم مع عمل حسابي في الخلفية، وتشغيل صوت في الزمن الحقيقي من الإنترنت، وإعادة ترجمة نواة يونكس. تراوحت النسبة الوسطية لزمن الدخول إلى النواة بين و ميكروثانية، حيث كان حوالي نصف مرات الدخول عبارة عن استدعاءات نظام. وبالتالي فإنه من الممكن تشغيل مؤقت برمجي يعطي إشارة كل 12 ميكروثانية، على الرغم من أنه قد يتجاوز الفترة المطلوبة أحياناً. من أجل تطبيقات مثل إرسال رزم أو الاستجواب عن وصول رزم واردة، يكون التأخير لفترة 10 ميكروثانية بين الحين والآخر أفضل من هدر 35 % من وقت المعالج لمعالجة المقاطعات.

طبعاً، قد تحصل فترات ليس فيها أي استدعاءات نظام أو أخطاء صفحات، وفي هذه الحالة لا يستطيع المؤقت البرمجي أن يصل إلى نهايته. لوضع حد أعلى لهذه الفترات، يمكن استخدام المؤقت العتادي الثاني بحيث يعطي إشارة كل 1 ميلي ثانية. إذا كان التطبيق يستطيع التأقلم مع معدل إرسال رزمة / ثانية في فترات زمنية عرضية، فإن دمج المؤقتات البرمجية مع مؤقت عتادي منخفض التردد قد يكون أفضل بكثير من الدخل / الخرج المقاد بالمقاطعات بشكل كامل أو الاستجواب الصافي.

9 4 -ساعات الكيان الصلب

تتفاعل النواة مع ثلاث ساعات: ساعة الزمن الحقيقي , عداد الختم الزمني, و مؤقت الفترات القابل للبرمجة. إن أول جهازين يسمحان للنواة بالاحتفاظ بالوقت الحالي لليوم, أما الأخير فتتم برمجته بواسطة النواة ليقدح مقاطعات في ترددات ثابتة و معرفة مسبقا.

9 4 1 -ساعة الزمن الحقيقي "Real Time Clock"

تمتلك جميع الحواسيب الشخصية ساعة تسمى ساعة الزمن الحقيقي " Real Time Clock(RTC)" , والتي هي مستقلة عن المعالج والشرائح الأخرى.تتابع الـ RTC نبضاتها حتى عندما يكون الحاسب الشخصي مطفئا , حيث تستمد طاقتها من بطارية صغيرة. تتوضع ال CMOS RAM و الـ RTC على شريحة واحدة , وهي Motorola 146818 أو ما يعادلها.

إن الـ RTC قادرة على توليد مقاطعات ذات فترات على IRQ8 على ترددات تتراوح بين 2 Hz و 8192 Hz. كما يمكن برمجتها لتفعل المقاطعة IRQ8 عندما تصل الـ RTC إلى قيمة محددة , وهكذا تعمل كساعة منبه.

9 4 2 -عداد الختم الزمني

تحوي جميع المعالجات الصغرية لـ Intel 80x86 مدخل CLK , والذي يتلقى إشارة من دارة مهتزة خارجية، تحتوي العديد من معالجات Intel 81x86 مسجل عداد ختم زمني ب 64 بت (TSC) والذي يمكن قراءته بوساطة تعليمة المجمع rdtsc، حيث أن هذا المسجل هو عداد يتم زيادته عند كل إشارة ساعة: على سبيل المثال , إذا كانت نبضات الساعة عند 400 Hz, يزداد عداد الختم الزمني كل 2.5 نانو ثانية.

9 4 3 -مؤقت الفترات القابل للبرمجة

بالإضافة لساعة الزمن الحقيقي و عداد الختم الزمني , تحتوي الحواسيب المتوافقة مع IBM نوعاً ثالث من أجهزة قياس الوقت يدعى مؤقت الفترات القابل للبرمجة "Programmable Interval Timer(PIT)". إن مبدأ الـ PIT مشابه لساعة المنبه في فرن الميكروويف: لجعل المستخدم ينتبه أن زمن الطبخ قد انتهى, بدلا من رن الجرس يقوم هذا الجهاز بقدح مقاطعة تدعى مقاطعة المؤقت "timer interrupt" , والتي تعلم النواة أن فترة أخرى قد مرت. فرق آخر عن ساعة المنبه أن الـ PIT تستمر في قدح المقاطعات إلى اللانهاية عند تردد ثابت يحدد من النواة.

يحتوي كل جهاز متوافق مع IBM على PIT واحدة على الأقل , والتي تنفذ عادة بشريحة CMOS 8254 مستخدمة منافذ الدخل والخرج 0x40-0x43 و بشكل عام إن النبضات الأقصر تؤدي إلى استجابة أفضل من النظام. ذلك أن استجابة النظام تعتمد على مدى السرعة التي يتم فيها منع المعالجة التي تعمل من قبل معالجة ذات أفضلية أعلى , وأكثر من ذلك , تفحص النواة هل يجب أن يتم منع المعالجة عند معالجة مقاطعة المؤقت. هذه ما يسمى trade_off ولكن تتطلب النبضات الأقصر من المعالج أن يمضي أجزاء أكثر من وقته وهو يعمل في مستوى النواة. مما يؤدي إلى أجزاء أقل على مستوى المستخدم.وكنتيجة ستعمل برامج

المستخدم ببطء أكثر. لذلك الآلات القوية جدا تستطيع أن تعمل بنبضات صغيرة و تقدم أداء عالي.

9 5 -برنامج تخديم مقاطعات المؤقت

يقدح كل حدوث لمقاطعة المؤقت الأحداث الرئيسية التالية:

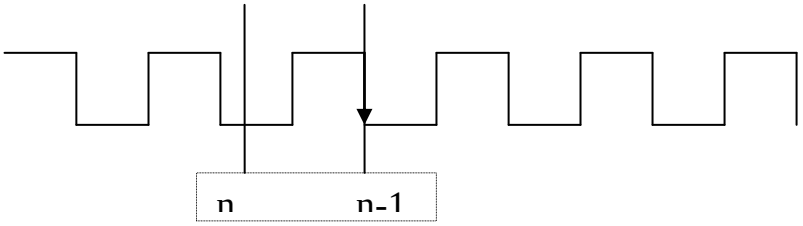
تحديث الوقت الذي مضى منذ تشغيل إقلاع النظام.
تحديث الوقت والتاريخ.
تحديث كم من الوقت مضى على المعالجة الحالية وهي تعمل في الـCPU وتمنعها إذا كانت قد تجاوزت الوقت المسموح.
تحديث إحصائيات استخدام الموارد.
التحقق فيما إذا كان قد انتهى الوقت المحدد لكل مؤقت مرن , واستدعاء التابع المناسب.

يعتبر العمل الأول هام جدا , لذلك يتم إنجازه بواسطة مقاطعة المؤقت نفسها.أما المهام الأربع الأخرى فهي أقل أهمية لذلك يتم إنجازها باستدعاء التوابع، حيث تستخدم النواة تابعين أساسيين للاحتفاظ بالوقت: يحافظ الأول على الوقت الحالي صحيحاً ويعد الثاني عدد المايكرو ثانية التي قد مرت بما في ذلك الثانية الحالية.

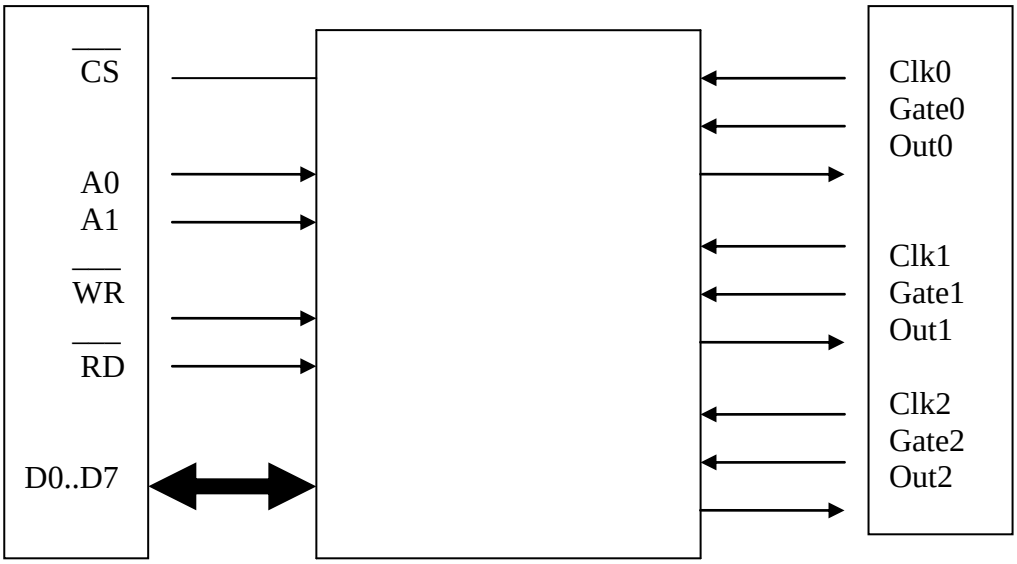
هناك طريقتان مختلفتان للحصول على تلك القيم: الطريقة الأكثر دقة متاحة إذا احتوت الشريحة على عداد الختم الزمني TSC والطريقة الأقل دقة في الحالات الأخرى. لذلك تنشئ النواة متحولين لتخزين التابع المستخدم , مؤشرة المتحولين بالتوابع المستخدمة لـ TSC عند وجودها.

9 6 -مؤقت النظام

هو عبارة عن دارة الكترونية هي Intel 8253 تتألف من ثلاثة مؤقتات تنازلية مستقلة مبرمجة بطول 16 بت نرمز للعدادات بـ counter0,counter1,counter2 تعمل جميع هذه العدادات على الجبهة الهابطة لإشارة الساعة أي أن انتقال قيمة العداد من n إلى n-1 أو من n-2 إلى n-3 يتم بنفس وقت انتقال إشارة الساعة من high إلى low.



يمكن رؤية هذا المؤقت من خلال مداخله ومخارجه على الشكل التالي:



يقوم مؤقت النظام داخل الحاسب الشخصي بتأدية الكثير من المهام. حيث تعبر المداخل Clk0 و Clk1 و Clk2 عن مداخل إشارة الساعة للعدادات الثلاثة. وهي تتصل ببعضها البعض داخل الحاسب الشخصي لتعمل على نفس التردد وهو 1193180Hz أي بدور مقداره 838.1 نانو ثانية. يؤثر المؤقت في الحاسب الشخصي عن طريق مخارجه الثلاثة Out0,Out1,Out2. إن الإشارات الناتجة في المخارج الثلاثة السابقة تتعلق بالعوامل التالية:

ساعة الدخل للعدادات.
إشارات التحكم بالعدادات (إشارة هاردوير).
ملاحظة:

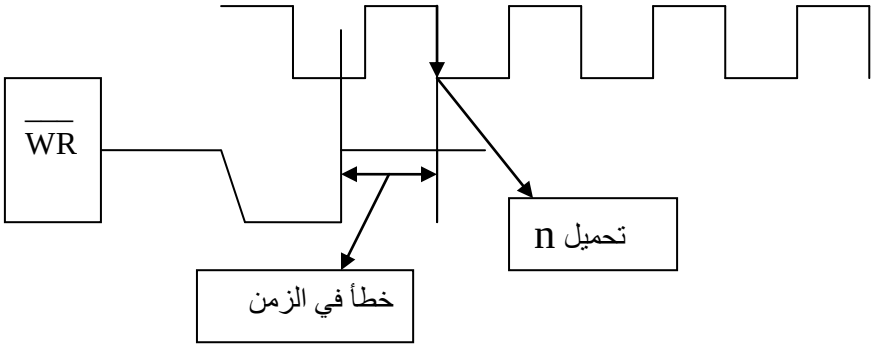
```
Gate0 = Gate1 = High
```

القيم الابتدائية للعدادات.
نمط العمل (Timer Mode) للعدادات.

يشير الشكل السابق أن المؤقت يتصل بخطي عنونة وهذا ما يؤدي إلى وجود أربعة عناوين بحيث يتم التعامل مع هذا المؤقت و برمجته عن طريقها.هذه العناوين في الحاسب الشخصي هي:

(0X40) خاصة بالعداد (0) لقراءة قيمة العداد وتحمي لقيمة ابتدائية له.
(0X41) خاصة بالعداد (1).
(0X42) خاصة بالعداد (2).
(0X43) تفيد في تحديد نمط العمل في العدادات الثلاثة.

تلعب إشارات التحكم بالعدادات دوراً مختلفاً بحسب نمط العمل الذي يتم اختياره. تتزامن تحميل القيم الأولية للعدادات مع كتابتها. يتم تحميل القيمة n للعداد بعد عملية الكتابة (بعد وصول الجبهة الصاعدة للإشارة WR) ومرور جبهة صاعدة وأخرى هابطة لنبضات الساعة.



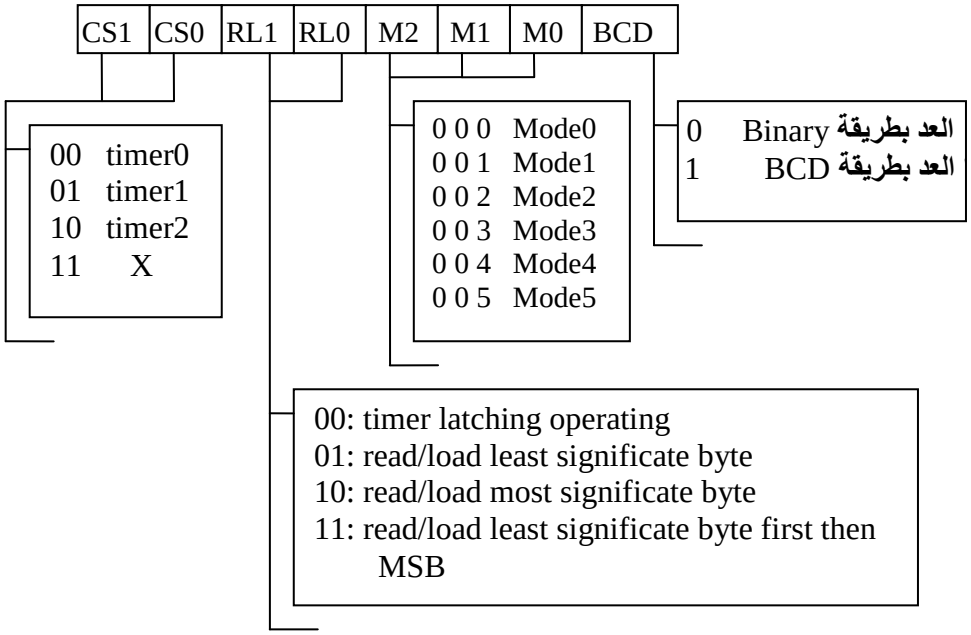
الشكل 90

9 7 ـ علاقة الموقت بالمقاطعة IRQ0

يتم ربط خرج العداد Timer0 وهو Out0 بمدخل طلب المقاطعة IRQ0 للدارة Master PIC وبالتالي فإن كل جبهة صاعدة على الخط Out0 تمثل طلباً للمقاطعة IRQ0 عندئذ تتم برمجة هذه الإشارة بحسب المطلوب وذلك من خلال اختيار نمط العمل للعداد TIMER0 من بين 6 أنماط مختلفة واختيار قيمة أولية للعد. مع العلم أن المدخل Gate0 متصل بالمستوى المنطقي High بشكل دائم.

9 8 ـ أنماط العمل للعداد

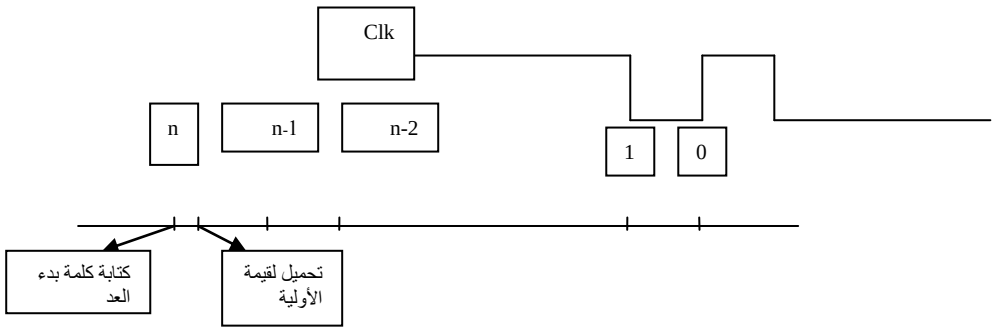
يتم التحكم بنمط العمل عن طريق كتابة كلمات التحكم في مسجل كلمة التحكم في العنوان (0X43).



الشكل 91

9 8 1- النمط 0 مقاطعة عند نهاية العد

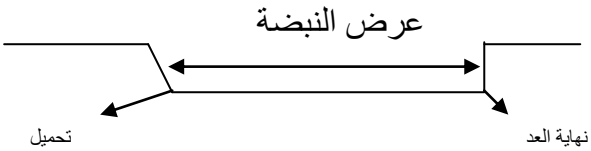
بعد كتابة كلمة التحكم أو تحميل كلمة بداية العد يبقى مستوى إشارة الخرج Out0 منخفضاً.



بعد شحن كلمة بداية العد للعداد تبدأ قيمة العداد بالتناقص عند كل جبهة هابطة للساعة وبنفس اللحظة التي تصل فيها قيمة العداد إلى الصفر يرتفع مستوى إشارة الخرج إلى High يستمر الوضع هكذا لحين حصول إعادة تحميل لكلمة بداية العد.

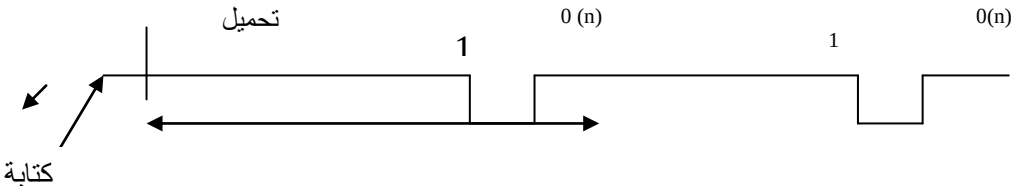
الشكل92

لا معنى لهذا النمط في الحاسب الشخصي لأنه يتعلق بقدوم جبهة صاعدة على المدخل Gate0 وذلك لأن قيمة هذا المدخل هي دائماً 1.



9 8 3 -النمط 2 مولد نسبة (Rate Generation)

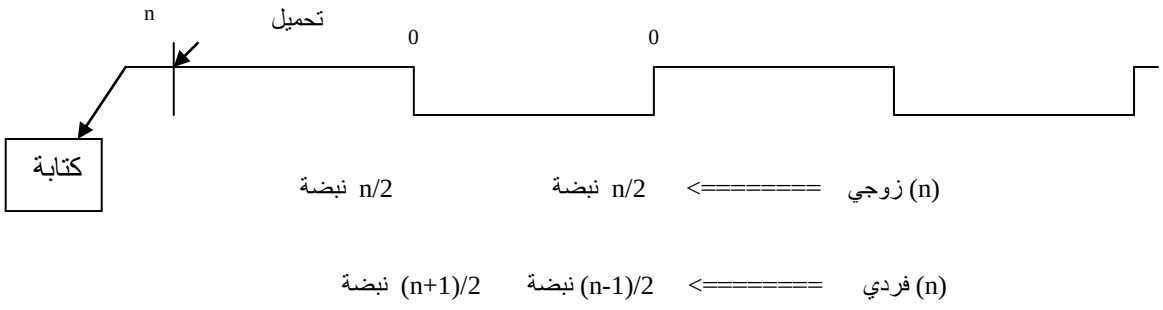
يبقى مستوى إشارة الخرج بشكل High بعد كتابة كل من كلمتي التحكم وبداية العد



يتم شحن قيمة بداية العد للعداد بشكل تلقائي بعد الكتابة , تبدأ بعدها عملية العد التنازلي لغاية الوصول إلى العدد (1)حيث يهبط مستوى الإشارة إلى Low ويستمر لفترة نبضة واحدة فقط للساعة , تكون عندها قيمة العداد أصبحت مساوية للصفر ولكنهنا وبدل إنقاص القيمة إلى 0 يتم شحن قيمة بداية العد بشكل تلقائي لتبدأ عملية العد من جديد وهكذا.

9 8 4 -النمط 3 مولد نسبة موجة مربعة

هو النمط الافتراضي في الحاسب الشخصي.هذا النمط شبيه بالنمط 2 من كونه يولد إشارة دورية ولكن بدل أن تكون قطاراً من النبضات. لدينا هنا تولد موجة مربعة على الشكل:

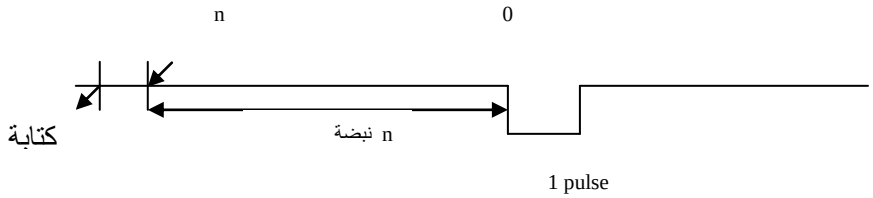


يختلف نمط العد هنا بشكل جذري عن الحالات السابقة ويتحقق ذلك من خلال إتباع مايلي:

في حال كون n زوجي: يتم طرح قيمة 2 بدلاً من 1 وعند الوصول إلى نهاية العد يتم عكس إشارة الخرج.

في حال كون n فردي: في بداية المستوى High يطرح لمرة واحدة العدد 1 ثم يتم طرح العدد 2 لغاية الوصول إلى القيمة 0 (n) , وفي بداية المستوى Low يطرح لمرة واحدة العدد 3 ثم يطرح العدد 2 لغاية الوصول إلى القيمة 0 (n).

9 8 5 -النمط 4 توليد نبضة برمجياً

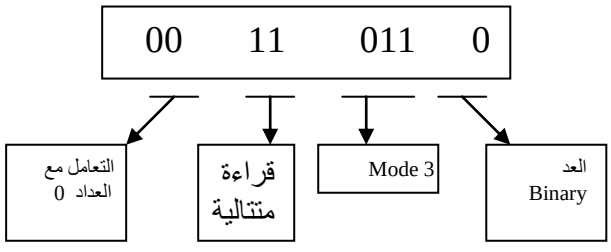


الشكل 93

يتم توليد نبضة ذات مستوى Low ومنها نبضة واحدة من إشارة الدخل Clk وذلك بعد انقضاء n نبضة تلي عملية كتابة بدء العد(يوجد تحميل تلقائي يلي عملية الكتابة).

9 8 6 -النمط 5توليد نبضة عن طريق القدح الخارجي
Hardware – trigger shote

يكافئ النمط 4 ولا يمكن استخدامه لأنه يحتاج إلى نبضة على Gate. والقيمة الافتراضية الخاصة بالعداد 0 وكلمة بداية العد هي (0X36) وتكافئ



الشكل 94

إذا كان نمط العد Binary يكون العد من 0000 إلى FFFF
إذا كان نمط العد BCD يكون العد من 0000 إلى 9999
يمكن حساب تردد إشارة الخرج الناتجة على الشكل:
تردد Clk = 1193180
نقوم بالتقسيم على n = 65536.
فيكون الناتج 18.20648 Hz.
أي كل ثانية يكون هنالك 18.2 مرة مقاطعة.
ويكون دورها عندئذ مساوي لـ 54.925 ms.

9 9 -تغيير تردد المقاطعة IRQ0

يمكن زيادة تردد المقاطعة وذلك بتغيير كلمة بداية العد للعداد timer0 وذلك على الشكل:

Freq d / N= 1193180

فإذا أردنا قيمة Freq =20 => n =59659 و بالست عشري 0XE906

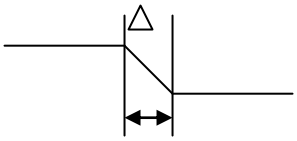
9 10 -كتابة القيمة الابتدائية للعد

تتم هذه الكتابة بأي تسلسل كان بالنسبة للعدادات الثلاثة أما للعداد نفسه فتتم الكتابة بحسب كلمة التحكم وبالتحديد بحسب قيمة R/L كما مرر معنا سابقاً تتم هذه الكتابة في أي وقت أي ليست مشروطة بأن تتبع مباشرة عملية كتابة كلمة التحكم.

9 11 -قراءة قيمة العداد في لحظة ما

يمكن تنفيذ هذه الطرقة بطريقتين:

الأولى: قراءة العداد وبشكل مباشر بما يتناسب مع كلمة RL (أي بدون Latch قراءة مباشرة من سجلات العداد). تعتبر هذه الطريقة سيئة لأن القراءة يمكن أن تتم بوقت تكون فيه قيمة العداد غير مستقرة.ولا يمكن الوثوق بالقيمة المقروءة إلا عن طريق إيقاف العد أولا ثم القراءة ثانياً وهذا غير ممكن برمجياً.



الشكل95

الثانية: قراءة قيمة العداد بشكل آمن وبدون التأثير على عملية العد , يتم ذلك بإرسال كلمة التحكم بالعداد ومع قيمة R/L=00 (latch) حيث يؤدي ذلك إلى تخزين داخلي Hardware لقيمة العداد ويتم قراءة هذه القيمة لاحقاً.

يتم قراءة القيمة المخزنة مرة واحدة فقط بحيث:

Send latch control	save counters values
Read counter value	read saved values
Read counrter value	read current counters

إن إرسال كلمة Latch لا تؤثر على كلمة التحم المرسلة سابقاً لذلك ليس هنالك داعي لإعادة كتابتها.إن استخدام تتعلق بالزمن مثل Delay يمكن أن تؤثر على قيمة العداد وبالتالي يجب عدم الربط بين قيمتي العداد قبل وبعد هذه التعليمات.وتعليمة Delay تؤثر على القيم المخزن باستعمال Latch أي تفقدها مضمونها.

9 12 -ملاحظات خاصة بالمقاطعة IRQ0

بعد انتهاء تنفيذ إجراء خدمة المقاطعة IRQ0 يتم استدعاء برنامج خدمة المقاطعة البرمجية 0X1C بشكل تلقائي أي أن هذه المقاطعة مخصصة لأي إجراء يكتبه المستخدم ومرتبط بالمقاطعة IRQ0 وذلك بغاية عدم التأثير على عمل النظام لذلك يفضل دائماً استخدام 0X1Cبدلاً من IRQ0 وذلك بحالة عدم تغير تردد المقاطعة.

إن استخدام IRQ0 يجب استدعاء إجراء التخديم التابع للنظام برمجياً بغية عدم التأثير على عمل النظام كما يجب إعادة تنصيب هذا الإجراء كما كان وذلك بعد الانتهاء لاستخدامنا IRQ0.

عند تغيير تردد المقاطعة IRQ0 يفضل استدعاء اجراء النظام الخاص بتخديمها طبقاً للتردد القديم وليس الجديد.

إن تغيير التردد من 18.2 إلى 182 يجب استدعاء إجراء النظام الخاص بتخديمها كل عشرة مقاطعات وليس كل مقاطعة.

9 13 -ساعة الزمن الحقيقي RTC

إن ساعة الزمن الحقيقي RTC هي عبارة عن جزء من معالج MC146818 كم إنتاج شركة موتورولا والذي يحتوي بالإضافة إلى هذه الساعة على ذاكرة من نوع RAM مكون من 64 بايت هنالك فقط 15 بايت تخص ساعة الزمن الحقيقي وهي على الشكل التالي:

0	sec
1	alarm sec
2	min
3	alarm min
4	hour
5	alarm hour
6	week day 1=sunday
7	month day 1..31
8	month 1..12
9	year
A.	reg A (Upadte Stat)
B.	reg B (Control)
C.	reg C (int source & ack)
D.	reg D (battery stat)
32	century

14 9 -عمل ساعة الزمن الحقيقي

تقوم هذه الساعة بشكل مستمر بتعديل قيم مسجلات الوقت والتاريخ وذلك تبعاً لنظام محدد بمسجل التحكم reg B وتقوم هذه الساعة وعن طريق اتصالها مع الدارة slave PIC بإرسال طلب المقاطعة IRQ8 ذات الرقم 0X70 والذي يمكن أن يعبر عن إحدى ثلاث مقاطعات محتملة تتعلق بساعة الزمن الحقيقي وهي:

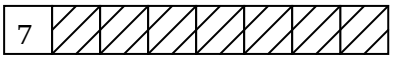
المقاطعة الدورية (periodic) وتحصل بتردد قدره 1024Hz.

الإنذار أ التنبيه (Alarm). يحدث عندما تتطابق مسجلات الزمن الخاصة بالمنبه مع المسجلات المقابلة لها في الساعة.(تحدث المقاطعة مرة أو اثنتين بحسب أسلوب التعبير عن الساعة 12-24)

مقاطعة التحديث Update: وتحدث عندما تتم عملية تحديث الوقت أو التاريخ داخل RTC.

لا تحدث هذه المقاطعات إلا إذا كانت فعالة (enabled) في مسجل التحكم regB.

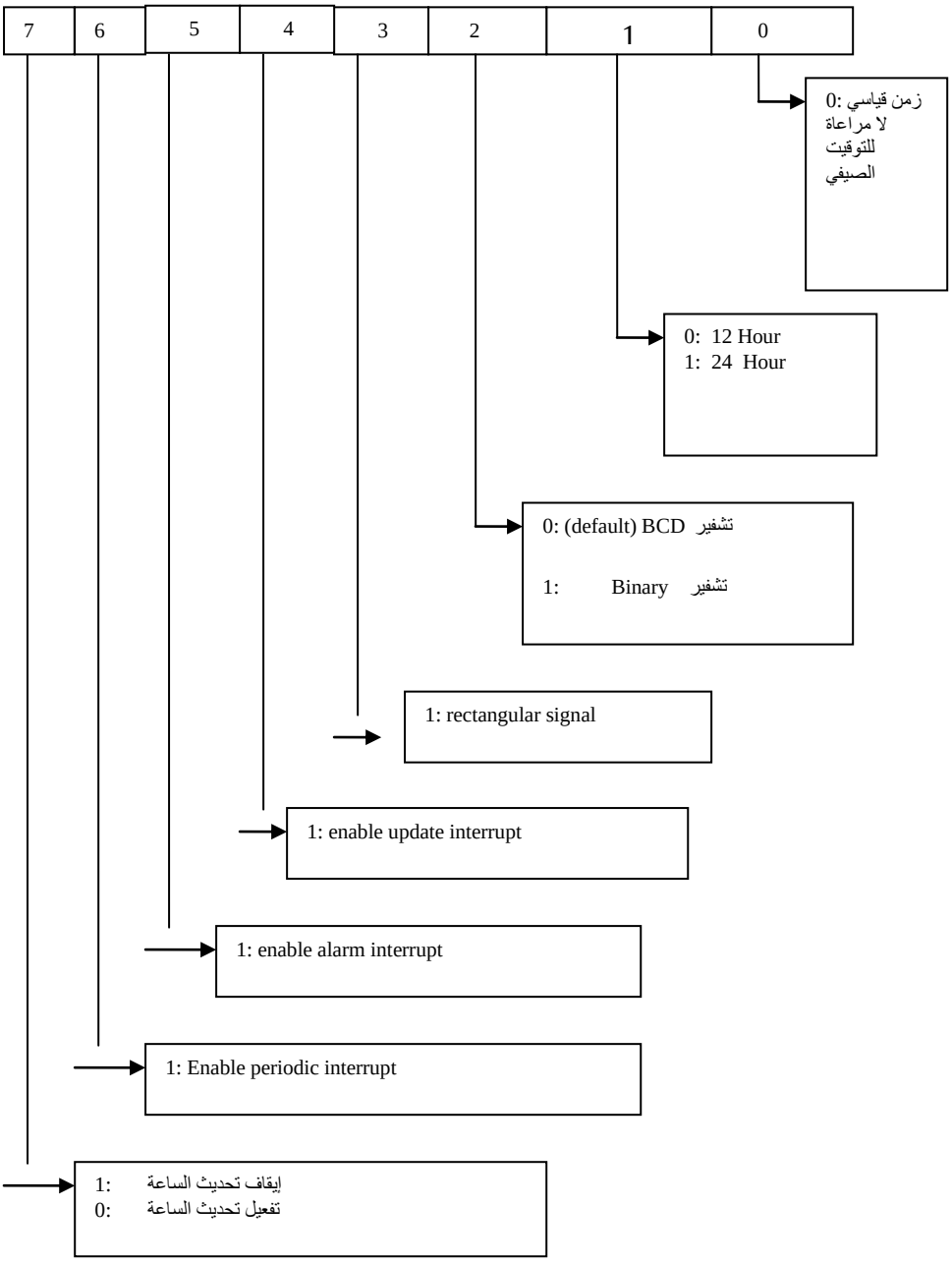
9 15 -دلالات المسجلات A,B,Cالمسجل A



UIP: (Update in progress)

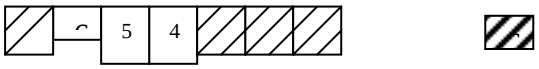
الشكل 96

9 15 1- المسجل B(مسجل التحكم)



الشكل 97

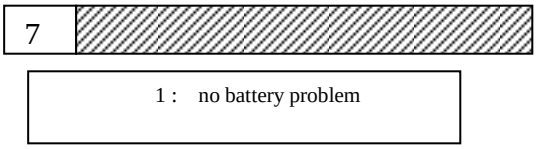
9 15 2- المسجل regC



يفيد في معرفة مصدر المقاطعة من بين المصادر الثلاثة المحتملة.

يجب قراءة المسجل regC حتى في حالة وجود مصدر مفعل وحيد للمقاطعة لأن هذه القراءة تفيد بإجراء عملية Ack داخل RTC تقوم بتصفير الخانة في حال إنهاء تخديم المقاطعة , ولا تستطيع دارة RTC إرسال طلبات مقاطعة جديدة لدارة Slave PIC إن لم تتم قراءة قيمة هذا المسجل بعد حدوث مقاطعة سابقة.لا يمكن أن يوجد في نفس الوقت أكثر من مقاطعة واحدة و إن تغير المصدر.

9 15 3- المسجل regD



الشكل 98

9 16- الوصول إلى مسجلات ساعة الزمن الحقيقي

بالرغم من وجود ذاكرة RAM داخلية بسعة 64 بايت إلا أن عمليتي الكتابة لها والقراءة منها تتم عن طريق بوابتين هما بوابة العنوان RTC Add Port و بوابة المعطيات RTC data port.

9 16 1- بوابة العنوان RTC Add Port

عنوانها 0X70 وتستخدم للكتابة فقط. وهي متصلة مع ممر العناوين address bus للذاكرة الداخلية RAM وبالتالي فهي تفيد في تحديد المسجل الداخلي المرغوب التعامل معه. وبعبارة أخرى تفيد هذه البوابة بربط مسجل داخلي ما مع البوابة الأخرى (بوابة المعطيات)

9 16 2- بوابة المعطيات RTC data port

عنوانها 0X71 تستخدم للقراءة والكتابة من وإلى أي من المسجلات الداخلية للدارة RTC.تتصل هذه البوابة مع بايت واحدة من الذاكرة RAM يحدد عن طريق القيمة المرسلة إلى بوابة العنوان.

9 17- الجانب العملي

9 17 1- مقدمة

يتألف الجانب العملي للمؤقت وتطبيقاته لدينا من 9 مكتبات هي على الترتيب:

- **Time.h**
- **kernel/time.h**
- **times.h**
- **clock.h**
- **time.c**
- **asctime.c**
- **ctime.c**
- **localtime.c**
- **clock.c**

وسنتطرق لهذه المكتبات بالقليل من التفصيل تبعا لأهمية المكتبة.

9 17 2- المكتبة time.h

تحتوي هذه المكتبة على ماكرو يقوم بحساب إذا كانت السنة كبيسة أم لا, وذلك من خلال اتفاق عالمي تم التعارف عليه لتعريف السنة الكبيسة , وهو أن السنة الكبيسة هي السن التي تكون من مضاعفات الأربعة على ألا تكون في نفس الوقت من مضاعفات المائة أو أن تكون من مضاعفات الأربعمائة.ويكون التابع بالشكل:

```
isleap(y) ((((y) % 4) == 0 && ((y) % 100) != 0) || ((y) %
400) == 0)
```

كما تحتوي هذه المكتبة على بنية تحوي متحولات لكل من السنة والشهر واليوم في السنة واليوم في الأسبوع واليوم في الشهر والساعات والدقائق و الثواني يستخدم كي نخزن فيه الوقت المستخدم في تابع الحصول على الوقت mktime().

وتحتوي على بنية تستخدم لنحدد عدد الثواني و أجزاء الثواني عندما نريد التأخير بشكل دقيق.وتحتوي أيضا على التابع mktime() الذي يكون دخله هو الوقت الحالي المأخوذ من ساعة الزمن الحقيق.أما خرجه فهو عدد الثواني التي مرت منذ 1970-01-01 00:00:00 وهو ما يعرف بالختم الزمني لينكس (UNIX timestamp)،وفي النهاية نجد تصريح عن توابع ستستخدم في المكتبات ctime,asctime,localtime والتي سنشرحها الآن.

9 17 3 -المكتبة asctime

تحتوي هذه المكتبة على التابع asctime_r()* الذي يقوم بأخذ التوقيت من ساعة الزمن الحقيقي (أي بالشكل المعروف ولكن كأرقام) ثم يربط رقم يوم الأسبوع بمقابله من سلسلة ثلاثية وكذلك رقم الشهر أيضا بسلسلة ثلاثية ثم يقوم بتخزينها في buffer.أما التابع الآخر asctime() فيقوم بطباعة قيمة البفر إلى الشاشة.

9 17 4 -المكتبة ctime

تحتوي هذه المكتبة على التابع ctime_r () * والذي يشابه إلى حد كبير التابع السابق في المكتبة السابقة إلا أنه يختلف عنه بأن دخله ليس قيم ساعة الزمن الحقيقي RTC وإنما تكون عدد الثواني منذ الختم الزمني لينكس (UNIX timestamp) ويقوم التابع من خلال عمليات قسمة وباقي قسمة بالحصول على الزمن المطلوب , ثم يقوم بالربط بين رقم يوم الأسبوع رقم الشهر بالاسم المناسب لكل منهما , ويخزنه في بفر , ثم يقوم التابع ctime() بطباعة قيمة البفر إلى الشاشة.

9 17 5 -المكتبة localtime

تحتوي هذه المكتبة على التابع localtime_r() الذي يشابه كثيرا التابع السابق ctime_r () من حيث طريقة حساب الوقت إلا أنه يختلف عنه بالإظهار فهنا يقوم بتخزين القيم الرقمية في البفر , ثم يأتي التابع localtime () ليطبع محتوى البفر على الشاشة.

9 17 6 -المكتبة kernel/time.h

تعتبر هذه المكتبة غاية في الأهمية من حيث التعامل مع دارة المؤقت والتي تم شرحها في القسم النظري.وتحتوي هذه المكتبة على بعض الاسنادات (التعاريف)التي تعبر عن خانات ساعة الزمن الحقيقي RTC ومسجلاتها ومنافذها كي يسهل التعامل معها ,كما تحتوي على بنية struct من أجل قراءة الزمن فيها , وتحتوي على تصريح عن توابع القراءة والكتابة من وإلى ساعة الزمن الحقيقي , وتصريح عن الإجراء المنفذ عند طلب استدعاء النظام time() والذي مهمته إرجاع عدد الثواني منذ الختم الزمني لينكس 00:00:00 01-01-1970

9 17 7 -المكتبة time.c

تحتوي هذه المكتبة على تنفيذ التوابع المذكورة سابقاً..فالتابع time_read() يقوم بالقراءة من دارة ال CMOS إلى بنية الـ struct لساعة امن الحقيقي التي تم ذكرها منذ قليل.أما التابع time_write() الغير مستخدم حاليا فمهمته المستقبلية أن يقوم بالكتابة إلى ساعة الزمن الحقيقي.وهنالك أيضا التابع sys_time() الناتج كما قلنا عن طلب استدعاء النظام time() حيث يقوم هذا التابع باستدعاء التابع mktime() المشروح سابقاً.

9 17 8 -المكتبة clock.h

تحتوي هذه المكتبة على بعض الاسنادات التي تعرف منافذ المؤقت ونمط العمل وتردده كما تعرف بنية:

```
typedef struct timeout
{
	dword start_ticks;
	dword ticks_to_wait;
} timeout_t;
```

التي تساعد في عملية التزامن والتنبيه عند انقضاء وقت معين.كما تحتوي على تصاريح توابع التأخير والتنبيه والتعامل مع المؤقت للحصول على النبضات.

9 17 9 -المكتبة clock.c

تحتوي هذه المكتبة على تنفيذ التوابع السابقة , فمثلاً التابع delay يمرر له عدد الميلي ثانية المراد تأخيرها ثم يقوم بحفظ عدد النبضات الحالي ثم يقارن حتى يصبح الفرق بين عدد النبضات المحفوظ (لحظة البدء) وعدد النبضات الحالي أكبر أو يساوي القيمة الممررة في التابع عندها يخرج من الحلقة.كذلك نجد التابع set_timeout الذي سيقوم بإسناد النبضات الحالية وفترة التأخير المرادة إلى المتحول من نوع البنية timeout. والمرتبط جذرياً بالتابع is_timeout ولذي يقوم بدوره بإرجاع قيمة TRUE عندما تنقض المدة time out.وتكون بنية هذا التابع على الشكل التالي:

```
bool is_timeout(timeout_t *t)
{
	register uint32_t elapsed = ticks - t->start_ticks;
	return( elapsed >= t->ticks_to_wait );
}
```

وهنالك أيضا التابع الذي يرجع عدد النبضات الحالي من المؤقت.ومن الأجزاء الهامة في هذه المكتبة هو برنامج تخديم مقاطعة المؤقت والذي يتم استدعاءه كل نبضة والذي يقوم بزيادة المتحول الذي يحوي عدد النبضات بمقدار واحد , وكذلك يستدع التابع scheduale() الذي يقوم بتنظيم عمل المعالجات المنتظرة في الأرتال المختلف وقد تم شرح آلية عمله بالتفصيل في قسم إدارة المعالجات. ويكون بالشكل التالي:

```
void clock_handler( irq_context_t *c )
{
	// Increment the system clock ticks.
	ticks++;

	// Put here every periodical task...
	floppy_thread();

	// Ack the irq line.
	end_of_irq( c->IRQ );

	// Call the scheduler.
	schedule();
}
```

وأخيراً نجد التابع ()init_clock الذي يتم استدعاءه عند إقلاع النظام وهو يقوم بتهيئة المؤقت ليعمل على النمط الأول كما شرحنا في القسم النظري.

10 -لوحة المفاتيح

مُقدِّمة

تعتبر لوحة المفاتيح الجهاز المحيطي الأساسي المستخدم في التعامل مع الحاسب الشخصي و إدخال البيانات النصية و الرقمية إليه, و كنظام تشغيل من الصعب أن نتخيله بدون عملية إدخال أوامر من خلال لوحة المفاتيح, و لذا فإن لوحة المفاتيح تعتبر من أجهزة الدخل إلى النظام.

و سنبحث في هذه الدراسة في النقاط التالية:

- البنية العتادية للوحة المفاتيح
- مقاطعات الموظفة لتخديم هذه المحيطية
- برنامج تشغيل لوحة المفاتيح.
- البنية البرمجية للتعامل مع لوحة المفاتيح.

سنقوم في هذه الدراسة بذكر بعض الوظائف المصطلحات و عناوين رئيسية و من ثم سنكامل هذه العناوين الرئيسية في سيناريو كامل لآلية عمل لوحة المفاتيح, لذا عزيزي القارئ أنصحك بالصبر قليلا في الصفحات القادمة ريثما تصل لفقرة سيناريو عمل لوحة المفاتيح. و بسم الله نبدأ.

10 -1 -البنية العتادية للوحة المفاتيح

إن الأجهزة المحيطية التي تتصل بالحاسب الشخصي تصنف بشكل عام ضمن صنفين رئيسين و ذلك من وجهة نظر المعلومات المتناقلة بين الجهاز و الحاسب:

الأجهزة المحرفية
أجهزة دفق المعلومات.

وموقع لوحة المفاتيح من هذين الصنفين هو في الصنف المحرفي, وذلك لأنها تقدم للحاسب الشخصي معلومات محرفية.

إن البنية الداخلية للوحة المفاتيح كدارة الكترونية هي بنية مصفوفية, أي أن لوحة المفاتيح هي عبارة عن مصفوفة مربعة من الأسلاك وعلى كل نقطة تقاطع من هذه الأسلاك يتوضع زر من أزرار لوحة المفاتيح, و هنالك في لوحات المفاتيح القياسية ATA لدنيا 101 مفتاح تتوضع على نقاط تقاطع مصفوفة الأسلاك. وتكون الأسلاك في نقاط التقاطع متباعدة عن بعضها البعض و لدى الضغط على المفتاح يتم حدوث تلامس بين نقاط التقاطع هذه لكي يتم

تشكيل نبضة كهربائية ذات جبهة ويتم من خلال هذه النبضة المتشكلة من حدوث تماس سلكي نقطة التقاطع التعرف على السلكين الذين تم حدوث النبضة عليهما ويتم بالتالي تحديد موقع الزر الذي ضغط.

متحكم لوحة المفاتيح

عادة يكون هذا المتحكم هو عبارة عن شريحة متحكم INTEL 8048 و بواسطة هذه المتحكم فإن هذه الكومبيوترات تصبح قادرة على الاتصال ثنائي الاتجاه بين الـ CPU و لوحة المفاتيح علما أن الكومبيوترات الأقدم لا تمتلك هذه المتحكمات.

وظيفة هذه المتحكمات هي عملية اكتشاف الأسلاك التي تم حدوث التماس بينها على المصفوفة. وتحويل مواقع هذه الأسلاك إلى قيمة ثنائية تدعى بشفرة المسح Scan Code.

كما أن وظيفة هذه المتحكمات هي التحكم بأضواء الإشارة على لوحة المفاتيح و أيضا من وظائف هذا المتحكم القيام بضبط سرعة تواتر لوحة المفاتيح وسرعة الاستجابة لضغط المفتاح. ويقوم هذه المتحكم بعملية الاتصال التسلسلي المتزامن مع الحاسب الشخصي وإرسال قيمة شفرة المسح إلى الحاسب والتحقق من وصول هذه القيمة. و يتحقق الاتصال المتزامن باستخدام خط الساعة و خط المعطيات.

إن لوحة المفاتيح هي عبارة عن وحدة مستقلة ترتبط مع نظام الحاسب الشخصي و هذه الوحدة المستقلة يتم إدارتها من خلال المتحكم الصغري الخاص بها كما نعلم أن هذا المتحكم له ذاكرته الخاصة و مسجلاته الخاصة به. و مهمة هذا المتحكم الصغري هي أخبار نظام الحاسب الشخصي بحدوث حدث معين في لوحة المفاتيح و بالطبع هذا الحدث هو النقر على أحد ازرار لوحة المفاتيح و هذه المهمة تتم من خلال إرسال شفرة المسح للزر الذي تم الضغط عليه أو تحريره.

وكما ذكرنا أن اتصال لوحة المفاتيح مع نظام الحاسب الشخصي يتم من خلال نظام الاتصال التسلسلي التزامني و باستخدام خطي اتصال ثنائيي الاتجاه للإرسال و الاستقبال. حيث يستخدم أحد خطي الإرسال و الاستقبال لإرسال المعطيات بت و راء الآخر بشكل تسلسلي بينما الخط الآخر يستخدم كساعة تزامن للإرسال. و في كل عملية إرسال يتم إرسال بايت كامل و الذي هو عبارة عن شفرة المسح كما نعلم حيث أن هذا البايت يتم إرساله بت بت و يكون البت الأول هو البت الأقل أهمية. و بتم إرفاق هذا البايت ببت التوقف و بت الفحص و بت البداية.

وبالتالي الـ Block المرسل في كل عملية إرسال يتألف من bit11 يحوي على بايت معطيات.

ملاحظة: بت الفحص يقوم على حساب عدد البتات في حال كانت زوجية أم لا.

كما يجب أن نعرف أن هنالك فرق بين لوحات المفاتيح الخاصة بحواسب PC XT والتي تستخدم المتحكمات ذات النوع Intel 8048 و بين لوحات المفاتيح الخاصة بحواسب الـ PC AT التي تستخدم المتحكمات 8042 حيث أصبحت الخدمات المتاحة في هذه اللوحات أكثر وكما أن اتصالها مع الحاسب الشخصي أصبح أكثر تعقيدا.

أما من الناحية البرمجية و التي تهمنا نحن كمبرمجين هي معرفة بنية مسجلات هذا المتحكم وكيفية التعامل معها حيث أن عمليات الاتصال والإدخال والإخراج تتم من خلال ذواكر وسيطية وعناوين بوابات محددة و تتألف هذه البوابات من:

مسجل حالة Status Register.
ذاكرة إدخال.Input Buffer
ذاكرة إخراج Output Buffer .

هذه الذواكر تستخدم لنقل ما يلي من المعطيات:

- رموز أزرار لوحة المفاتيح أو شفرة المسح بشقيها الضغط و التحرير الـ Make&Break.
- المعطيات و المعلومات التي يحتاج نظام الحاسب الشخصي أن يعرفها عن لوحة المفاتيح.

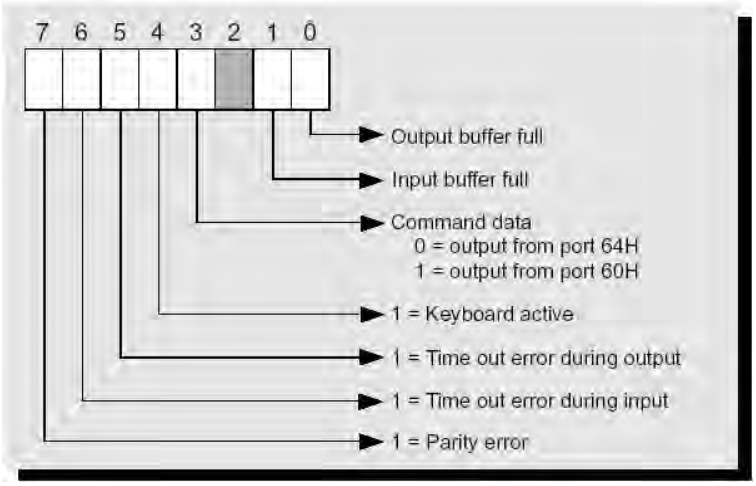
10 .1 .1 -مسجلات متحكم لوحة المفاتيح.

يمكن أن نتعامل مع ذاكرة الإدخال Input Buffer من خلال كلا العنوانين التاليين 60H والبوابة 64H. حيث يتم الكتابة في ذاكرة الإدخال هذه من خلال هاتين البوابتين بنفس الطريقة ولكن تستخدم إحداهما لكتابة تعليمات التحكم بمتحكم لوحة المفاتيح 60H و البوابة الأخرى لكتابة المعطيات إلى لوحة المفاتيح على العنوان الـ 64H.

لكن كيف تمييز لوحة المفاتيح بأن ما نكتبه الآن في الـ Input Buffer هو قادم من العنوان H60 أو 64H ؟ يتم هذا التمييز من خلال بت الثالث في مسجل الحالة المبين في الشكل التالي:

حيث تقوم هذه الإجرائية بفحص المعطيات على البوابة 64H من خلال قراءة محتويات مسجل الحالة بشكل متكرر و في حال كانت البوابة فارغة و مستعدة لاستقبال معطيات جديدة فعندها يتم إرسال بايت التحكم و من ثم يتم فحص وصول البايت إلى لوحة المفاتيح من خلال قراءة مسجل الحالة على البوابة 64H في حال كان الإرسال ناجحا فيجب أن تكون القيمة المقروءة لمسجل الحالة هي 0XFAH و أي قيمة غير هذه فإن ذلك يعني وجود خطأ و بالتالي يجب إنقاص عداد هامش الأخطاء المتاح لإرسال بايت و يتم تكرار العمل مرة أخرى إلى نصل فيها إلى انعدام هامش الأخطاء المتاح للإرسال عندها يتم إلغاء عملية الإرسال و إلا فإن الإرسال يكون قد تم بنجاح.

مسجل الحالة:



الشكل99

إن كل من البتين الأول و الثاني في المسجل يلعبان دورا مهما في عملية الاتصال بين الحاسب ولوحة المفاتيح وتحقيق التزامن خلالها كما يلي

البت الأول Bit0: يشير لحالة ذاكرة الإخراج Output Buffer

Bit0= 1: تكون ذاكرة الخرج ممتلئة بالمعلومات و لم تقرأ حتى الآن من قبل الـ Port 60H و في حال تم قراءة هذه الذاكرة فإنه مباشرة سيتم تصفير هذا البت مباشرة.

البت الثاني Bit1: يعمل نفس عمل البت الأول و لكن بالنسبة للـ Input Buffer. حيث أنه في حال كانت قيمة البت = 1 فإنه لن تستطيع أن تكتب أي شيئ ضمن هذه الذاكرة.

كما يتم التعامل مع المنفذ 60H لقراءة المعطيات من لوحة المفاتيح والتي تحوي على شفرات الضغط و التحرير.

10 1 2 -معدل سرعة الاستجابة للوحة المفاتيح Typematic Rate

إن من بين العديد من تعليمات التحكم التي يقوم نظام التشغيل بإرسالها إلى لوحة المفاتيح يوجد تعليمتين تهمنا كمبرمجين.

الأمر الأول هو الذي يحدد معدل سرعة تكرار للوحة المفاتيح, أي عدد شفرات المسح التي يمكن للوحة المفاتيح أن ترسلها إلى نظام الحاسب و ذلك عندما نضغط على الزر و نبقيه مضغوطا. و هذا التواتر يبلغ عادة 30 شفرة مسح في الثانية. و الآن للتحكم بهذا المعدل و لكي لا يكون عشوائيا فإن هنالك إمكانية لضبط ذلك حيث أن التابع الذي يتم تكراره سوف لن ينفذ إلا بعد فاصل زمني محدد يتم وتحديده من خلال المبرمج.

إن معدل التكرار هذا يشفر ثنائيا و الجدول التالي يظهر معدل التكرار و الرقم الثنائي المقابل لها:

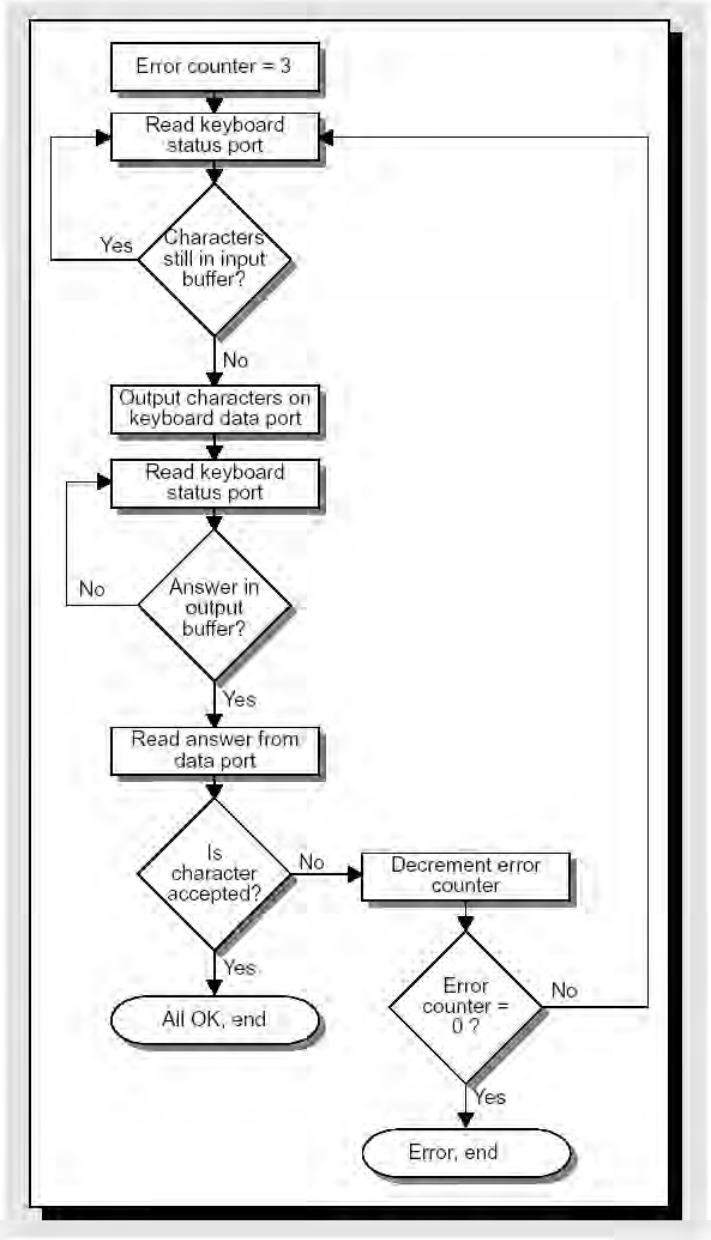
Typematic rate codes for the AT keyboard							
Code	RPS*	Code	RPS*	Code	RPS*	Code	RPS*
11111b	2.0	10111b	4.0	01111b	8.0	00111b	16.0
11110b	2.1	10110b	4.3	01110b	8.6	00110b	17.1
11101b	2.3	10101b	4.6	01101b	9.2	00101b	18.5
11100b	2.5	10100b	5.0	01100b	10.0	00100b	20.0
11011b	2.7	10011b	5.5	01011b	10.9	00011b	21.8
11010b	3.0	10010b	6.0	01010b	12.0	00010b	24.0
11001b	3.3	10001b	6.7	01001b	13.3	00001b	26.7
11000b	3.7	10000b	7.5	01000b	15.0	00000b	30.0
* Repetitions Per Second							

جدول12

للوهلة الأولى قد تعتقد عزيزي القارئ أن الأرقام الثنائية تقابل معدل التكرار بدون قاعدة مسبقة, لا هنالك قاعدة و علاقة رياضية تحدد العلاقة بين معدل التكرار و الرقم الثنائي المقابل له. و هي بفرض أن القيمة الثنائية للبتين الأول و الثاني و الثالث هي A و البت الرابع و الخامس هي B فإن معدل التكرار يكون حسب العلاقة التالية:

$$\text{Repeat Rate} = (8 + A) * 2B * 0.00417 * 1\ \text{second}^{-1}$$

و إن معدل التكرار هذا يتم إرساله إلى لوحة المفاتيح بعد أن نكون قد أرسلنا أمر التحكم المناسب (34H) و هذا الإرسال يتم إلى المنفذ 60H و لكن لن نستطيع أن نرسلها من خلال تعليمة الأسمبلي OUT العادية فقط و الاعتماد عليها في ذلك, بل يجب أن نستخدم بروتوكول نقل يتضمن قراءة حالة لوحة المفاتيح و يختبر إمكانية النقل و عدم حدوثها. و يجب أن يتم عمل هذا البروتوكول على كلا البايتين المرسلين. و بالتالي علينا أن نكتب إجرائية تقوم بذلك و بنية هذه الإجرائية موضحة بالمخطط التدفقي التالي:



الشكل100 مخطط التدفق لإجرائية إرسال Byte إلى لوحة المفاتيح.

10 .1 .3 -أضواء الإشارة

إن الثنائيات الضوئية في لوحة المفاتيح تعتمد على شفرة أوامر لإضاءتها أو إطفائها(SET/RESET) بما يناسب مع حالة المفاتيح الموافقة لها و شفرة الأوامر هذه هي الرقم 0EDH و بعد أن تنقل هذه التعليمة بشكل صحيح تنتظر لوحة المفاتيح وصول بايت يعكس حالة هذه الثنائيات و كل بت يمثل حالة ثنائي واحد منها و التي تضاء في الحالة (1).

فمثلا يقوم الـBIOS عند تفعيل المفتاح CAPSLOCK بجعل قيمة علم داخلي تساوي الواحد و يرسل البايت مع جعل البتات المناسبة في حالة الواحد إلى لوحة المفاتيح لإضاءة الثنائي الموافق.

عادة يقوم المستخدم بتحديد حالة المفاتيح مثل CAPSLOCK بالضغط عليها و لكن بالإمكان تفعيلها بشكل أوتوماتيكي عند بدء برنامج ما و إضاءة الثنائيات لإعلام المستخدم بتفعيلها و بما أن موقع الأعلام (المسئولة عن تغيير الثنائيات) هو عبارة عن متحول ذاكري في منطقة الخاصة بالمتحولات العامة في BIOS كما سنجد ذلك في فقرة لاحقة و بتات الأعلام هذه معروفة فيها يصبح من السهل تغييرها.

فيما يلي المخطط الصندوقي لبرنامج يقوم لدى تنفيذه بتغيير إضاءة الثنائيات بشكل منتظم و متواتر.

10 1 4 -متحولات مقاطعات لوحة مفاتيح الـBIOS

تملك نظام الإدخال و الإخراج الأساسي BIOS ثماني متغيرات من أجل إدارة لوحة المفاتيح والاتصال بين برنامج خدمة المقاطعة الصلبة للوحة المفاتيح (المقاطعة 09H) وبرنامج خدمة لوحة المفاتيح المرن لوحة مفاتيح الـBIOS (المقاطعة 16H).

عنوان الإزاحة (19H) هو البايت المستخدم عند إدخال شفرة الـ ASCII مع المفتاح ALT واللوحة الرقمية.فعندما يتم ضغط مفاتيح الأرقام سيخزن الكود المدخل في هذا البايت.

إن متحولات مقاطعة لوحة مفاتيح الـ BIOS مسجلة ضمن القائمة التالية:

BIOS variables for keyboard management

Offset	Meaning	Type
17H	Keyboard status	1 byte
18H	Extended keyboard status	1 byte
19H	Code for ASCII input	1 byte
1AH	Next character in keyboard buffer	1 word
1CH	Last character in keyboard buffer	1 word
1EH	Keyboard buffer	16 words
80H	Start address of keyboard buffer	1 word
82H	End address of keyboard buffer	1 word

جدول13

BUFFER لوحة المفاتيح:

المتحولات الثلاثة التي تلي عناوين الإزاحة 1AH,1CH,1EH تدير مسجل لوحة المفاتيح. حيث أن معالج مقاطعة لوحة المفاتيح (المقاطعة 09H) يخزن ضغطة كل مفتاح بحيث يمكن للتطبيقات قراءتها باستخدام مقاطعة لوحة مفاتيح الـ BIOS (16H).

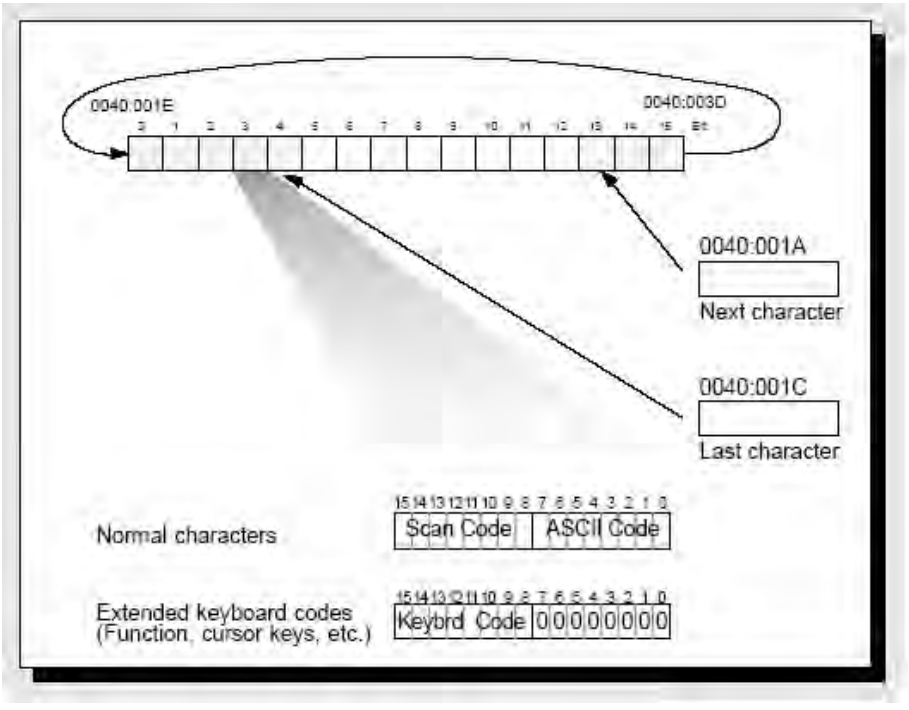
إن buffer لوحة المفاتيح مبني كـ buffer حلقي, و يستخدم لكتابة المحارف إلى الـ buffer وقراءتها منه بنفس الوقت.

في البداية كانت أفضل طريقة لتخزين المحارف في هذا الـ buffer هي الطريقة التسلسلية (المحرف الأول في الموقع الأول والمحرف الثاني في الموقع الثاني وهكذا....) فبعد أن تتم قراءة المحرف الأول سيمحى من موقعه الأول وتتم إزاحة بقية المحارف الأخرى.

على الرغم من أن هذه الطريقة تعمل بشكل جيد إلا أنها تنجز عملا لا حاجة له بالنسبة للمعالج لأنه في كل مرة يقرأ فيها محرف ما, يجب أن تتحرك كل المحارف الأخرى في الـ buffer, لذلك سيستخدم الـ buffer الحلقي مؤشران: الأول يشير إلى الموقع الذي سيقرأ منه المحرف التالي والثاني يشير إلى المكان الذي سيكتب فيه المحرف التالي الجديد. وبهذه الطريقة ستتحرك المؤشرات بدلا من المحتويات الداخلية للـ buffer.

في البداية يشير كلا من هذان المؤشران إلى بداية المسجل وسيتحركان باتجاه نهايته مع كل قراءة وكتابة لمحرف ما. عندما يصل كلا المؤشران إلى نهاية الـ buffer ستتم إعادتهما إلى بداية الـ buffer. ومن حركة المؤشر الدائرية يأتي اسم الـ buffer الحلقي.

مثال: افترض أن عنوان إزاحة المحرف التالي الذي سيقرأ من buffer لوحة المفاتيح هو (0x1AH) حالما تتم قراءته يتحرك المؤشر القراءة بايتين باتجاه نهاية الـbuffer أي ستصبح قيمته 0x1C .



الشكل101

يحدث نفس الشيء مع المؤشر الكتابة (1CH) الذي يشير إلى الموقع التالي للمحرف الأخير في الـbuffer.فإذا ضغط المستخدم مفتاح آخر سيخزن في هذا الموقع عندئذ يتحرك المؤشر بايتين باتجاه نهاية الـbuffer فإذا كانت قد خزنت ضغطة مفتاح جديدة في آخر word من الـbuffer عندها سيعود المؤشر ليشير إلى بداية الـbuffer.

إذن كل محرف يحتاج بايتين ليخزن في buffer لوحة المفاتيح , البايت الأول للـ ASCII CODE والثاني للـ SCAN CODE.

إن العلاقة بين مؤشري هذا الـ buffer تبين حالته التي تتحدد بالشرطين التاليين:

إذا كان لكلا المؤشرين نفس القيمة, هذا يعني أن مسجل لوحة المفاتيح فارغ.

إذا كان مؤشر النهاية يحاول أن يشغل نفس مكان مؤشر البداية, هذا يعني أن المخزن مليء.

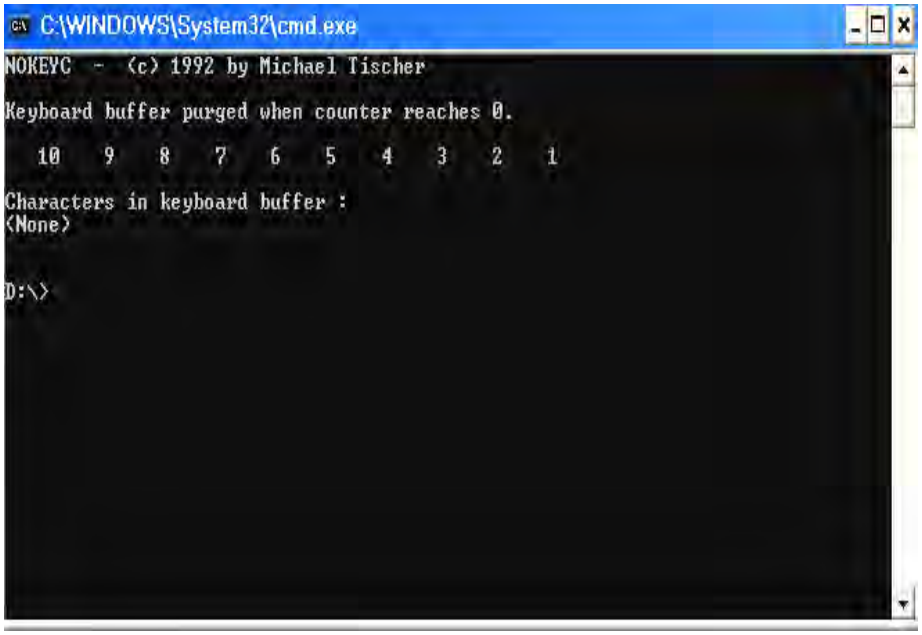
يتضمن buffer لوحة المفاتيح 32byte وبما أن كل محرف يحتاج بايتين ليخزن فيه, اذا يمكن للـ buffer تخزين 16 محرف هذا من أجل محرف الـ ASCII العادي, أما من أجل كود لوحة المفاتيح الموسعة سيكون الـ ASCII CODE مساويا للقيمة (0) لأن كود المحرف الفعلي يكون في البايت اللاحق.

و كما نجد في الجدول السابق, يتوضع buffer لوحة المفاتيح في عنوان الإزاحة (0X1EH).

مثال: NOKEY.C

هذا البرنامج يبين كيفية معالجة متحولات الـBIOS, ويستخدم تابع هام جدا لبرمجة لوحة المفاتيح الأساسية ولتنظيف محتوى buffer لوحة المفاتيح.

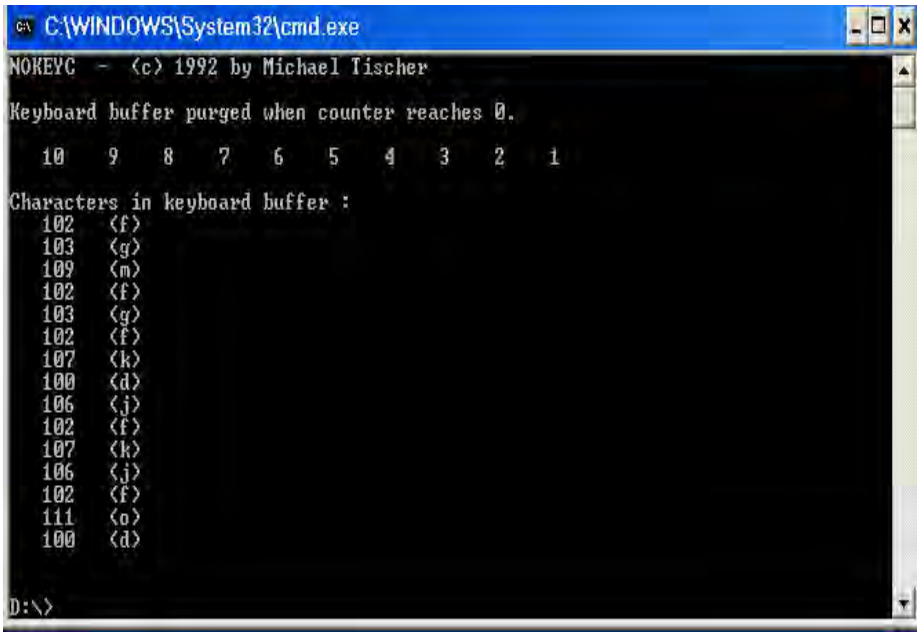
إذا كان برنامجك مثلا يحتاج لأن يجرب المستخدم إظهار رسالة حذف ملفات أو تهيئة أقراص وأنت لا تريد أية ضغطة مفتاح غير مقصودة تخزن في buffer لوحة المفاتيح, خلال عملية استعرض المستخدم لرسائل هذه العملية الخطيرة لأن المفتاح الخاطئ يعني أنه سوف يتم إبلاغ برنامجك أن يبدأ قبل أن يقرر المستخدم متابعة هذه العملية الخطيرة أم لا, و بالتالي لن يمكنك السيطرة على حدوث حدث غير طبيعي في لوحة المفاتيح و بالتالي هذا البرنامج سوف يؤمن لك عملية إيقاف عمل لوحة المفاتيح بمعنى آخر قتل كل ضغطة مفتاح قادمة من لوحة المفاتيح و ذلك بتفريغ buffer لوحة المفاتيح وهذه الوظيفة نراها مثلا في لغة دلفي عند وضع إسناد قيمة Null للمفتاح المضغوط حاليا.



الشكل102

أثناء تنفيذ هذا البرنامج إذا لم يتم إدخال أي محرف سيبقى الـ buffer فارغا, وبعد الانتهاء من كتابة الأعداد تنازليا من 10 إلى 1 ستكتب العبارة الموضحة بالشكل " characters in keyboard buffer: <None>".

أما لو ضغط أي مفتاح أثناء التنفيذ سيخزن هذا المفتاح في الـbuffer ويظهر بعد الانتهاء من كتابة الأعداد السابقة على الشاشة كما هو موضح:



الشكل103

10 2 -شفرات الضغط و التحرير

تتشكل شفرة المسح أيضا عند تحرير المفتاح و هذا يحدد للنظام إذا كان المفتاح ما يزال مضغوطا أو تم تحريره للتو و هذا مهم جدا لأنها الطريقة الوحيدة التي يستطيع فيها الكومبيوتر أن يترجم و بشكل صحيح الحالة عندما يتم ضغط أكثر من مفتاح في وقت واحد. و بدون هذه الخاصة لن يكون بالإمكان تنفيذ مهام خاصة مثل طباعة الأحرف الكبيرة أو إعادة تشغيل الكومبيوتر بواسطة CTRL+ALT+DEL.

يستخدم النظام ما يسمى شفرات الضغط و التحرير للتمييز بين شفرات المسح للمفاتيح المضغوطة (الضغط) و المفاتيح التي تحررت (التحرير), و كل شفرة من هذه الشفرات هي بحجم 1Byte. الفرق الوحيد بين شفرة الضغط و التحرير. هو البت 7 الذي يكون بقيمة 1 في شفرة التحرير و هذا يؤدي إلى أمرين مهمين:

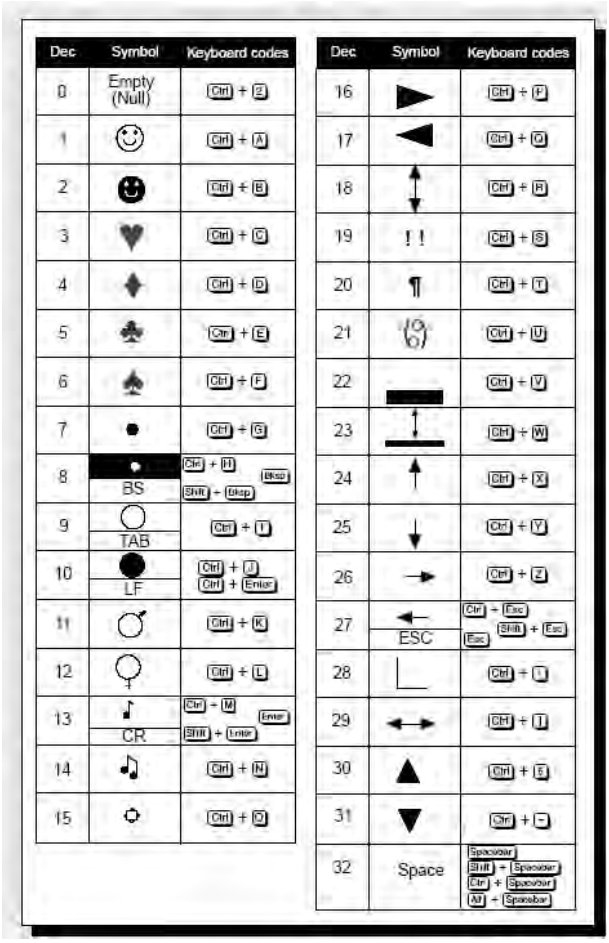
الأول: شفرات التحرير دائما أكبر من 128 و شفرات الضغط أصغر منها.

الثاني: لا يمكن للوحة المفاتيح أن تحوي أكثر من 128 مفتاح و إلا سوف تتجاوز شفرات الضغط شفرات التحرير.

المثال الأكثر وضوحا على ضرورة وجود شفرتي مسح مختلفتين لكل مفتاح هو ضغط أكثر من مفتاح معا فمثلا لكتابة الحرف الكبير A على المستخدم أن يستمر بالضغط على SHIFT اليميني و بعد ذلك الضغط على A. إن متحكم لوحة المفاتيح سيمرر شفرة الضغط لـSHIFT وهي (36H) ثم شفرة التحرير للحرف A و هي (1EH) للكومبيوتر و طالما لم يتسلم النظام شفرة القطع لـ SHIFT يفترض أن المفتاحين مضغوطان معا و يولد حرف كبير بدلا من الصغير.

10 3 -شفرات التحكم

كما نعلم سابقا أنه أي محرف من محارف الـ ASCII يمكن أن يتم إدخاله من خلال لوحة المفاتيح باستخدام مفتاح الـ ALT و مفاتيح لوحة المفاتيح الرقمية كما أنه من الممكن استخدام مفتاح الـ CTRL لتلك الوظيفة أيضا و ذلك في نظام التشغيل DOS. وعند استخدام هذه المفاتيح فإنه يمكن إظهار أحرف الـ ASCII التي شفرتها أصغر من الـ 32 و الشكل التالي يبين ما هي هذه المفاتيح و شفرة المسح الموافقة لها ووظيفتها. و هذا الجدول يكون عادة محمل بشكل قياسي في ذاكرة الـ BIOS.



Dec	Symbol	Keyboard codes
0	Empty (Null)	Ctrl + 2
1	☺	Ctrl + A
2	☻	Ctrl + B
3	♥	Ctrl + C
4	♦	Ctrl + D
5	♣	Ctrl + E
6	♠	Ctrl + F
7	•	Ctrl + G
8	BS	Ctrl + H, Shift + Bksp, Bksp
9	TAB	Ctrl + I
10	LF	Ctrl + J, Ctrl + Enter
11	♂	Ctrl + K
12	♀	Ctrl + L
13	CR	Ctrl + M, Shift + Enter, Enter
14	♫	Ctrl + N
15	☼	Ctrl + O
16	►	Ctrl + P
17	◄	Ctrl + Q
18	↕	Ctrl + R
19	‼	Ctrl + S
20	¶	Ctrl + T
21	§	Ctrl + U
22	▬	Ctrl + V
23	↨	Ctrl + W
24	↑	Ctrl + X
25	↓	Ctrl + Y
26	→	Ctrl + Z
27	ESC	Ctrl + Esc, Esc, Shift + Esc
28	∟	Ctrl + \
29	↔	Ctrl +]
30	▲	Ctrl + 6
31	▼	Ctrl + -
32	Space	Spacebar, Shift + Spacebar, Ctrl + Spacebar, Alt + Spacebar

الشكل 104

10 4 - شفرة الـ ASCII

لدى تنفيذ المقاطعة 16H التي سنأتي على ذكرها لاحقا يتم إعادة قيمة في المسجل AL فإذا كانت هذه القيمة مختلفة عن الصفر فهي قيمة شفرة الـ ASCII ونجد الـ SCAN CODE في المسجل AH و ذلك للمفتاح المفتعل حاليا, و هنالك بعض الاختلافات بالنسبة لمفاتيح التحكم وعلى سبيل المثال فإن معظم البرامج تترجم الرمز 27 على أنه تعليمة هروب وليس محرف نصي و لكن في حال رغبتنا بقراءة نص من هذا المفتاح. و لتحقيق ذلك يتم استخدام مفتاح وظيفي F1 مثلا للإشارة إلى رغبتنا في استخدام هذا المفاتيح في النصوص. فلدى الضغط على مفتاح F1 يتم وضع قيمة في متحول منطقي للإشارة إلى رغبتنا في عمل مفتاح الهروب 27 كمفتاح نصي ويتم إضافة تابع لبرنامج خدمة المقاطعة 16 لفحص هذا المتحول وتغيير مسار عمل البرنامج وفقا لهذا المتحول. و هذه العملية سنراها كثيرا في استخدام المفاتيح الوظيفية.

10 5 -شفرات لوحة المفاتيح الموسعة

تقدم توابع BIOS شفرات لوحة المفاتيح الموسعة أيضا و لكن في هذه الحالة نجد 00 في المسجل AL و قيمة الـ CANE CODE في المسجل AH و لذلك فإنه عند استدعاء أحد

التوابع 01H, 00H الخاصة بالمقاطعة 16H للـ BIOS سيجد البرنامج المستدعي القيمة 00 في AL و بهذه الطريقة نستخدم 128 شفرة جديدة.

جدول14

ASCII control codes on PCs			
Code	Meaning	Code	Char.
8	Backspace	BS	
9	Tab	TAB	
10	Linefeed (Ctrl + Enter)	LF	XXX
13	Carriage Return	CR	
27	Escape	ESC	

جدول15

Extended key codes				
Code (hex)	Code (dec)	Key(s)		
0FH	15	Shift	Tab	
10H	16	Alt	Q	(2nd keyboard series)
11H	17	Alt	W	
12H	18	Alt	E	
13H	19	Alt	R	
14H	20	Alt	T	
15H	21	Alt	Y	
16H	22	Alt	U	
17H	23	Alt	I	
18H	24	Alt	O	
19H	25	Alt	P	
1EH	30	Alt	A	(3rd keyboard series)

جدول16

Extended key codes			
Code (hex)	Code (dec)	Key(s)	
1FH	31	Alt S	
20H	32	Alt D	
21H	33	Alt F	
22H	34	Alt G	
23H	35	Alt H	
24H	36	Alt J	
25H	37	Alt K	
26H	38	Alt L	
2CH	44	Alt Z	(4th keyboard series)
2DH	45	Alt X	
2EH	46	Alt C	
2FH	47	Alt V	
30H	48	Alt B	
31H	49	Alt N	
32H	50	Alt M	
03BH	59	F1	
3CH	60	F2	
3DH	61	F3	
3EH	62	F4	
3FH	63	F5	
40H	64	F6	
41H	65	F7	
42H	66	F8	
43H	67	F9	
44H	68	F10	
47H	71	Home	
48H	72	↑	
49H	73	Page Up	
4BH	75	←	
4DH	77	→	
50H	80	↓	
51H	81	Pg Dn	
52H	82	Insert	
53H	83	Del	
54H	84	Shift F1	
55H	85	Shift F2	

جدول17

Extended key codes			
Code (hex)	Code (dec)	Key(s)	
56H	86	Shift F3	
57H	87	Shift F4	
58H	88	Shift F5	
59H	89	Shift F6	
5AH	90	Shift F7	
5BH	91	Shift F8	
5CH	92	Shift F9	
5DH	93	Shift F10	
5EH	94	Ctrl F1	
5FH	95	Ctrl F2	
60H	96	Ctrl F3	
61H	97	Ctrl F4	
62H	98	Ctrl F5	
63H	99	Ctrl F6	
64H	100	Ctrl F7	
65H	101	Ctrl F8	
66H	102	Ctrl F9	
67H	103	Ctrl F10	
68H	104	Alt F1	
69H	105	Alt F2	
6AH	106	Alt F3	
6BH	107	Alt F4	
6CH	108	Alt F5	
6DH	109	Alt F6	
6EH	110	Alt F7	
6FH	111	Alt F8	
70H	112	Alt F9	
71H	113	Alt F10	
73H	115	Ctrl ←	
74H	116	Ctrl →	
75H	117	Ctrl End	
76H	118	Alt	
77H	119	Alt Home	
78H	120	Alt 1	(1st keyboard series)
79H	121	Alt 2	
7AH	122	Alt 3	

جدول18

Extended key codes			
Code (hex)	Code (dec)	Key(s)	
7BH	123	Alt 4	
7CH	124	Alt 5	
7DH	125	Alt 6	
7EH	126	Alt 7	
7FH	127	Alt 8	
80H	128	Alt 9	
81H	129	Alt 0	
82H	130	Alt	
83H	131	Alt '	

10 6 -مقاطعات لوحة المفاتيح

يقدم نظام الإقلاع الأساسي مجموعة من برامج تخديم الأجهزة المحيطية الأساسية مثل لوحة المفاتيح و الفأرة, و هذه البرامج يتم تنفيذها لدى حدوث المقاطعات و بالتالي هي برامج لتخديم المقاطعات.

بالنسبة للوحة المفاتيح فإن نظام الإدخال والإخراج الأساسي يقدم لنا برنامجي تخديم مقاطعة للوحة المفاتيح:

الأول 0x09H: برنامج تخديم المقاطعة الصلبة للوحة المفاتيح IRQ1 و التي تحدث لدى كل ضغطة مفتاح على لوحة المفاتيح. و هذه المقاطعة عنوانها في جدول أشعة المقاطعات 0x09H.

الثاني 0x16: برنامج تخديم المقاطعة البرمجية للوحة المفاتيح و التي تستخدم لقراءة المحارف و حالة لوحة المفاتيح و يتم مناداتها من قبل المستخدم من خلال تعليمة لغة التجميع التالية:

INT 0x16H;

10 6 1 -برنامج خدمة المقاطعة 0x09H

و يخدم هذا البرنامج مقاطعة لوحة المفاتيح IRQ1 و التي تحدث لدى كل ضغط أو تحرير مفتاح على لوحة المفاتيح. وهذا البرنامج كما ذكرنا هو من البرامج القياسية لنظام الدخل و الخرج الأساسي BIOS. ووظيفة هذا البرنامج هي عملية استقبال المحارف المضغوطة على المنفذ 60H وتخزينها في مكان buffer لوحة المفاتيح الموجودة في العنوان 0x0040H:0x0017H

حيث تقدح المقاطعة الصلبة IRQ1 في كل مرة ترسل فيها لوحة المفاتيح شفرة ضغط أو تحرير إلى الكومبيوتر و هذا بدوره يستدعي المقاطعة 09H التي فيها الروتين المسئول عن تخديم عملية استقبال شفرات الضغط والتحرير و يحولها إلى شفرات ASCII الموافقة و التي تحدد موقع شكل الحرف في ذاكرة كرت الشاشة في Plain2 في جدول المحارف Character Table المحدد (راجع فصل الـ Graphic) التي يمكن قراءتها من قبل التطبيق الحالي.

هنالك عدة مهام يجب إنجازها قبل أن يتمكن التطبيق من قراءة المفاتيح:

الأولى: يجب على برنامج خدمة لوحة المفاتيح أن يقرأ شفرة الضغط أو التحرير من لوحة المفاتيح مباشرة باستخدام منفذ دخل\خرج و يكون عنوان هذا المنفذ عادة 61H و يقرأ من هذا المنفذ شفرات الضغط و التحرير فقط.

وكما رأينا ليست كل نتائج الضغط على المفاتيح هي مرئية على الشاشة فمثلا الحرف A لم يظهر إلا بعد الضغط على الحرف نفسه و حالما يتعرف برنامج تخديم لوحة المفاتيح على

المحرف المدخل يقوم بتحويل المحرف إلى الشفرة التي يستطيع التطبيق الحالي فهمها, حيث أن شفرات المسح بحد ذاتها غير مستخدمة لأنها ليست قياسية و غير مفهومة.

و يتم قراءة شفرة المسح من لوحة المفاتيح كما يلي:

يتم قراءة المنفذ 60H
يتم التأكد من أنه تم وصول كامل الشفرة وانتهاء معالج لوحة المفاتيح من إرسال الحرف المضغوط من خلال قراءة قيمة المصافحة على المنفذ 61H.
و من ثم يتم إرسال قيمة المصافحة عبر المنفذ 61H إلى لوحة المفاتيح لإخبار لوحة المفاتيح باستلام المحرف المضغوط.

و يجب أن يتم حفظ الأعلام قبل عملية القراءة و إيقاف مقاطعة لوحة المفاتيح مؤقتا ريثما يتم استلام المحرف و التأكد من وصوله.

```
local_irq_save( flags) ;
code = inportb(KEYB_PORT);              // Get scan code
val =  inportb(KEYB_ACK);               // Get keyboard acknowledge
outportb(KEYB_ACK, val | 0x80);         // Disable bit 7
outportb(KEYB_ACK, val);                // Send that back
local_irq_restore( flags ) ;
```

الثانية: التحويل من شفرة المسح إلى شفرة الـ ASCII, تحول شفرة المسح إلى شفرة الـ ASCII و التي هي الشفرة القياسية في كل الحواسب ,حيث تتألف شفرة الـ ASCII من 255. المستخدم منها عادة هو أول 128 حرف و الباقي يستفاد منه في عملية تحميل محارف للغات أخرى كما سنرى ذلك لاحقا.

لا يمرر محرف الـ ASCII المحول مباشرة إلى التطبيق بل يخزن أولا في buffer خاص لهذا الغرض و هو buffer لوحة المفاتيح السابق الذكر. و بعد حشر المحرف في buffer الحلقي يتم تعديل مؤشر الكتابة لهذا buffer.

الثالثة: استقبال و معالج شفرات التحكم: و هذه الشفرات موضحة في الجدول التالي و هي ذات وظائف معروفة ويأتي دور برنامج خدمة المقاطعة هنا في تنفيذ وظائف هذه المفاتيح.

10 6 2 -برنامج خدمة المقاطعة 0x16H

إن برنامج خدمة هذه المقاطعة وظيفته هي الوصول إلىbuffer لوحة المفاتيح الحلقي له ثلاث وظائف فرعية هي:

الخدمة الأولى 0x00H: عندما تحدث المقاطعة 16H فإذا كان المحرف موجودا في الـBUFFER فإن يتم حذفه منها و تمريره إلى البرنامج المستدعي أما إذا كانت الـBUFFER فارغة يتم انتظار الإدخال ثم إعادة القيمة إلى البرنامج.

الخدمة الثانية 0x01H: يقوم هذا التابع أيضا بقراءة لوحة المفاتيح مثل التابع السابق و لكن على عكس التابع 00H لا يتم حذف المحرف من الـBUFFER و يتم إعلام البرنامج عن طريق علم التصفير(ZEROFLAG) فإذا كان مساويا الواحد لا يوجد محرف و في الحالة المعاكسة يحوي كل من AL,AH على معلومات عن المفتاح الذي تم ضغطه

الخدمة الثالثة 0x02H.: لهذا التابع مهمة مختلفة فهو يقوم بقراءة حالة مفاتيح التحكم حيث نضع 02H في المسجل AH و نجد شعاع الحالة في المسجل AL بعد الاستدعاء وعلى سبيل المثال إذا كان البت الثالث في الشعاع مساويا الواحد فإن مفتاح ALT مضغوط علما ان شعاع الحالة هذا مفيد في برامج TSR التي تفحص حالة المفاتيح باستمرار.

10 .7 -جداول شفرات المسح و مقابلاتها في شفرة الأسكي

الجدول التالي يوضح لنا شفرات الأسكي القياسية التي تقابل شفرات المسح القادمة من لوحة المفاتيح هذا الجدول ضروري حتى نستطيع ترجمة شفرات المسح إلى موافقاتها من شفرات الأسكي وذلك لدى عملنا بدون الإعتماد على مقاطعات نظام الإدخال و الإخراج الأساسي كما سنرى لاحقا.

19 جدول

Key	Scan Code
ESCape	1
1 2 3 4 5 6 7 8 9 0 - =	2-13
Backspace	14
Tab	15
Q W E R T Y U I O P [ ]	16-27
ENTER	28
CTRL	29
A S D F G H J K L ; ' `	30-41
SHIFT (left)	42
\	43
Z X C V B N M ,. /	44-53
SHIFT (right)	54
Print Screen	55
ALT (left)	56
SPACEBAR	57
CAPS LOCK	58
F1 to F10	59-68
NUM LOCK	69
SCROLL LOCK	70
7 8 9 (Numeric keypad)	71-73
gray -	74
4 5 6 (Numeric keypad)	75-77

Key	Scan Code
gray +	78
1 2 3 (Numeric keypad)	79-81
0 (Numeric keypad)	82
DELETE	83
SYSTEM REQUEST	84
Key	Scan Code
NUL character	3
SHIFT-TAB	15
ALT-Q,W,E,R,T,Y,U,I,O,P	16-25
ALT-A,S,D,F,G,H,J,K,L	30-38
ALT-Z,X,C,V,B,N,M	44-50
F1 to F10	59-68
HOME	71
UP ARROW	72
PAGE UP	73
LEFT ARROW	75
RIGHT ARROW	77
END	79
DOWN ARROW	80
PAGE DOWN	81
INSERT	82
DELETE	83
SHIFT-F1 to F10	84-93
CTRL-F1 to F10	94-103
ALT-F1 to F10	104-113
CTRL-PRINT SCREEN	114
CTRL-LEFT ARROW	115
CTRL-RIGHT ARROW	116
CTRL-END	117
CTRL-PAGE DOWN	118
CTRL-HOME	119
ALT-1,2,3,4,5,6,7,8,9,0,-,=	120-131
CTRL-PAGE UP	132

Key	Scan Code
F11, F12	133, 134

10 8 -خريطة لوحة المفاتيح Keymap

كما وجدنا في الفقرة السابقة أن لكل شفرة مسح هنالك شفرة أسكي مقابلة لها. و كما ذكرنا أن من مهام برنامج تخديم لوحة المفاتيح هي عملية ترجمة شفرة المسح المرسلة من قبل لوحة المفاتيح إلى شفرة مقروءة والتي هي شفرة أسكي, و يتم ذلك باستخدام مفهوم خريطة لوحة المفاتيح.

حيث أن هذه الخريطة عبارة عن جدول بحث LookupTable أو مصفوفة أحادية البعد عناصرها مرتبة حسب ترتيب شفرة المسح و قيمة كل عنصر من هذه المصفوفة مساوية لقيمة شفرة الأسكي المطلوبة.

لنأخذ على سبيل المثال الحرف A. لدى الضغط على هذا الحرف فإنه ترسل شفرة المسح ذات القيمة 0X61H و يأتي دور برنامج تخديم لوحة المفاتيح في ترجمة هذه القيمة إلى شفرة الأسكي 0X41H وتخزينها في مخزن الأحرف المؤقت Buffer. و لتحقيق هذه العمل نستخدم الخريطة التالية

```
Char   keymap = {0X0000 - - 0X6141 - - 0X5D}
```

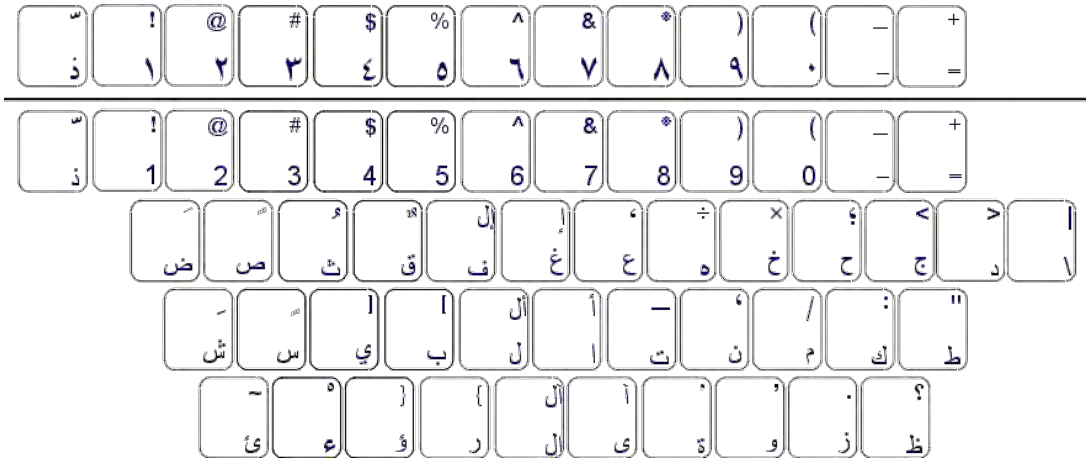
و هذه الخريطة مؤلفة من 93 خانة تحتوي على جميع شفرات المسح القادمة من لوحة المفاتيح. و سنعتمد على هذه الخريطة بشكل كبير لتنفيذ عملية ترجمة شفرات لوحة المفاتح إلى شفرة الأسكي الخاصة باللغة العربية كما سنأتي على ذلك لاحقا.

10 9 -استخدام لغات أخرى في لوحات المفاتيح

إحدى أهم التحديات التي واجهت مشروع بناء نظام التشغيل باللغة العربية هي عملية التعريب و سنأتي على تفاصيل هذا الموضوع في قسم التعريب, أما في هذه الفقرة فإننا سوف نتكلم عن دور لوحة المفاتيح في عملية التعريب.

فلكي تكتمل دورة تعريب النظام يجب أن يؤمن نظام التشغيل للمستخدم إمكانية إدخال الحروف العربية من خلال لوحة المفاتيح وذلك وفقا لتوزع المفاتيح العربية على لوحة المفاتيح كما موضح في الشكل التالي:

الشكل 105



الشكل106

و بالتالي من مهام المشروع هي القيام برسم خريطة للوحة المفاتيح العربية بحيث يتم ترجمة شفرة المسح للمفتاح القادم وذلك لرقم الأسكي المناسب للحرف العربي, و كما سنجد في قسم التعريب أن تحميل حروف الأسكي للغة العربية يبدأ من مجال 128 حتى 255 يتم فيها تحميل الشكل العربي للحرف حسب حالات ورود الحرف كما سنجد لاحقا في قسم التنفيذ.

10 10 -العمل في النمط المحمي

كما وجدنا في فصل نظام العمل في النمط المحمي, أن نظام المقاطعات BIOS لا يمكننا الاستفادة منه ولا يمكننا استخدامه, ولذلك نحن مضطرين لكتابة برامج شبيهة ببرامج تخديم لوحة المفاتيح (Handlers) الموجودة في الـ BIOS. و ذلك بما يوافق نظام عمل المقاطعات الجديد في النمط المحمي. و بالتالي كل ما سبق من حديث عن المقاطعات و وظائفها الجاهزة سوف نضعه جانبا, وسيقع علينا نحن عبء تنفيذ و برمجة هذه الوظائف من البداية. لا تقلق عزيزي القارئ فالانترنت و المصادر المفتوحة ستحل لك هذه المشكلة كل ماعليك هو الفهم الدقيق لكيفية تركيب المكاتب المفتوحة المصدر و توليفها مع نواة نظامك و ربطها بأشعة مقاطعاته الخاصة. و هذا ما سنجده لاحقا.

يـتم تنفيـذ ذلـك كمـا سـنجد فـي قسـم التنفيـذ ضـمن مـا يسـمى مشـغل لوحـة المفـاتيح Keyboard Driver. ويحتوي هذا المشغل على التوابع الأساسية للتعامل مع لوحة المفاتيح, و يتم كتابة هذا المشغل بحيث نتعامل مباشرة مع لوحة المفاتيح في المستوى الأدنى.

10 11- مشغلات الأجهزة المحيطية

إن الحواسب هي عبارة عن أجهزة غبية تنفذ ما تؤمر به دون مناقشة و لكنها تعترض في بعض الأحيان كما وجدنا في فقرة الاستثناءات و الاعتراضات في قسم أنماط عمل المعالجات, و يتم إدخال هذه الأوامر و إخراج نتيجة تنفيذه من خلال أجهزة محيطية. مثل الطابعة و لوحة المفاتيح و الشاشة, و هذه الأجهزة حتى يتم تشغيلها فإنه تتطلب برامج خاصة بها لتشغيلها وقيادة عملها و التحكم بها. و تكون هذه البرامج عادة (Drivers) جزء من نظام التشغيل و سنراها في نظامنا مجموعة ضمن مجلد الـ /os/arch/drivers/.

الآن سنخوض قليلا في هيكلية المشغل وتصميمه, فكما سبق وذكر أن نمط العمل المحمي لمعالجات 80386 يمنع المستخدم العادي الذي يعمل في المستوى 3, من الوصول المباشر إلى نواة النظام أو إلى الأجهزة المحيطية, و إنما يتم ذلك باستخدام مفهوم البوابات Gates و استدعاءات النظام System Call. و يتم الوصول إلى هذه الأجهزة من خلال واجهات مكتوبة ضمن برامج تشغيل هذه الأجهزة في النظام. بحيث تؤمن هذه الهيكلية العزل بين المستخدم و الأجهزة و كما تؤمن عزل المهام و منعها من التداخل وحصول عمليات تنازع على الأجهزة.

حيث يكون مشغل الجهاز موجود في مستوى نواة النظام و يحتوي على التوابع الأساسية للدخل والخرج التي تتعامل مع الجهاز المحدد, بالإضافة إلى واجهات Interfaces للمنح التطبيقات و المكتبات التي هي على مستوى المستخدم من التعامل مع هذه الجهاز المحيطي, ولكن يتم تغليف هذه الواجهات أو لنقل التعامل مع هذه الواجهات من خلال مناداة استدعاءات النظام system calls التي تقوم بدورها بطلب هذه الواجهات و استحصال ما تحتاجه مكاتب النظام من معلومات من الجهاز المحدد. و هذه الهيكلية كما قلنا تكون قد حققت لنا عمليات العزل المطلوبة للمستخدمين عن بعضهم البعض أو للمهام عن بعضها البعض. و الشكل التالي يوضح البنية الهيكلية وموقع المشغلات. و سنأتي في لاحقا على ذكر أمثلة عملية عن تحقيق هذه البنية من خلال لوحة المفاتيح أو بطاقة الإظهار.

جدول20

عمليات الدخل\الخرج على مستوى المستخدم
استدعاءات نظام التشغيل system call
برنامج مشغل الجهاز device driver
برنامج تخديم مقاطعة الجهاز interrupt handler
الجهاز المحيطي hardware

10 12 -قسم التنفيذ

الآن ننتقل لتطبيق الدراسة النظرية لما سبق عن لوحة المفاتيح لتطبيقه في نظام التشغيل. فكما لاحظنا من خلال الدراسة النظرية للوحة المفاتيح في نظام عمل النمط الحقيقي أن المطلوب في لوحة المفاتيح من خدمات في مجال عمل النظام هي الوظائف التالية:

تهيئة لوحة المفاتيح و ضبط المتحكم الخاص بها بالقيم المناسبة.
استقبال شفرات المسح من لوحة المفاتيح ومعالجتها.
تقديم واجهات لعمليات القراءة من لوحة المفاتيح.

إن آلية عمل برنامج تشغيل لوحة المفاتيح و سيناريو العمليات في البرنامج يتم على الشكل التالي وذلك بالإستفادة مما سبق من سرد لبعض المصطلحات والعناوين النظرية. و سنبدأ بسيناريو مشغل لوحة المفاتيح بدء من ضغط المستخدم للزر على لوحة المفاتيح و انتهاء بعرض المحرف المضغوط على شاشة العرض.

عندما يقوم المستخدم بالضغط على مفتاح في لوحة المفاتيح فإنه هذه الضغطة تؤدي لحصول تماس بين سلكي العقدة الموجود عليه المفتاح في مصفوفة أسلاك لوحة المفاتيح. و لدى حصول التماس فإنه سوف يؤدي إلى حصول جهد كهربائي على طرفي السلكين, وخلال هذا الجهد الحاصل على السلكين يقوم معالج لوحة المفاتيح.بتحديد موقع المفتاح المضغوط من خلال التعرف على السلكين و من ثم يقوم ببناء قيمة ثنائية حسب هذين السلكين تميز هذا المفتاح و هذ القيمة الثنائية التي تدعى شفرة المسح يتم إرسالها عبر منفذ PS2 مثلا بشكل تسلسلي متزامن على شكل 11بت 8 منها هي شفرة المسح و الباقي للمصافحة والمزامنة و التحقق من صحة الإرسال ويتحسس المعالج في الحاسب المتصل بلوحة المفاتيح بعملية الإرسال هذه من خلال قدح المقاطعة 0x09h و التي تقطع عمل المعالج مؤقتا ويقوم المعالج بتنفيذ برنامج خدمة المقاطعة هذه والذي يقوم باستقبال الشفرة المرسلة من لوحة المفاتيح على المنفذ 0x60h ويقوم البرنامج بإرسال المصادقة بوصول المحرف بشكل كامل وصحيح على المنفذ 0x61h. وكما يقوم برنامج خدمة مقاطعة لوحة المفاتيح باستقبال شفرة المسح و ترجمته لشفرة الأسكي المناسبة وذلك في حال كان المفتاح المضغوط هو مفتاح محرفي و إلا يتم فإنه يتم تنفيذ الوظيفة المرتبطة بالمفتاح في حال كان مفتاح وظيفي أو يتم تنفيذ أمر التحكم في حال كان مفتاح تحكم وفي حال المحارف يتم تخزين المحرف المضغوط في ذاكرة تخزين مؤقتة buffer بحجم 32 كلمة حيث كل كلمة يتم فيها تخزين قيمة الأسكي للمحرف و قيمة شفرة المسح, حيث البايت الأعلى أهمية يحوي على شفرة المسح بينما البايت الأقل أهمية يحتوي على شفرة الأسكي.

الآن بعد أن وصلت المحارف إلى الحاسب وتم تخزينها في ذاكرة الحاسب, أو لنقل في مكان مخصص للتطبيقات للقراءة منه, حيث انه في نظام التشغيل لدينا بنية تغلف عمل لوحة المفاتيح بالإضافة لبطاقة العرض على الشاشة تدعى بـ Consol و أحد اجزاء هذه البنية هي ذاكرة لوحة المفاتيح المؤقتة. و بالتالي لدينا لكل consol ذاكرة مؤقتة للوحة المفاتيح خاصة به. وبالتالي لدى ورود المقاطعة يقوم برنامج خدمة المقاطعة بتخزين المفتاح المضغوط في ذاكرة التخزين للـ Consol الحالي الفعال.

الآن بعد أن تم تخزين المحرف في ذاكرة التخزين المؤقتة فإنه سيتم قراءة هذا المحرف من خلال التطبيق الذي يعمل حاليا و المرتبط بالـ Consol.حيث تتم عملية القراءة من مخزن الـ Consol الحالي ويتم تمرير ناتج القراءة لتابع عرض المحارف على الشاشة.

و يتم تحقيق ماسبق من سيناريو في التوابع والواجهات التي سنأتي على ذكرها في الفقرة التالية.

10 12 1- تهيئة لوحة المفاتيح

كما وجدنا سابقا في الدراسة النظرية أنه لدى إقلاع نظام التشغيل يتم تهيئة لوحة المفاتيح من خلال إرسال أوامر إلى المتحكم بلوحة المفاتيح وفق قيم معينة.

حيث يجب أن نقوم في هذه العملية بالخطوات التالية:

تهيئة سرعة تواتر ضغطة المفتاح على التردد 30 و التأخير 0 أي أقل قيمة ممكنة للتأخير للوحة المفاتيح. و ذلك بإرسال القيم مباشرة إلى مسجلات معالج لوحة المفاتيح من خلال المنفذ 243.
إضاءة ليدات الإشارة على لوحةالمفاتيح بالقيم القياسية.
تنصيب مقاطعة لوحة المفاتيح. وربطها برقم المقاطعة المحددة 0X01.
و عملية التهيئة هذه تتم من خلال التابع init_keyboard في المكتبة keyboard.c مشغل لوحة المفاتيح.

10 12 2- استقبال شفرات المسح و معالجتها

إن المهمة الأساسية لمشغل لوحة المفاتيح هي عملية استقبال شفرة المسح من لوحة المفاتيح و معالجة هذه الشفرات وتحويلها لشفرات أسكي مفهومة من قبل المبرمج. و تمرير شفرات الأسكي الناتجة إلى مخزن لوحة المفاتيح المؤقت وتخزين هذه الشفرات أولا بأول وذلك في حال كان هنالك متسع في هذا المخزن الحلقي الذي سبق و تكلمنا عن بنيته سابقا.

إن مخزن لوحة المفاتيح المؤقت keyb_buffer هو عبارة عن مصفوفة بحجم 64 بايت أي يتسع لـ 32 محرف مضغوط, و يكون هذا المخزن عادة موجود في بنية تدعى بالكونسول سنأتي على تفصيلها لاحقا.

و يتم التحكم بهذا المخزن كما وجدنا في الدراسة النظرية سابقا من خلال متحولين:

متحول الكتابة في المخزن keyb_buf_write.
متحول القراءة من المخزن keyb_buf_count.

و في كل عملية إضافة للمخزن يتم زيادة متحول الكتابة و في كل عملية قراءة من المخزن يتم إنقاصه و ذلك لتحقيق البنية الحلقية للمخزن.

الآن نعود لعملية معالجة شفرات المسح, فكما وجدنا في الدراسة النظرية أن شفرات المسح هي نوعين

شفرات تحكم.
شفرات محارف.

شـفرات الـتحكم: و هـي مثـل شـفرات التـي تـتحكم بأضـواء الإشـارة الثلاثـة (Cpaslock, Numlock, Scrollock) حيث نقوم باستقبال هذه الشفرات و تمييزها و على أساسها يتم إضاءة أو إطفاء الأضواء من خلال إرسال الأوامر إلى متحكم لوحة المفاتيح. و هذه الأوامر يتم إرسالها من خلال تابع update_leds و ذلك بعد أن نقوم بتعديل قيم متحولات عامة في النظام تحديد قيمة هذه الأضواء الحالية و تكون هذه المتحولات العامة موجودة ضمن بنية الكونسول.

بالإضافة لشفرات التحكم الثلاث السابقة لدينا شفرات تحكم خاصة بتحديد الوضع الحالي لمفاتيح Shift و Control و Alt اليميني و اليساري ويتم تعديل قيم متحولات عامة أخرى اعتماد على شفرات الضغط لهذه المفاتيح. و يستفاد من هذه المتحولات كما في تحديد الوضع الحالي لهذه الأزرار فمثلا يلزمنا لتنفيذ عملية إعادة الإقلاع من خلال الضغط المتوالي على المفاتيح الثلاث Alt + Ctr + Del لتنفيذ هذه العملية فإنه يلزمنا تحديد الوضع الحالي لهذه المفاتيح هل هي في حالة ضغط أم تحرير و يتم ذلك على أساس فحص البت الثامن كما وجدنا في الدراسة النظرية و الكود التالي يعطينا مثال عن تنفيذ هذا الكلام.

و هذا الكود جزء من كود برنامج تخديم لوحة المفاتيح الموجود ضمن التابع keyboard_handler في مكتبة مشغل لوحة المفاتيح Keyboard.c.

```
switch (keypressed)
{
      case 42:                    // LShift
   shift=1;
        break;
      case (42+128)
        shift=0;
        break;

      case 54:                    // RShift
        shift=1;
        break;
      case (54+128)
        shift=0;
        break;
            case 56:                    // ALT
        alt=1;
        break;
            case (56+128):
        alt=0;
        break;
            case 29:                            // CTRL
        ctrl=1;
        break;
      case (29+128):
        ctrl=0;
        break;
      case 58:                          // CAPS LOCK
        curr_cons->caps_lock ^= 1;
        update_leds();
        break;
      case 69:                          // NUM LOCK
        curr_cons->num_lock ^= 1;
        update_leds();
        break;
      case 70:                          // SCROLL LOCK
        curr_cons->scroll_lock ^= 1;
        update_leds();
        break;
}
```

الآن الصنف الثاني هي شفرات المسح للمحارف. و هذه الشفرات هي التي يقوم بمعالجتها برنامج تخديم لوحة المفاتيح و يقوم بتحويل شفرات المسح إلى شفرات أسكي موافقة ومن ثم يتم تمرير هذه الشفرات إلى مخزن لوحة المفاتيح المؤقت. و كما نوهنا سابقا إلى أن عملية ترجمة شفرات المسح إلى شفرات أسكي تتم وفق ما يسمى خريطة لوحة المفاتيح Keymap. و نحن لدينا في نظام التشغيل ثلاث خرائط للوحة المفاتيح و ذلك بالنسبة للغة الانكليزية هي كالتالي:

الخريطة لطبيعية للوحة المفايتح: و هي الخريطة التي تحوي على شفرات الأسكي للوحة المفاتيح و هي طبعا تكون فقط للمحارف الانكليزية

خريطة لوحة المفاتيح في حال ضغط المفتاح Alt و هي تحتوي على شفرات الأسكي للمحارف التي يجب أن تظهر لدى الضغط على مفتاح Alt

خريطة لوحة المفاتيح في حال ضغط المفتاح Ctrl و هي مشابهة لما قبلها في حال ضغط المفتاح Ctrl.

لدينا أيضا خرائط للوحة المفاتيح باللغة العربية المستخدمة في التعريب. و لدنيا للغة العربية أربع أنوع من الخرائط هي كالتالي:

خريطة محارف اللغة العربية

فكما نعلم أن الحروف في العربية هي حروف متصلة ببعضها البعض. و بالتالي شكل الحرف الواحد يختلف حسب موقعه من الكلام. وفرضنا أن كل حرف له أربع أشكال محتملة حسب موقعه من الكلام. و بالتالي قمنا بإعطاء كل حرف تقريبا أربع شفرات أسكي مختلفة يتم اختيار الشكل المناسب للحرف وفق خوارزمية معينة و ذلك تبعا لما قبل الحرف.

و بالتالي بالإضافة لفكرة عملية ترجمة شفرة المسح القادمة إلى شفرة أسكي موافقة للغة العربية يأتي دور برنامج تخديم لوحة المفاتيح في عملية اختيار الشكل المناسب للحرف العربي و ذلك وفقا لخوارزمية معينة.

الآن و بعد استحصال شفرة الأسكي الموافقة لشفرة المسح القادمة من لوحة المفاتيح. يتم تحميل هذه حشر شفرة الأسكي هذه وشفرة المسح و التي تكون في كلمة واحدة word في المخزن المؤقت للوحة المفاتيح.

وبرنامج خدمة المقاطعة هذا الذي يقوم مقام برنامج خدمة المقاطعة 0x09H في نظام الدخل والخرج الأساسي. و هو عبارة عن تابع موجود في مكتبة Keyboard.c واسم هذا التابع Handler, ويتم ربط هذا التابع مع واصف المخصص له في جدول IDT (راجع النمط المحمي) لدى تهيئة لوحة المفاتيح ضمن التابع Init_keyboard ضمن مكتبة مشغل لوحة المفاتيح أيضا.

10 12 3 -واجهات لعمليات القراءة من لوحة المفاتيح

وجدنا في القسم النظري سابقا أن نظام الدخل والخرج الأساسي BIOS يقدم لنا مقاطعة برمجية وهي المقاطعة 0x16H وكانت وظيفة هذه المقاطعة هي القيام بعملية القراءة من لوحة المفاتيح بالإضافة لقراءة حالة اللوحة. وكنا نقوم بتغليف هذه المقاطعات في توابع سهلة وبسيطة لتنفيذ عمليات القراءة من لوحة المفاتيح.

الآن دورنا في هذه المرحلة هي القيام بنفس الشيىء ولكن سنبدأ من الصفر حيث سنقوم بكتابة توابع المقاطعة و استدعاء هذه التوابع من خلال مكتبات برمجية أعلى على مستوى المستخدم للحصول على دخل من لوحة المفاتيح.

و بالتالي فإنه لدينا الآن مستويين من واجهات القراءة من لوحة المفاتيح:

10 12 4 -واجهات على مستوى النواة

الواجهات على مستوى النواة وهي التوابع التي تقرأ من لوحة المفاتيح مباشرة وموجودة في مشغل لوحة المفاتيح وهي التابعين التاليين:

keyb_read(); :و هو شبيه بالتابع 0x01H في برنامج خدمة المقاطعة 0x16H.
وهو يقوم بقراءة لوحة المفاتيح مهما كانت الحالة لمخزن لوحة المفاتيح و في حال كانت ذاكرة لوحة المفاتيح فارغة فإنه يتم يرجع القيمة -1..

kgetchar(); :وهو مثل الخدمة 00 في برنامج خدمة المقاطعة 0x16H, وهو يعيد المحرف الذي تم ضغطه في حال كانت مخزن لوحة المفاتيح ممتلئ وفي حال كانت المخزن فارغا فإنه يقوم بالانتظار على رتل الانتظار.

10 12 5 -واجهات على مستوى المستخدم

إن عمليات القراءة من لوحة المفاتيح والتي نراها في العادة في مكاتب لغات البرمجة مثل C و الـ Pascal تكون في العادة عبارة عن واجهات لعمليات القراءة على المستوى الأدنى في مشغل لوحة المفاتيح. وهذه الواجهات هي التالية:

scanf
getchar
gets
getline

حتى نتمكن من الفهم الصحيح للهيكلية المكاتب التي تتعامل مع لوحة المفاتيح فإننا سوف نستعرض هذه المكتبات ووظائفها ومن ثم سنشرح توضع هذه المكتبات و

Getchar- 1 5 12 10

يستخدم هذا التابع لقراءة محرف من لوحة المفاتيح. وهو عبارة عن استدعاء نظام system call لتابع kgetchar ضمن مكتبة KeyboardDriver.

getline- 2 5 12 10

يقوم هذا التابع بقراءة مجموعة من المحارف يحدد عددها من خلال حجم المصفوفة الممرر في البارامتر الثاني من التابع. ويستخدم عادة هذا التابع في مكتبة محث الأوامر shell لقراءة الأوامر.

scanf- 3 5 12 10

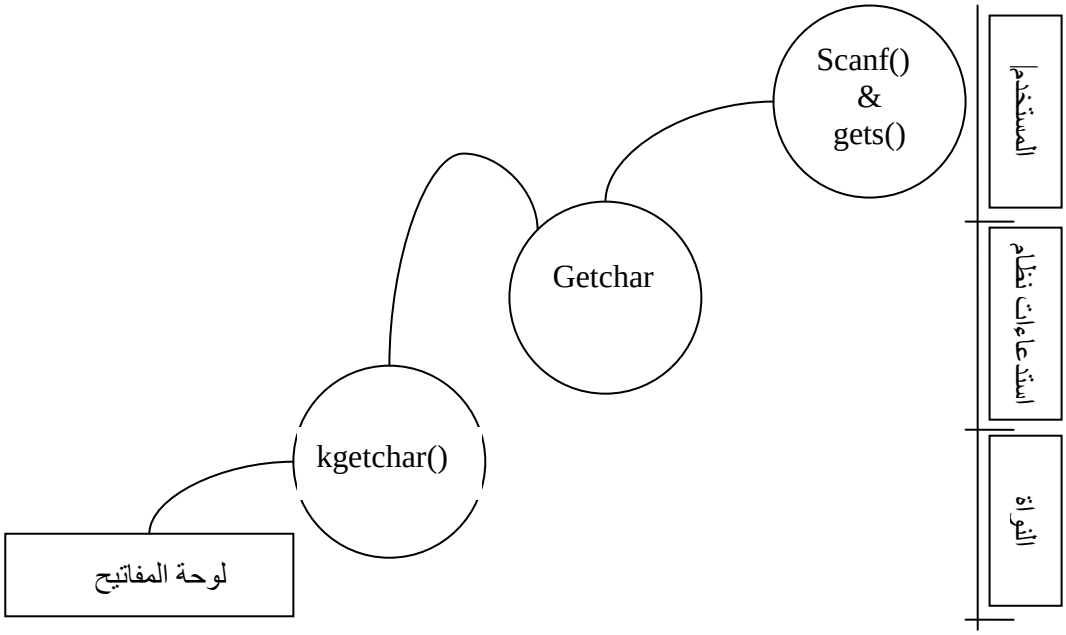
وهو عبارة عن واجهة تستخدم لعملية قراءة المعطيات والقراءة المنسقة من لوحة المفاتيح وهي تقوم بنفس وظيفة تابع لغة C القياسية.

Gets- 4 5 12 10

يستخدم هذا التابع لقراءة الدخل من لوحة المفاتيح طالما يقوم المستخدم بالضغط على لوحة المفاتيح وتستمر في قراءة الدخل حتى طول 1024 محرف.

الآن ترتيب هيكلية هذه المكتبات و استدعائها من مستوى إلى آخر. تتم بدء من مكتبة scanf و هذه تتم على مستوى المستخدم و تقوم باستدعاء تابع gets الذي بدوره يقوم بطلب استدعاء النظام getchar الذي يقوم بقراءة محرف. و كذلك الآمر بالنسبة لتابع getline الذي يقوم باستدعاء تابع getchar مباشرة.

و نجد ذلك موضحا في الشكل التالي:



10 12 6 -المكاتب المعتمدة في تخديم لوحة المفاتيح.

لتحقيق ماسبق من مشغل لوحة المفاتيح وواجهات التعامل مع لوحة المفاتيح, قمنا بتوظيف العديد من الخدمات التي يتيحها لنا نظام التشغيل و نواة النظام. وهذه الخدمات سنقوم بذكرها سريعا من خلال سرد أسماء المكاتب المستخدمة في كل من مشغل لوحة المفاتيح و الواجهات التي تم ذكرها و سنقوم بتوضيح دور هذه المكتبات في هذه الواجهات.

التوابع المستخدمة	المكتبة	الغاية منها
Install_trap_handler() Local_irq_save(); Local_irq_restore();	Interrupt.h	و يتم استخدامها لربط المقاطعة الصلبة 9 مع برنامج تخديمها. والتابع المرتبط بهذه المكتبة هو التابع
Struct Conol_t	Consol.h	و يتم استخدام هذه المكتبة للتعامل مع consol المهمة الحالية و التعامل مع الـ buffer المتعلق بها.
Inportb() Outbortb();	I386.h	نستخدم منها توابع الإرسال و الاستقبال مع مسجلات لوحة المفاتيح
	Font.h	و هي المكتبة الخاصة باللغة العربية
DECLARE_WAITQUEUE	Queue.c	تستخدم لتزودنا برتل انتظار المهام لجهاز لوحة المفاتيح, حيث يتم تعريف رتل انتظار خاص بلوحة المفاتيح و كل

		ما تم طلب مشغل لوحة المفاتيح او برنامج خدمة مقاطعته فإنه يتم وضع المهمة التي طلبت ذلك على الرتل في حال كانت مهمة أخرى تطلب الجهاز.
wakeup_queue() wait_event()	sched.h	تستخدم هذه المكتبة لتأمين عمليات حماية المهام من الوصول المشترك إلى جهاز لوحة المفاتيح و ذلك بالاستفادة من الرتل الذي تم تعريفه الخاص بهذا الجهاز.
Udelay()	delay.h	تستخدم هذه المكتبة لتأمين عمليات التأخير الزمني المطلوبة للتعامل مع معالج لوحة المفاتيح وذلك لتحقيق التزامن في الإرسال و الاستقبال.

11 -بطاقة العرض

مُقدِّمة

قدمت السنوات الأخيرة تطورات كبيرة في تقنيات الكيان الصلب للحاسب، حيث ازدادت سرعة المعالجات المستخدمة، كما تطورت بطاقات الشاشة بشكل واضح من حيث الدقة العالية للإظهار والطيف اللوني الأوسع والأشمل.

سنوضح في هذه الفقرة تطور أنماط بطاقات الشاشة من الأقدم الى الأحدث:

11 -1 بطاقة العرض أحادي اللون Monochrome Display Adapter (MDA)

هي أقدم نوع محول رسوميات قابل للاستخدام في الحاسب الشخصي. كان هذا النمط هو القياسي أو المعياري عندما أصدرت شركت IBM أول حاسب شخصي عام 1980.تدعم هذه البطاقة نمط عمل وحيد، وهو نمط نصي مؤلف من 80 عمود و 25 سطر. كما تحوي حجم صغير من ذاكرة الإظهار Video RAM، لذلك لا تستطيع تخزين سوى صفحة واحدة في الذاكرة.قليل من أجهزة الكمبيوتر تستخدم هذا النوع من بطاقات الشاشة في هذه الأيام، كما أن IBM أوقفت انتاجها منذ عدة سنوات .

11 -2 بطاقة الرسوميات اللونية Color/Graphics Adapter (CGA)

ان معيار CGA ظهر الى الوجود في اوائل عام 1981. اصبح هذا النمط، القادر على عرض الرسوميات، بديلا أساسيا عن بطاقات MDA. كما بطاقات MDA، تستطيع بطاقات CGA العمل بنمط عرض نصي مكون من 80 عمود و 25 سطر. بالإضافة الى قدرتها على العمل وفق نمط رسومي بدقة 320 × 200 بكسل بأربع ألوان، و بدقة 600×200 بكسل بلونين.على الرغم من الاختلاف بين بطاقات CGA و MDA، إلا أنها تعتمد على نفس نوع متحكم الإظهار Motorola MC6845.

11 -3 بطاقات هرقل الرسومية Hercules Graphics Card (HGC)

تعتمد هذه البطاقات على المتحكم Motorola MC6485 في عملها، وهي متوافقة تماما مع بطاقات MDA. تستطيع هذه البطاقة عرض صفحتين رسوميتين بدقة 720× 348 بكسل. تجمع بطاقات HGC بين وثوقية بطاقات MDA مع الإظهار الرسومي لـ CGA.

11 4 -محول الرسوميات المطور
Enhanced Graphics Adapter (EGA)

تم إصدار هذا النمط من محولات العرض في عام 1985، وقد قدمت ثورة في مجال الحواسيب الشخصية. تملك هذه المحولات نمط اظهار خاص بها، بالاضافة لكونها متوافقة تماما مع أنماط CGA و MDA. تعتبر EGA أول بطاقة عرض قادرة على التعامل مع كلا الشاشات أحادية اللون و الشاشات الملونة.

تكون بطاقات EGA أكثر فعالية عندما تستخدم مع شاشة EGA، وهي مشابهة لشاشة CGA ما عدا أن دقة نمط الرسوميات فيها أكبر (640×350) و خيارات لونية أكثر. كما أن بطاقات EGA تحوي حجم أكبر من ذاكرة العرض مما يتيح لها امكانية تخزين أكثر من صفحة في نفس الوقت.

11 5 -نظام رسوم الفيديو
Video Graphics Array (VGA)

ظهر هذا النمط من بطاقات الشاشة في عام 1987 , حيث جمع بين تقنيات جديدة مع تلك الموجودة في بطاقات EGA.وقد حافظ بذلك على التوافق مع التقنيات السابقة، و فتح المجال لاستخدام ألوان أكثر ودقة أكبر. أهم ما تتميز به VGA عن سابقتها EGA هو وجود تكامل منطقي كامل على شريحة واحدة. على العكس من EGA، فإن هذه البطاقات ترسل اشارات لونية تشابهية الى الشاشة عوضا عن الاشارات الرقمية. أكبر دقة ممكنة في بطاقات VGA هي 640×480 بكسل. لذا فإنها تحوي على ذاكرة عرض بحجم 256 كيلوبايت كحد أدنى، يمكن زيادتها حتى 512 كيلوبايت.

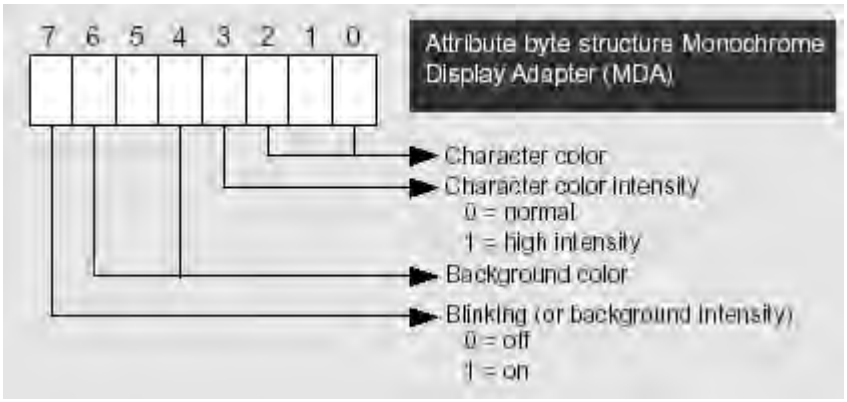
11 6 -نظام الدخل و الخرج القياسي الخاص بالفيديو
The VIDEO BIOS

تقوم BIOS بعدة مهام لإنجاز مختلف خدمات العرض على الشاشة، تجمع هذه الخدمات كتوابع فرعية للمقاطعة رقم 10H.لا يمكن تطبيق توابع المقاطعة رقم 10H الا على بطاقات MDA و CGA. أي أن BIOS الإفتراضي لا يدعم بطاقات EGA و بطاقات VGA، لذلك تقوم هذه البطاقات بتضمين توسيع لـBIOS خاص بها على شريحة ذاكرية للقراءة فقط. يتم تفعيل هذا التوسيع عند اقلاع الحاسب.إن توابع BIOS الخاصة بالإظهار ليست الطريقة الوحيدة للرسم على الشاشة، حيث يمكن استخدام توابع الاظهار الخاصة بـ DOS، أو توابع BIOS للوصول المباشر الى الذاكرة. اذا أردنا تصنيف هذه الطرق تبعا لفعاليتها فإننا نجد أن توابع BIOS تقع بين توابع DOS وتوابع الوصول المباشر للكيان الصلب.

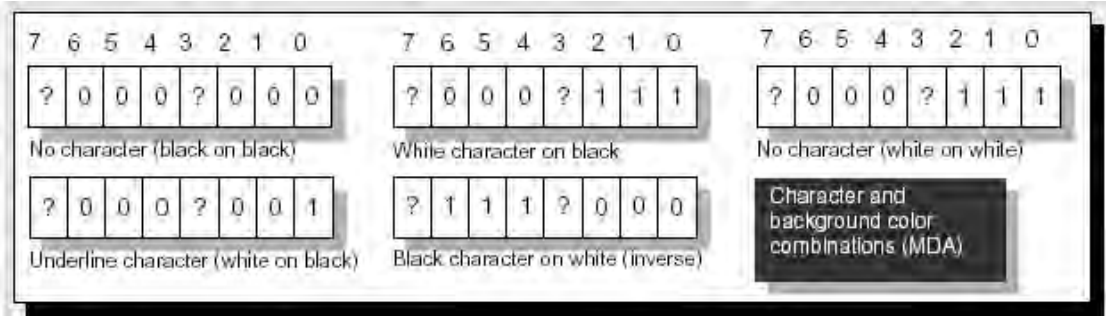
تتميز توابع DOS بالاستقلالية الكاملة عن الاجهزة، حيث يمكن توجيه خرج تابع الاظهار الى الشاشة أو الطابعة أو الى ملف على القرص الصلب. لكن تنفيذ هذه التوابع يعتبر بطيء نسبيا، كما أنها لا تتمتع بالمرونة الكافية.تقدم توابع الوصول المباشر الى الكيان الصلب سرعة أكبر ومرونة أكبر، الا انها تعتمد على نوعية الجهاز بشكل كبير. فمثلا اجرائية لعرض محرف في CGA سوف لن تعمل عند استخدامها على بطاقات MDA. ان توابع BIOS الاظهارية ليست سريعة كما توابع الوصول المباشر الى الذاكرة، لكنها متوافقة مع مختلف انماط بطاقات الشاشة.

11 6 1- اختيار لون الخط و لون الخلفية

تتطلب بعض توابع اظهار المحارف من المبرمج تمرير لون المحرف و لون الخلفية. كل محرف يملك بايت خاص باللون، يقسم هذا البايت الى قسمين يتألف كل منهما من أربعة خانات. تستخدم الخانات الأقل أهمية تحدد لون الحرف، بينما تستخدم الخانات الأكثر أهمية لتحديد لون الخلفية.



الشكل107



الشكل108

في بطاقات العرض اللونية (CGA) يتم اختيار اللون من لوحة ألوان مكونة من 16 لونا مختلفا. الشكل التالي يظهر هذه الألوان مع أرقامها:

جدول21

Color/Graphics Adapter color palette							
Decimal	Hex	Bin	Color	Decimal	Hex	Bin	Color
0	00H	0000(b)	Black	8	08H	1000(b)	Dark gray
1	01H	0001(b)	Blue	9	09H	1001(b)	Light blue
2	02H	0010(b)	Green	10	0AH	1010(b)	Light green
3	03H	0011(b)	Cyan	11	0BH	1011(b)	Light cyan
4	04H	0100(b)	Red	12	0CH	1100(b)	Light red
5	05H	0101(b)	Purple	13	0DH	1101(b)	Light purple
6	06H	0110(b)	Brown	14	0EH	1110(b)	Yellow
7	07H	0111(b)	Light gray	15	0FH	1111(b)	White

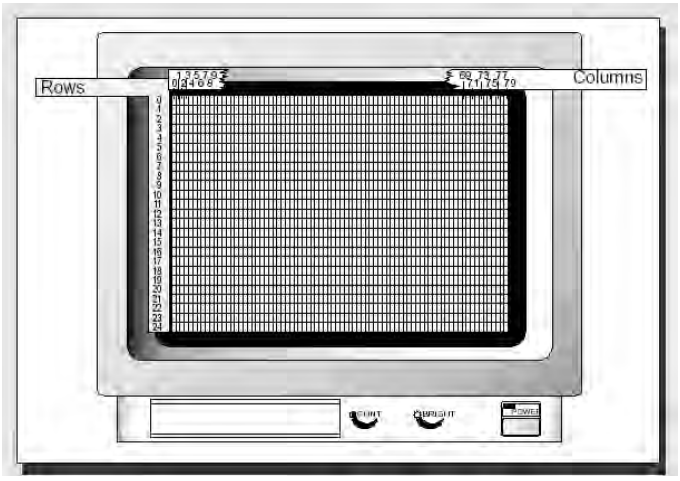
11 6 2 -مجموعة محارف ASCII

يستخدم الحاسب مجموعة محارف مكونة من 256 محرف، تحوي رموز و أحرف و أرقام و محارف خاصة. معظم هذه المحارف هي أحرف لغات أجنبية أو رموز رياضية.

11 6 3 -نظام احداثيات الشاشة

معظم توابع الاظهار معينة تتطلب تمرير احداثيات كبارامترات للعرض، حيث يتم من خلالها تحديد مكان اظهار المحرف. لذلك يجب فهم هذه الاحداثيات قبل استدعاء تلك التوابع.

ان مركز الاحداثيات- سواء في نمط الاظهار النصي أو الرسومي- يكون في الزاوية اليسرى-العليا للشاشة. تزداد قيمة X كلما اتجهنا يمينا، بينما تزداد قيمة Y كلما اتجهنا للأسفل.



الشكل109

11 ـ 7 ـ توابع BIOS الاظهارية

The video BIOS services

سنعرض هنا أهم توابع التحكم و الخدمات المستخدمة في النمط النصي.

جدول22

Video BIOS functions and support from EGA, VGA and standard BIOS

No.	Meaning	BIOS*	No.	Meaning	BIOS*
00H	Determine video mode	SEV	0EH	Terminal character output	SEV
01H	Define cursor size	SEV	0FH	Determine video mode	SEV
02H	Set cursor position	SEV	10H	EGA/VGA color options	EV
03H	Read cursor position	SEV	11H	Character generator access	EV
04H	Read light pen	SEV	12H	Set/read video configuration	EV
05H	Define current screen page	SEV	13H	Write string (AT only)	SEV
06H	Scroll screen up	SEV	14H	Reserved	—
07H	Scroll screen down	SEV	15H	Reserved	—
08H	Read character and attribute	SEV	16H	Reserved	—
09H	Write character and attribute	SEV	17H	Reserved	—
0AH	Write character to cursor position	SEV	18H	Reserved	—
0BH	Set color palette for graphics mode	SEV	19H	Reserved	—
0CH	Set screen pixel in graphics mode	SEV	1BH	Toggle between video cards	V
0DH	Read screen pixel in graphics mode	SEV	1CH	Save/restore video card status	V

* S = Standard BIOS E = EGA BIOS V = VGA BIOS

الخطوة الأولى لاستدعاء هذه الاجرائيات هي تحميل المسجل AH برقم الاجرائية، وعند وجود إجراءات فرعية يجب تحميل رقم الاجراء الفرعي في المسجل AL.

11 ـ 7 ـ 1 ـ المقاطعة Int 10H من أجل النمط النصي

تسهل المقاطعة 10H معالجة عمليات الشاشة كاملا. وتستخدم في كثير من المهام التي سنشرحها الآن. ويتطلب تنفيذ وظائف هذه المقاطعة وضع رقم الخدمة في المسجل AH الذي يحدد طبيعة الوظيفة التي تقوم بها المقاطعة , وتحتفظ المقاطعة بقيم المسجلات BP, SI, DI , DX, CX, BX لاستعمالها الخاص تبعاً لنوع الخدمة المطلوبة , علماً أن المقاطعة تحافظ على كافة قيم المسجلات السابقة باستثناء المسجل AX.

11 ـ 7 ـ 1 ـ 1 ـ الخدمة 00 من المقاطعة H10 تحديد نمط الإظهار

SETTING THE MODE

يزود نظام الحاسب مقاطعة خاصة لتحديد نمط العرض النصي أو النمط الرسوم البيانية , ويستطيع روتين هذه المقاطعة وضع نمط الإظهار للبرنامج الذي يجري تنفيذه حالياً ويستطيع الانتقال بين النمط النصي ونمط الرسوم البيانية عند استخدام moniter ملون , علما أن تحديد نمط الإظهار يؤدي بدوره إلى عملية مسح الشاشة. يبين الشكل التالي أنماط الإظهار

المختلفة التي وضعت أرقامها في العامود اليميني من الجدول , فالنمط 03 على سبيل المثال يقابل نمطا نصيا ملونا.

```
MOV  AH,00                ;Requst set mode
MOV  AL,03                ;80 X 25  Standard color  text
INT 10H
```

جدول23

Video mode sub-functions from video BIOS function 00H		
Code	Mode	Card
00H	40x25 character text, 16 colors, no color display	CEV
01H	40x25 character text, 16 colors	CEV
02H	80x25 character text, 16 colors, no color display	CEV
03H	80x25 character text, 16 colors	CEV
04H	320x200 pixel graphics, 4 colors	CEV
05H	320x200 pixel graphics, 4 colors, no color display	CEV
06H	640x200 pixel graphics, 2 colors	CEV
07H	80x25 character text, mono	MHE"
08H	Reserved	
0CH	Reserved	
0DH	320x200 pixel graphics, 16 colors	EV
0EH	640x200 pixel graphics, 16 colors	EV
0FH	640x350 pixel graphics, mono	E*
10H	640x350 pixel graphics, 16 colors	EV
11H	640x480 pixel graphics, 2 colors	V
12H	640x480 pixel graphics, 16 colors	V
13H	320x200 pixel graphics, 256 colors	V
EGA card on MDA monitor M = MDA H = Hercules C = CGA E = EGA V = VGA		

INT 10H- 2 1 7 11 الخدمة 01 تحديد حجم المشيرة

SET CURSOR SIZE

لا تشكل مشيرة الشاشة محرفاً من مجموعة محارف ASCII , كما أنها تتواجد فقط في النمط النصي (TEXT) , وتأخذ هذه مشيرة حجماً يعتمد على طريقتنا في التحكم بالكيان الصلب (Hardware) حيث يظهر شكل المشيرة نتيجة لعرض خطوط المسح (scan lines)التي يعتمد عددها على نوع مكيف الإظهار (adapter).ويأتي دور الخدمة 01 من المقاطعة 10 H بالتحكم بحجم المشيرة من خلال تحديد خط بداية المسح (start scan line) العلوي في المسجل CH (في الخانات من 0 إلى 4) و تحديد خط نهاية المسح (end scan line) السفلي في المسجل CL (في الخانات من 0 إلى 4 أيضا) , ولذلك يصبح حجم المشيرة محدوداً بين الخط العلوي والسفلي. ترقم خطوط المسح من 0 إلى 13 (0:13) في الشاشات أحادية اللون (monochrome)وشاشات الرسوم البيانية المطورة (enhanced graphics) وترقم من 0 إلى 7 (0:7) عند استخدام مكيف إظهار رسوم بيانية ملون.

يبين الشكل التالي كيفية توسيع حجم المشيرة من الأعلى والأسفل في شاشة أحادية اللون أو EGA:

```
MOV   AH,01              ;Requst set cursor size
MOV         CH,00              ; Start scan Line
MOV         CL,13                    ; End scan Line
INT         10H
```

INT 10H- 3 1 7 11 الخدمة 02 تحديد موقع المشيرة

SET CURSOR POSITION

تفيد هذه الخدمة في تحديد موقع المشيرة في أي مكان من الشاشة من خلال رقم سطر وعامود الموقع المطلوب. وعادة ما يكون موقع المشيرة في الصفحة 0 , لكننا نستطيع تحديد أي صفحة من 0 على 3 من أجل الشاشات التي يتألف نمطها النصي من 80 عامود حيث سيكون موقع المشيرة في كل صفحة مستقل عن الصفحات الأخرى

```
MOV   AH,02              ;Requst set cursor
MOV         BH,00              ;Page Number 0
MOV         DH,Row                   ; Row
MOV         DL,COL                   ; Column
INT         10H
```

INT 10H- 4 1 7 11 الخدمة03 قراءة موقع المشيرة وحجمها

READ CURSOR POSITION

يستطيع البرنامج استخدام هذه الخدمة لمعرفة رقم السطر والعامود لموقع المشيرة الحالي بالإضافة على معرفة حجم المشيرة , ويفيد ذلك عمليا عندما يستخدم البرنامج الشاشة بشكل مؤقت ثم عليه إعادة تهيئتها كما كانت.

```
MOV   AH,03              ;Requst  cursor Location
MOV   BH,00              ;Set Page Number 0 (normal)
INT         10H
```

وتعيد المقاطعة السابقة عند انتهائها خط بداية المسح للمشيرة في المسجل CH وخط نهاية المسح في المسجل CL , وتحمل أيضا المسجل DX برقم السطر والمسجل DL برقم العمود.

INT 10H- 5 1 7 11 الخدمة05 اختيار الصفحة الفعلية

SELECT ACTIVE PAGE

تسمح هذه الخدمة في الأنماط النصية ذات الأرقام من 0 إلى 3 أو 13 إلى 16 بتحديد الصفحة التي سيتم إظهارها على الشاشة.وتأخذ الصفحات في الشاشة المؤلفة من 80 عامودا الأرقام من 0 على 3 , في حين تأخذ في الشاشات المؤلفة من 40 عامودا الأرقام من 0 إلى 7

```
MOV   AH,05                ;Requst  active Page
MOV   ○○AL,Page #                  ; Page Number
INT          10H
```

جدول24

Number of available screen pages depends on video card and video mode			
Mode	Resolution	Card	Pages
7	80x25	MDA/Hercules	1
0/1	40x25	CGA	8
2/3	80x25	CGA	4
0/1	40x25	EGA/VGA	16
2/3	80x25	EGA/VGA	8

INT 10H- 6 1 7 11 الخدمة06 إزاحة معطيات الشاشة نحو الأعلى

SCROLL UP SCREEN

تستخدم هذه الخدمة لمسح الشاشة , حيث وضع القيمة 00 في المسجل AL يؤدي إلى مسح كامل الشاشة في النمط النصي من خلال إزاحة كافة المعطيات التي تظهر في الشاشة نحو الأعلى , ويتم ذلك عمليا من خلال وضع محارف الفراغ (blank) في كافة مواقع الشاشة. ففي كل مرة تزيح فيها المقاطعة معطيات الشاشة سطرا واحد نحو الأعلى تنتقل معطيات كل سطر نحو السطر الذي فوقه ويحل مكان السطر السفلي سطر يحتوي على محارف الفراغ.

أما إذا أردنا إزاحة معطيات الشاشة نحو الأعلى عددا من المرات , نضع ذلك العدد ضمن المسجل وعلى هذا فإنه لإزاحة معطيات الشاشة نحو الأعلى عددا من الأسطر ونضع قيمة بايت المواصفات في المسجل BH وإحداثيات موقع بداية الإزاحة (سطر: عامود) في المسجل CX وإحداثيات موقع نهاية الإزاحة (سطر: عامود) في المسجل DX (أي زوج المسجلات DX, CX يعرق نافذة تجري خلالها عملية الإزاحة). تحل هذه الخدمة مشكلة ظهور المعطيات فوق بعضها البعض (عند بلوغ السطر الأخير من الشاشة)

Exam1

```
Scroll For one Line
MOV   AX,0601H                ; Scroll UP One Line
MOV   BH,30H                  ; Black on cyan
MOV   CX,0000                 ; from 00,00
MOV   DX,184FH                ; to 24,79 (Full Screen)
INT          10H
```

Exam2

```
Scroll Five Lines
MOV   AX,0605H                ; Scroll Five Lines
MOV   BH,61H                  ; Blue on brown
MOV   CX,0A1CH          : from row 10, column 28
MOV   DX,184FH                ; to row 14, column52
INT   10H
```

11 ـ 7 ـ 1 ـ 7 INT 10H- الخدمة07 إزاحة معطيات الشاشة نحو الأسفل

SCROLL UP SCREEN

تزيح هذه الخدمة معطيات الشاشة نحو الأسفل في النمط النصي وسيختفي عندها السطر السفلي وسيظهر سطر كامل من محارف الفراغ في أعلى الشاشة في كل مرة نزيح فيها معطيات الشاشة نحو الأسفل , وتعمل هذه الخدمة كخدمة 06 السابقة

11 ـ 7 ـ 1 ـ 8 INT 10H- الخدمة08 قراءة المحرف الموجود عند المشيرة مع مواصفاته

Read Attraibute/Character At Cursor Position

تقرأ هذه الخدمة من ذاكرة الشاشة المحرف الموجود عند موقع المشيرة مع ما يتمتع من مواصفات وذلك في كلا النمطين النصي والرسوم البيانية , ويوضع عندئذ رقم الصفحة الاعتيادية 0 في المسجل BH , وتأخذ الصفحات في الشاشة المؤلفة من 80 عامودا الأرقام من 0 على 3 , في حين تأخذ في الشاشات المؤلفة من 40 عامودا الأرقام من 0 إلى 7

```
MOV   AH,08          ;Request  read attribute/character
MOV   BH,00          ;Page Number 0 (normal)
INT           10H
```

تعيد هذه الخدمة المحرف في المسجل AL ومواصفاته في المسجل AH وتعيد القيمة 00 في المسجل AL في نمط الرسوم البيانية إذا كان المحرف لا ينتمي إلى مجموعة محارف ASCII. تقرأ هذه الخدمة محرف واحد فقط لذلك نحن بحاجة على كتابة حلقة لقراءة مجموعة محارف متعاقبة.

11 ـ 7 ـ 1 ـ 9 INT 10H- الخدمة09 إظهار المحرف مع مواصفاته عند موقع المشيرة

Display Attraibute/Character At Cursor Position

تفيد هذه الخدمة في إظهار محرف في كلا النمطين (نصي ورسوم بياني) وفق المواصفات(attribute) التي نحددها كالوميض(blinking) أو العرض المرئي المعكوس (reverse video) و لتحقيق ذلك يتم وضع شفرة محرف ASCII في المسجل AL , ومواصفات هذا المحرف في المسجل BL , ورقم الصفحة في المسجل BH , وعدد مرات إظهار المحرف في المسجل CX

```
MOV   AH,09          ; Request Display
MOV   AL, Character; Character To display
MOV   BH, page #    : Pasge Number
MOV   BL, attribute  ;Attribute (text) or color (graphics)
MOV   CX, repetition         ;Number Of repeated character
INT           10H
```

يتطلب إظهار المحارف المختلفة تنفيذ حلقة LOOP حيث نمرر في كل دورة محرفا جديد إلى المسجل AL. تبين التعليمات التالية مثالا لإظهار محرف القلب (heart) (03) خمس مرات بعرض مرئي معكوس ومع صفة الوميض

```
MOV   AH,09          ; Request Display
MOV   AL, 03         ; Heart
MOV   BH, 00         : Page Number 0 (Normal)
MOV   BL, 0F0H               ; Blink reverse video
MOV   CX, 05         ; Five Times
INT   10H
```

نستطيع باستخدام هذه الخدمة تغيير محتويات أية صفحة باستثناء الصفحة الحالية ومن ثم نستخدم الخدمة 05 لإظهارها (تحضير الصفحة في الخفاء ثم إبرازها دفعة واحدة)

INT 10H- 10 1 7 11 الخدمة 0A إظهار المحرف عند موقع المشيرة

Display Character At Cursor Position

تؤدي هذه الخدمة عملية إظهار محرف في النمط النصي أو نمط الرسوم البيانية , والفرق الوحيد بين هذه الخدمة(0AH) والخدمة (09H) في النمط النصي أن خدمتنا هذه تستخدم المواصفات(Attribute) الحالية للمحرف, فلا يوجد حاجة لتحميل المواصفات للمحرف.

```
MOV   AH,09                  ; Request Display
MOV   AL, Character          ; Character To display
MOV   BH, page #             : Pasge Number
MOV   CX, repetition                 ;Number Of repeated character
INT   10H
```

INT 10H- 11 1 7 11 الخدمة 13 إظهار سلسلة من المحارف

Display Character String

يمكن باستخدام الإجرائية 13H طباعة سلسلة من المحارف على الشاشة. يتم ذلك بوضع رقم الإجرائية (13H) في المسجل AH، و رقم الصفحة المراد عرضها في المسجل BH، و مكان طباعة بداية السلسلة في المسجلين DH, DL. كما يجب وضع عدد المحارف المراد طباعتها في المسجل CX.يتم التأشير إلى بداية السلسلة المراد عرضها باستخدام المسجل ES:BP.

```
MOV   AH,13H                 ; Request Display
MOV   AL, function           ; 0, 1, 2 or 3
MOV   BH, page #             : Page Number
MOV   BL, attribute          ;Screen Attribute
LEA   BP,Address             ;Address of String in   ES:BP
MOV   CX, Length             ;Length of string
MOV   DX, Screen             ;Relative starting location on screen
INT   10H
```

11 8 -ذاكرة RAM الخاصة بالعرض

11 9 -Video RAM

قبل أن يتمكن المبرمج من التعامل المباشر مع بطاقة الشاشة، يجب عليه أن يكون على دراية كافية بمكان و آلية تنظيم ذاكرة العرض.تختلف طريقة التنظيم في النمط الرسومي بين أنواع بطاقات الشاشة، بينما تتبع جميعها التنظيم ذاته في النمط النصي.

11 9 1 -توضع ذاكرة العرض ضمن ذاكرة الحاسب:

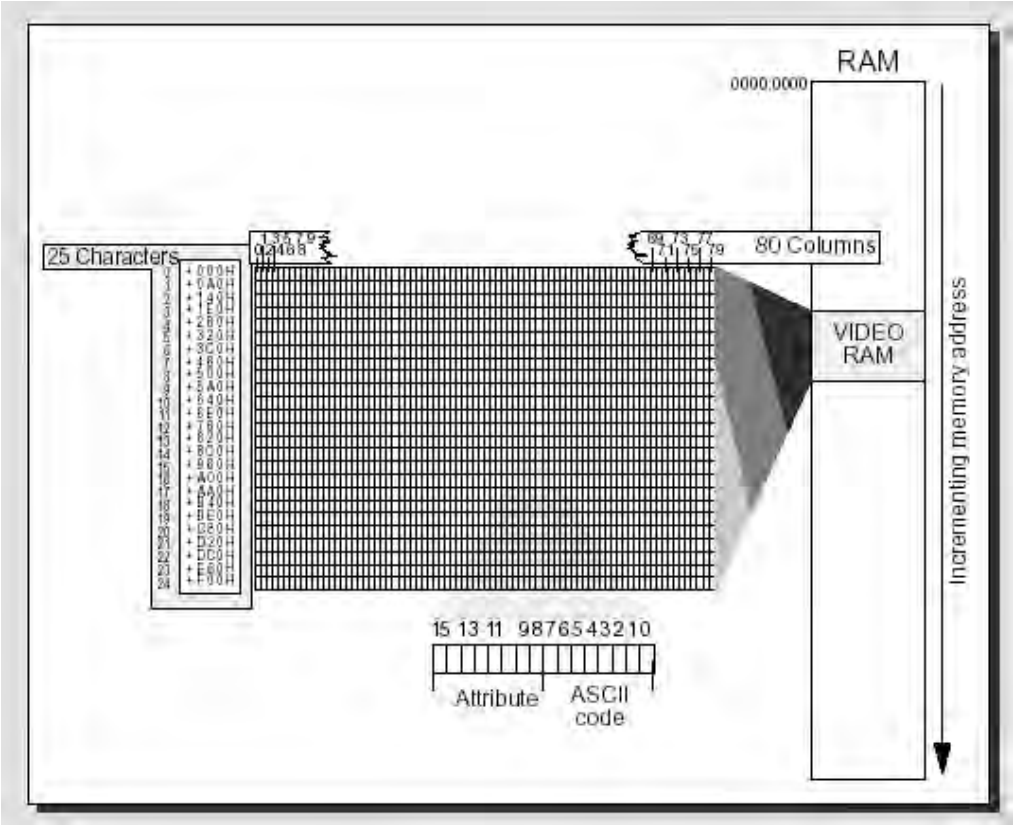
تبدأ الذاكرة الفيديوية من العناوين A000H و B000H.تستخدم بطاقات الشاشة الأحادية اللون MDA ذاكرة العرض ضمن المجال B000:0000 وحتى B000:7FFF، بينما تتعامل البطاقات اللونية معها بدءا من العنوان B800:0000.لايكون موضع ذاكرة العرض واضحا في بطاقات EGA و VGA.عادة لا تستعمل بطاقة الشاشة كامل الحيز الذاكري المحجوز لها. حيث تتعامل بطاقات MDA مع أول 4 كيلو بايت من ذاكرة العرض، بينما تستخدم بطاقات CGA أول 16 كيلو بايت منها، وتستخدم بطاقات هرقل 32 كيلو بايت من كامل ذاكرة العرض المتاحة.

11 9 2 -عنونة ذاكرة العرض

بما أن ذاكرة العرض هي جزء من ذاكرة الحاسب الأساسية، فإنه يتم الوصول اليها بنفس الطريقة تماما. يكون الوصول الى ذاكرة العرض غير ممكن أثناء تعامل اجرائيات مقاطعة BIOS معها. حيث تستخدم جميع بطاقات الشاشة تقنيات تمنع التداخل و الوصول المتعدد الى ذاكرة العرض.

11 9 3 -آلية تنظيم ذاكرة العرض في النمط النصي

ان معظم البرامج تعمل في النمط النصي، بعض المحارف يتم كتابتها بشكل مباشر على ذاكرة العرض، بغض النظر عن نوع بطاقة العرض المستخدمة. العديد من البرامج تعتمد على هذه التقنية، كبرنامج Lotus 1-2-3 و برنامج dBase IV.



الشكل 110

كما هو موضح في الشكل، يتم تمثيل كل موضع على الشاشة في ذاكرة العرض ب 2 بايت. يحوي الأول رقم المحرف في جدول ASCII. بينما يحتوي البايت الثاني على الخصائص اللونية للمحرف.

11 9 4 -تمثيل المحرف في ذاكرة العرض

حتى تتمكن من الوصول الى ذاكرة العرض، يجب معرفة طريقة تمثيل المحارف في الذاكرة. يتم ذلك بشكل مشابه لآلية عرض المحارف على الشاشة.ان المحرف الأول في الشاشة – المحرف الموجود في الزاوية اليسرى العليا- هو أيضا المحرف الأول في الذاكرة، حيث يتوضع عند الإزاحة 0000H. المحرف التالي الى اليمين يتوضع عند الازاحة بمقدار 0002H. تتبع جميع محارف السطر - 80 محرف – التنظيم ذاته.بما أن كل محرف يتطلب 2 بايت من الذاكرة، فإن كل سطر يمتد على 160 بايت من ذاكرة العرض. المحرف الأول من السطر الثاني يتبع المحرف الأخير من السطر الأول مباشرة.

11 9 5 -ايجاد موضع محرف في ذاكرة العرض

يمكن بسهولة ايجاد عنوان بداية سطر ما ضمن ذاكرة العرض، وذلك بضرب رقم السطر - بدءا من الصفر – بالعدد 160. ثم للانتقال الى عنوان بداية محرف ما في السطر يجب اضافة بعد هذا المحرف عن بداية السطر الى القيمة السابقة. و بما أن كل محرف يحتل 2 بايت من الذاكرة، فان ذلك يتم ببساطة بضرب رقم العمود بـ 2. يمكن تمثيل الحسابات السابقة بالصيغة التالية:

```
Offset_position(row, column) = row * 160 + column * 2
```

11 9 6 -الوصول الى عدة صفحات عرض

كما أشرنا سابقا، تدعم بطاقات الشاشة الحديثة (EGA, VGA) تقنية تعدد الصفحات، لأن ذاكرة العرض تقدم حيز أوسع لتخزين صفحة العرض.للوصول الى محتوى تلك الصفحات، يمكن استخدام الصيغة السابقة مع اضافة ترتيب الصفحة الى عنوان البداية و الى الازاحة.من أجل صفحات شاشة مؤلفة من 80 سطر و 25 عمود في النمط النصي، تكون العبارة الرياضية لحساب عنوان محرف ما في صفحة ما كالتالي:

```
Offset_position(column, row) = row * 160 + column * 2 + page * 4096
```

أما في حالة النمط المحمي Pmode فلا يجب أن نتستخدم مقاطعة البيوس BOIS المقاطعة العاشرة Int 10 فلذلك يتوجب علينا تغليف المقاطعة السابقة كلها وأن نؤمن وظائفها كاملة وذلك بالوصول المباشر إلى الذاكرة ولذلك يتوجب علينا فهم الذاكرة الفيديوية بشكل معمق وكذلك فهم مسجلات كرت الشاشة المختلفة وذلك لأننا سنستخدم HW مباشرة للوصول. و لا بد هنا من ملاحظة هامة:

تألف مجموعة محارف ASCII من 256 محرف، المحارف من 0 وحتى 127 تكون مخزنة على شريحة ROM، بينما المحارف من 128 وحتى 256 تؤخذ من جدول موجود في ذاكرة RAM.إن وجود هذا الجدول في RAM يتيح للمبرمج التعديل فيه بغرض إضافة رموز و محارف جديدة. لتمثيل كل محرف في الجدول نحتاج 8 بايت، أول 8 بايتات في الجدول تمثل المحرف 128, والثمانية بايتات التي تليها تمثل المحرف 129 في جدول ASCII، وهكذا.....حيث يمكن عرض محرف ما باستخدام التابعين 09H و 0AH كما سنرى ، ويتم استخدام الإجراء 02H لتحريك المشيرة إلى الموضع التالي.

11 10 -التعامل مع المسجلات و المستوى المنخفض (Hardware)

11 10 1 -متحكم السلسلة Sequencer controller

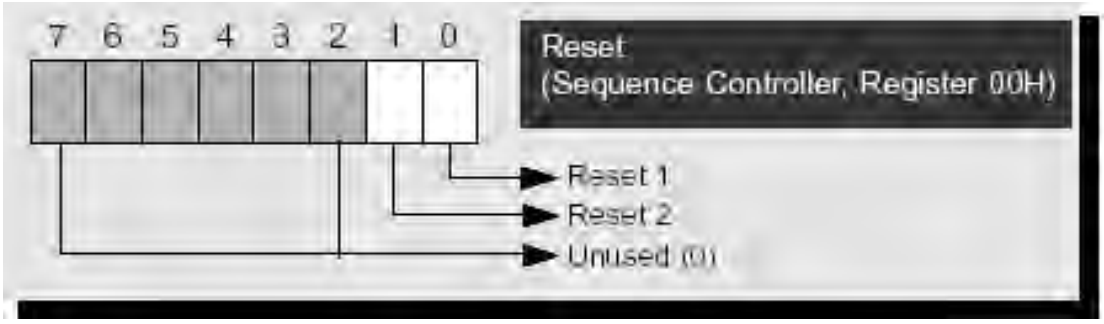
وهو يحتوي على خمسة مسجلات من 0 إلى 4 و يملك مسجل SEQUENCER بوابتين (عنوانين) للتعامل معه، العنوان الاول لتحديد أي مسجل فرعي تريد التعامل معه والعنوان الثاني من أجل Data والعنوانان هما 3d4 لل index و 3D5 لل data

```
outportb(VGA_SEQ_INDEX=3C4, i);
outportb(VGA_SEQ_DATA=3C5, *regs);
```

Number	Register Name
00H	Reset
01H	Clocking Mode
02H	Map Mask
03H	Character Map Select
04H	Memory Mode

الشكل111

Reset- register 00H



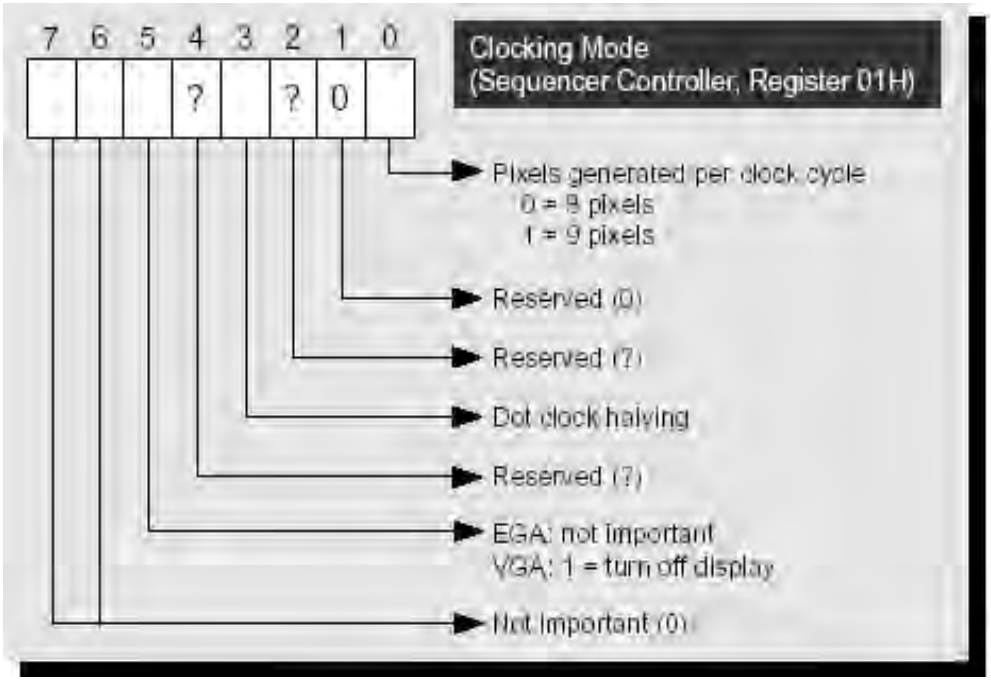
الشكل112

Clocking Mode

BIT 0 هو الذي يحدد عدد البكسلات التي يتم مسحها كل دورة ساعة عندما يكون 0 وبالتالي 8 بكسل وعندما يكون 1 يعني 9 بكسل.

Bit 3 هذا البت يكون 1 عند استخدام البيوس لل CGA بدقة 320* 200 وباقي الانوع يكون صفر

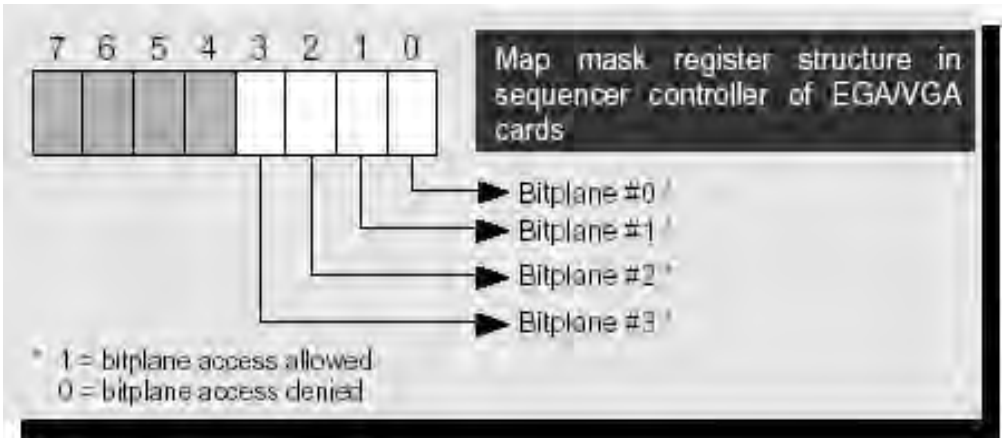
Bit 5 يكون 1 في بطاقة vga وذلك يعني اطفاء اشارات الفيديو وصول تام لل cpu لذاكرة الرام Video Ram



الشكل113

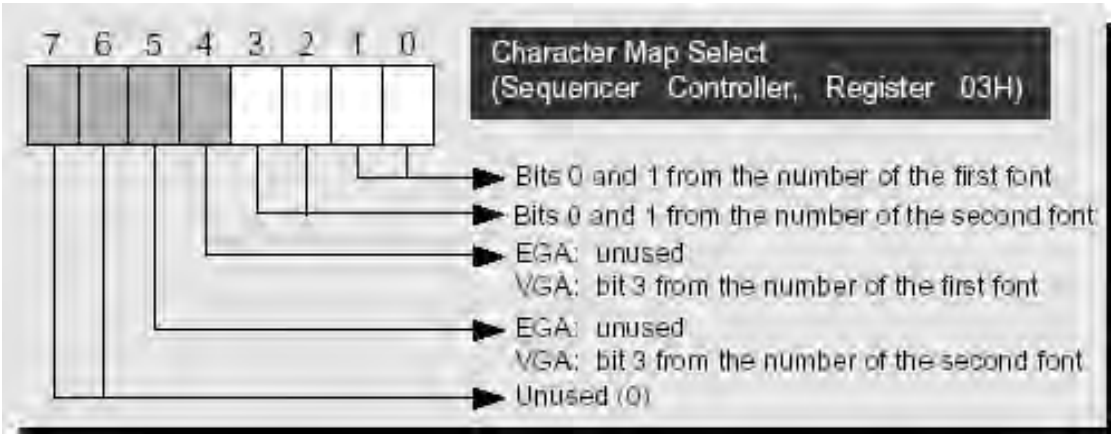
Map Mask 02H

يمكننا من الكتابة أو القراءة من Bitplaneالتابعة لذاكرة الفيديو ,وهذا مفيد عندما نريد أن نعالج فقط محتويات Bitplane واحدة لذاكرة الفيديو 4 Bitplanesويتم تمثيلها من خلال البتات الأربعة السفلية للمسجل.عندما يكون البت واحد نستطيع الوصول للذاكرة.

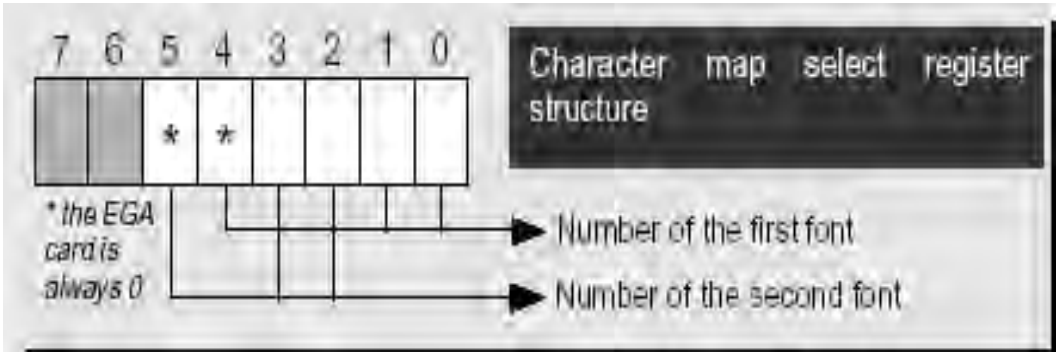


الشكل114

Character Map Select 03H



الشكل115

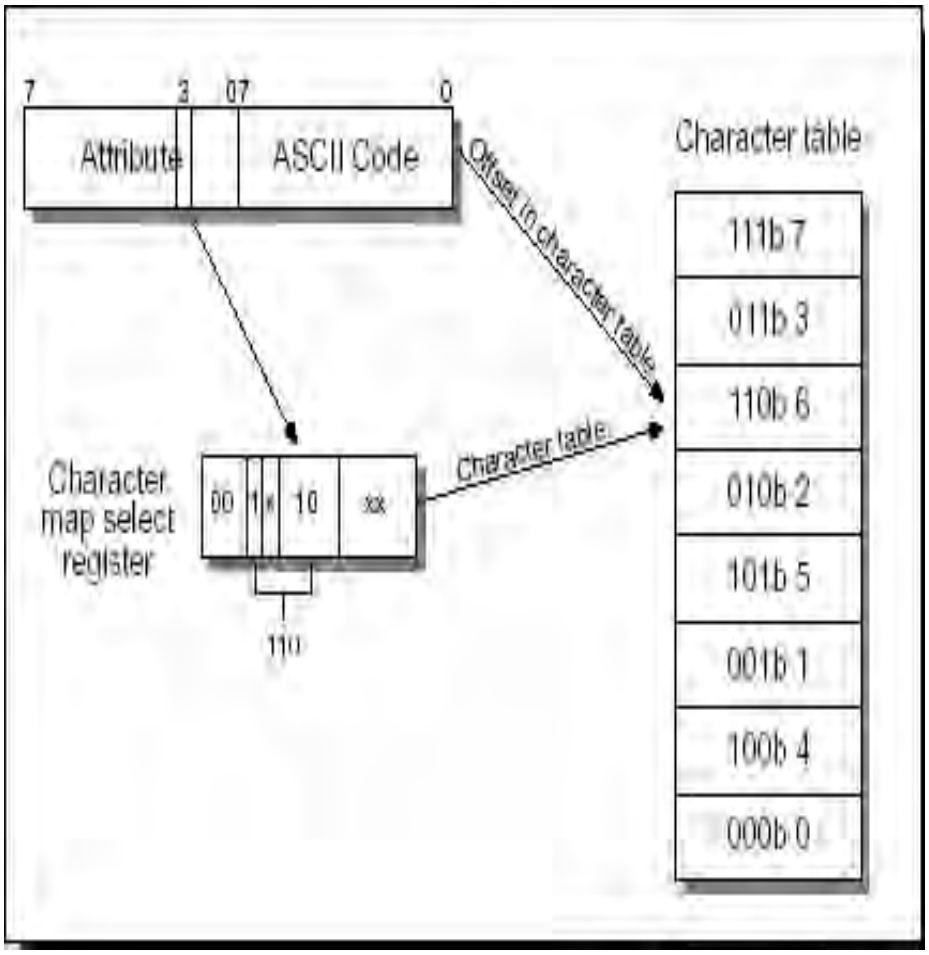


الشكل116

هذا المسجل يستخدم لإمكانية إظهار خطين بنفس الوقت ولكن كيف سنحدد أن الحرف الذي سوف يظهر ينتمي إلى أي خط يتم ذلك عن طريق بايت المواصفات (Attribute) إذا كان البت الثالث لهذا البايت 1 فهو ينتمي إلى الخط الاول وإذا كان صفر فهو ينتمي للخط الثاني....ولكن نحن نعلم أنه يمكن تحميل 8 Character Table في ال bitplane2 لذاكرة الفيديو وبالتالي كيف سيتم تحديد أي جدول يتم ذلك عن طريق المسجل Select map register.

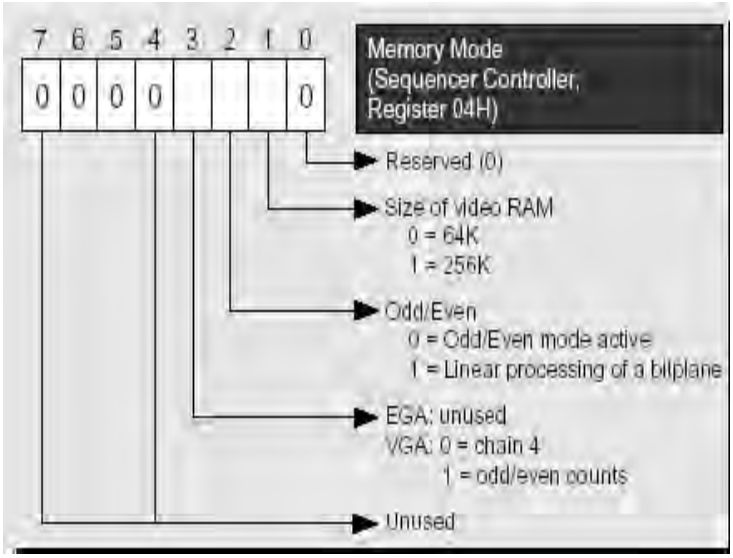
0 + 1 + 4: هذه البتات تستخدم لتحديد جدول الأحرف للحرف الذي نريد إظهاره ويتم ذلك عن طريق البت الثالث لبت المواصفات ,أي عندما يكون 0 بمعنى أخر عندما يكون عندنا حرف ما لكي نحدد من أي جدول سيأخذ شكله و يتم عن طريق البت الثالث لبايت المواصفات فإذا كان هذا البت صفر عندها يتحدد رقم الجدول من خلال هذه البتات الثلاثة

2 + 3+ 5 : نفس البتات السابقة ولكن عندما يكون البت الثالث لبايت المواصفات هو 1



الشكل 117

Memory Mode 04H



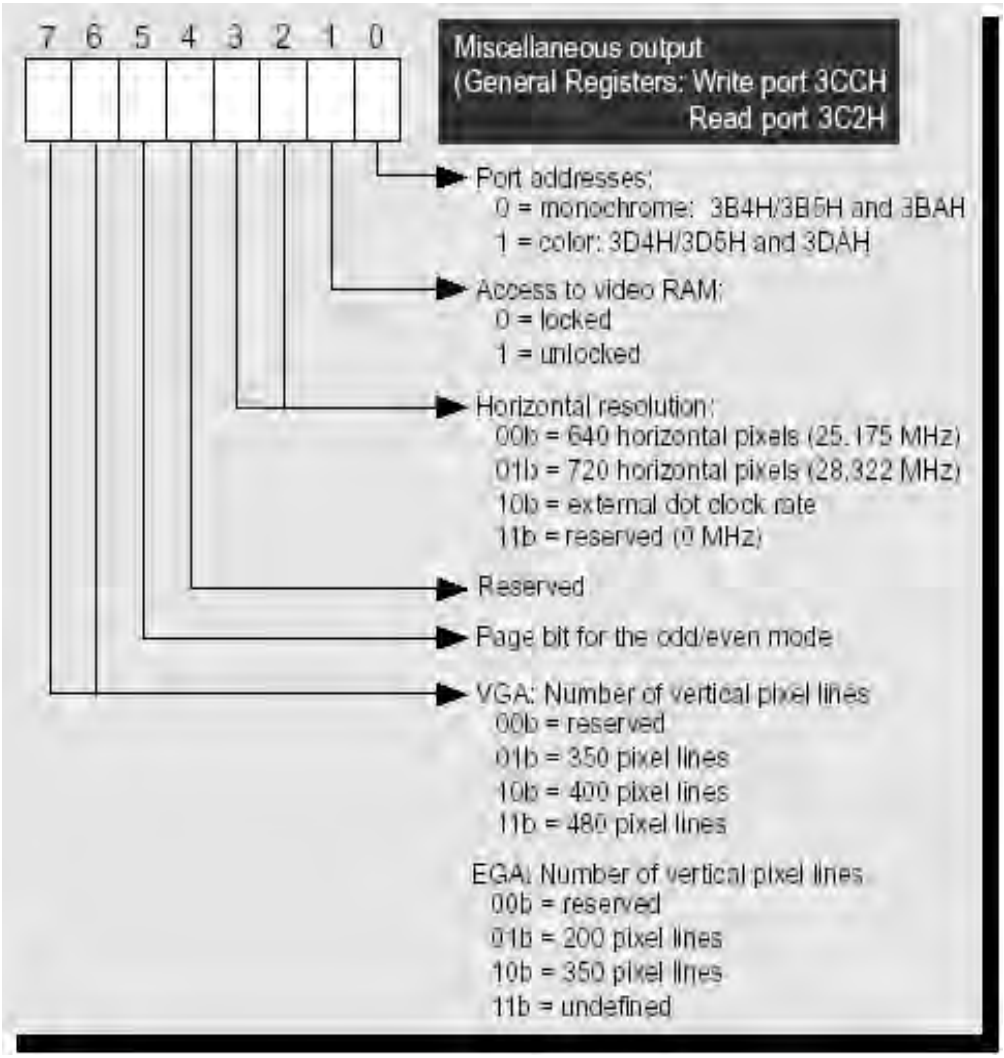
الشكل118

Bit 2: عندما يحتوي هذا البت القيمة0 هذا يعني أن عنونة الذاكرة ستكون بالطريقة الزوجية الفردية وعندما يكون 1 سيتم معالجة biplanes التابعة للذاكرة الفيديوية بطريقة خطية

ملا حظة:عادة ما تأخذ مسجلات SEQUENCER REGISTER القيم التالية:

المسجل	القيم الثنائية	القيمة الست عشرية
Reset- register 00H	00000011	0x03
Clocking Mode 01H	00000001	0x01
Map Mask 02H	00001000	0x08
Character Map Select 03H	00000000	0x00
Memory Mode 04H	00000110	0x06

المسجل المتنوع MISCELLANEOUS REGISTER:



الشكل119

وعادة يحتوي القيمة 0xE3 حيث:

B0= 1 ألوان متعددة 0 أحادي اللون
B1 =تمكين وحدة المعالج المركزية من الدخول إلى ذاكرة العرض
1عدم التمكين- 0 تمكين

CRTC REGISTERS- 2 10 11

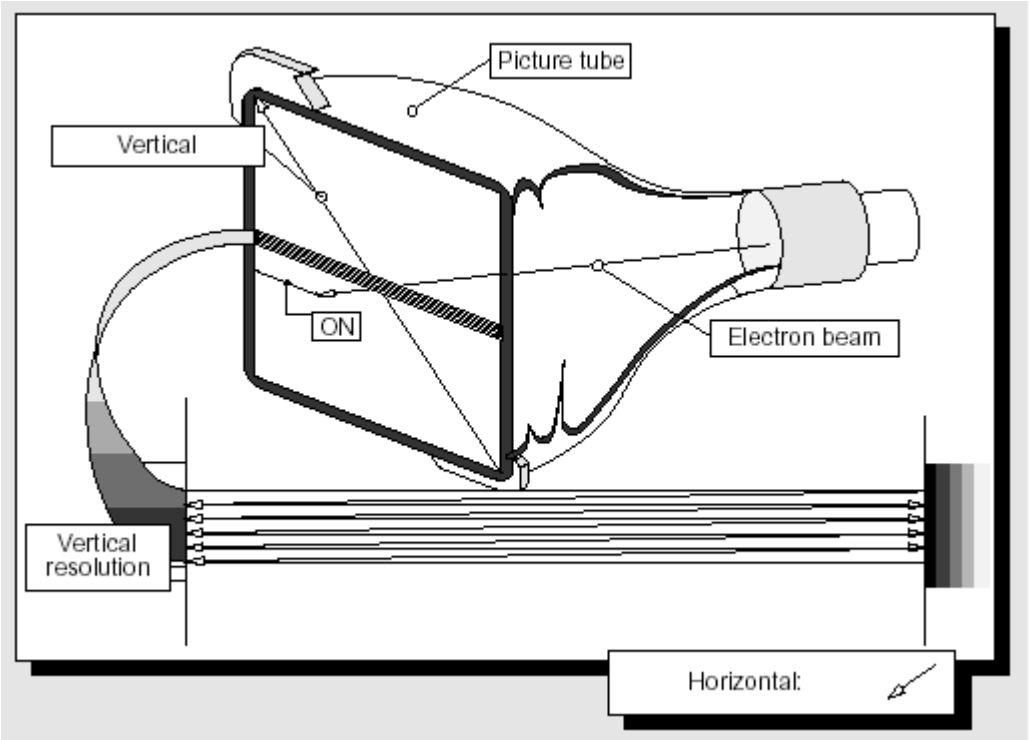
تكون مسجلات Crt Controller مسؤولة عن إظهار الصورة على الشاشة , حيث تتحكم بتولد إشارات فيديو للمرقاب (Monitor) باستخدام الحزمة الالكترونية المولدة من قبل أنبوب الصورة، حيث تملك عدد من المسجلات التي تتحكم بعملية التزامن بين المسح الأفقي والشاقولي والحزمة الالكترونية، وهذه المسجلات معقدة والكثير من المبرمجين لا يستمتعون بالعمل معها ولذلك يعتمدون على البيوس للتعامل مع هذه المسجلات ولكن عند العمل مع النمط

Pmode لا بد من التعامل المباشر مع الهارد وير دون المرور بالبيوس.بالاضافة إلى ذلك يوجد العديد من التأثيرات التي لا تستطيع البيوس عملها ولذلك فنحن مضطرون للعمل مع هذه المسجلات وتعلمها.يملك CRT Controller بوابتين (عنوانين) للتعامل معه العنوان الاول لتحديد أي مسجل فرعي تريد التعامل معه والعنوان الثاني من أجل Data والعنوانان هما 3D4 لل index و 3D5 لل data وهذا لل EGA & VGA

```
Write index   3D4h  - Read index 3D4h
Write data    3D5h -  Read data  3D5h
```

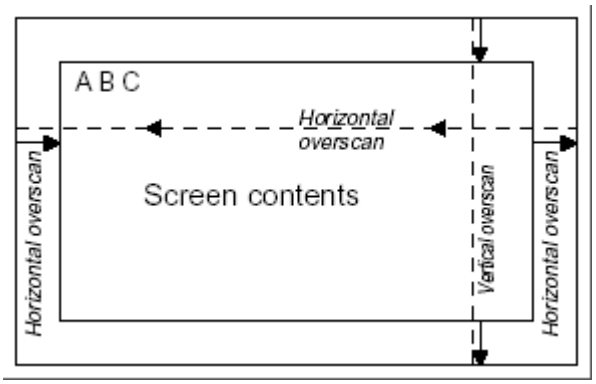
بينما لأحادي اللون 3B4 لل Index و 3B5 لل Data.

قبل الشروع بشرح المسجلات لابد من أرضية مشتركة نستطيع الإنطلاق منها.إن الحزمة الالكترونية تقوم بمسح الشاشة لإظهار الأحرف من الزاوية اليسارية العليا متجهة إلى اليمين ومن ثم تعود لتبدأ المسح من السطر الذي يليه وهكذا حتى يتم المسح الكلي للشاشة و أثناء المسح الأفقي للشاشة أو الشاقولي تستمر الحزمة الالكترونية بالعمل ليس فقط ضمن المراقب (Monitor) التي تظهر أمامنا بل خارجه ضمن مايسمى Over Scan وتقوم الحزمة الالكترونية بهذا العمل لإجراءات عديدة ليست موضوعنا هنا ولكن كيف سيتم تحديد كم ستسمر الحزمة بالمسح في منطقة Over Scan , يتم ذلك من خلال مسجلات Crt controller، و يوضح الشكل التالي تدفق الحزمة الالكترونية وكيفية عملها وطريقة مسحها الشاشة.



الشكل 120

كما نلاحظ عملية OverScan الأفقية والشاقولية في الشكل التالي.



الشكل 121

و تحتوي Crt Controller على 25 مسجل:

جدول25

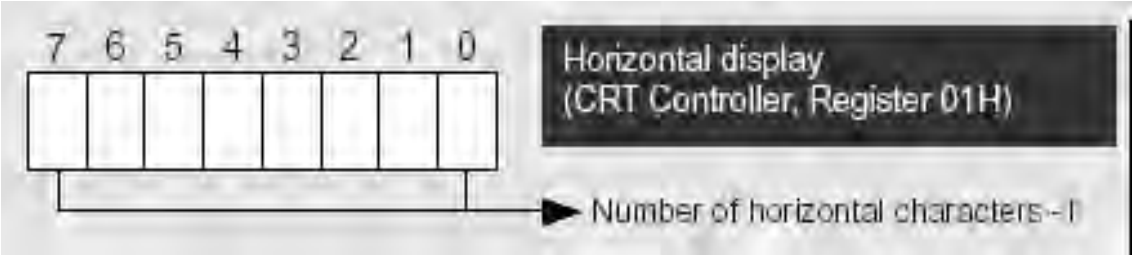
Number	Register Name	Number	Register Name
00H	Horizontal Total	0EH	Cursor Location High
01H	Horizontal Display End	0FH	Cursor Location Low
02H	Start Horizontal Blanking	10H	Start Vertical Rescan
03H	End Horizontal Blanking	11H	End Vertical Rescan
04H	Start Horizontal Rescan	10H	Light Pen Low (EGA only)
05H	End Horizontal Rescan	11H	Light Pen High (EGA only)
06H	Vertical Total	12H	Vertical Display End
07H	Overflow	13H	Offset
08H	Vertical Pel Panning	14H	Underline Location
09H	Maximum Scan Line	15H	Start Vertical Blank
0AH	Cursor Start	16H	End Vertical Blank
0BH	Cursor End	17H	Mode Control
0CH	Start Address High	18H	Line Compare
0DH	Start Address Low		

00H Horizontal Total

العرض الكلي للشاشة بمقياس الأسطر والأعمده وسيتم شرحه

Index 1 Horizontal Display:

العرض المرئي من الشاشة

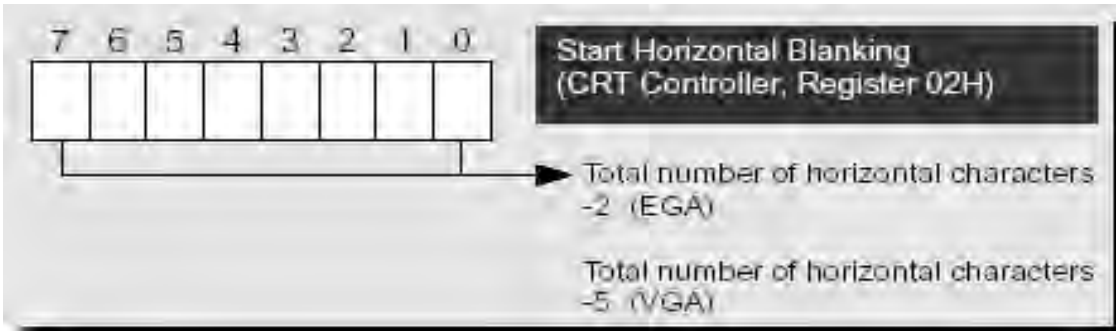


الشكل122

0 -7 هذا المسجل يحدد العدد الفعلي للأحرف التي ستظهر بشكل أفقي , حيث يتم إنقاص القيمة 1 منه في EGA&VGA

02H Start Horizontal Blanking

بداية المسح الأفقي وعادة يساوي العرض المرئي للشاشة



الشكل123

0 – 7: عند بداية الاشارة الأفقية لن يظهر ولا حرف على الشاشة لأن الحزمة الألكترونية المنطلقة من الأنبوب لم تبدأ بعد. وحالما تبدأ. ستكون البداية عند الرقم المهيء هنا، وقيم هذا المسجل VGA نا قص 5 و EGA ناقص 2.

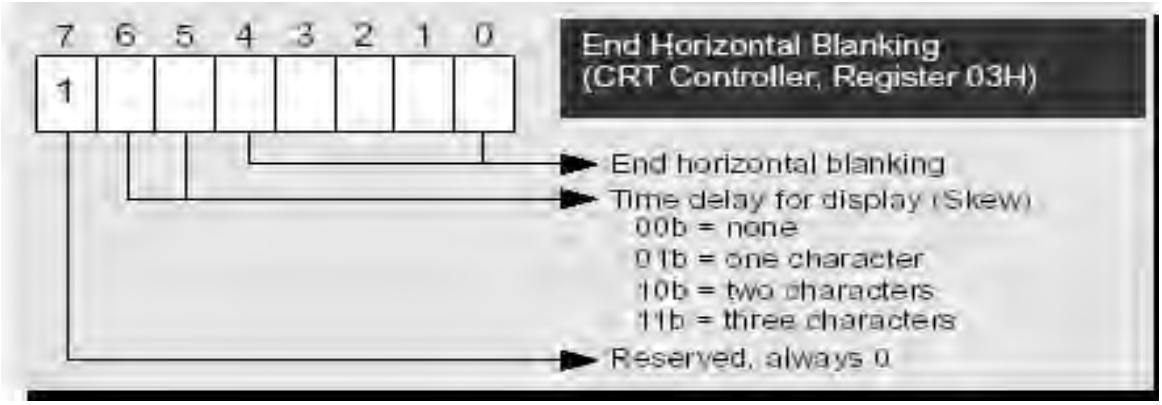
03H End Horizontal Blanking

نهاية المسح الأفقي للشاشة

b7= علامة إدخال مسجلات الصف العمودي.

b5 & b6= وهو وقت تأخير ضروري لكي نعطي ل Crt Controller وقت كاف لكي يقرأ الحرف ومواصفاته من الذاكرة الفيديوية وبعد ذلك يولد البكسلات المطلوبة بالتوافق مع مولد الأحرف character generator .

b0 - b4 = تحدد عدد ال Character (حيث وحدة القياس هي ال Character) التي يجب عدها حتى نهاية إعادة المسح الافقي حتى يبدأ تشغيله (بمعنى اخر عندما يصل المسح الافقي إلى نهاية السطر , كم عدد ال Character التي يجب أن يعدها حتى يعود لبدأ المسح).

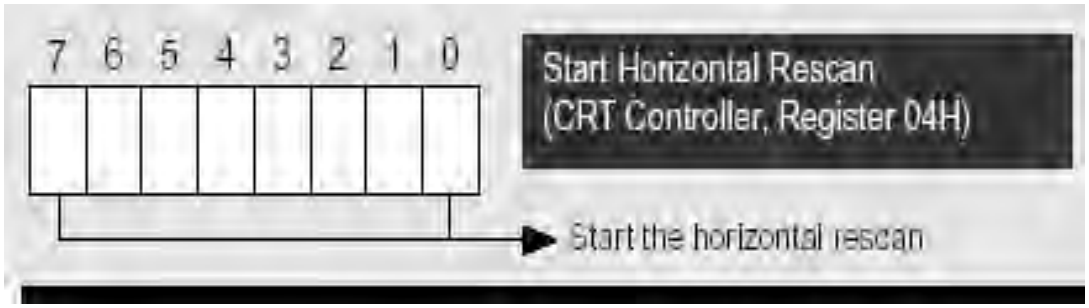


الشكل124

04H Start Horizontal Rescan

إعادة المسح الأفقي للشاشة بالجهة العكسية

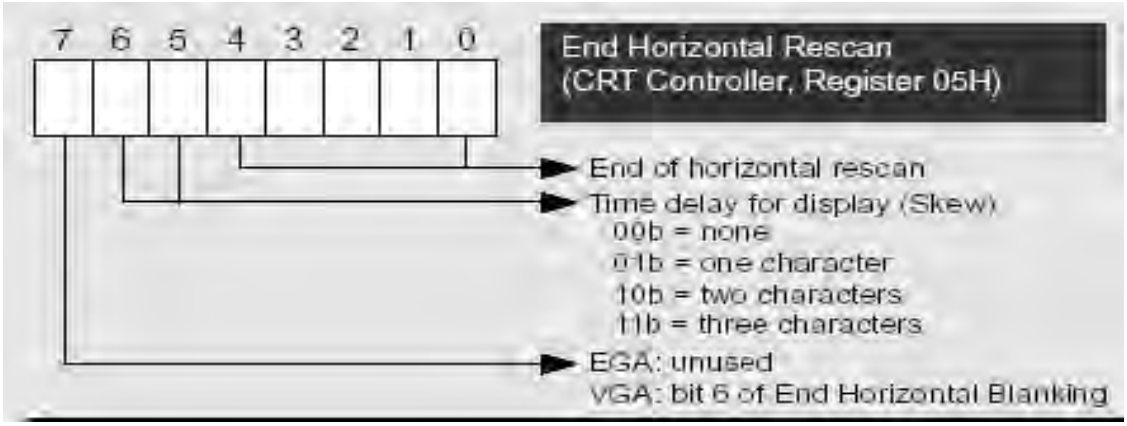
Index 4 = مجموع ما تم مسحة في المرة السابقة
=index1+(index0-Index1/2)+(80-Index 3)



الشكل 125

05H End Horizontal Rescan

نهاية المسح الأفقي للشاشة بالجهة العكسية.مشابه كثيرا للمسجل 03H ولكن بالجهة المعكوسة



الشكل126

06H Vertical Total

بداية العرض العمودي للشاشة:
العرض الكلي للشاشة - العرض المرئي من الشاشة تقسيم 2 = Index6

Index 12: نهاية العرض العمودي للشاشة

إرتفاع الشاشة = Index6 – Index12 .

07H Overflow

سجل الفيضان للمسجلات , في EGAو VGAنحتاج إلى مسجل الفيضان (Overflow)وذلك لأن عدد البكسلات الشاقولية في السطر تزيد عن 256ولذلك لا يمكن تمثيلها بمسجل وحيد ذو 8خانات ولذلك البت الزيادة يحفظ في مسجل الفيضان لمسجلات التي تحتاج 9 بتات.

08H Vertical pel panning

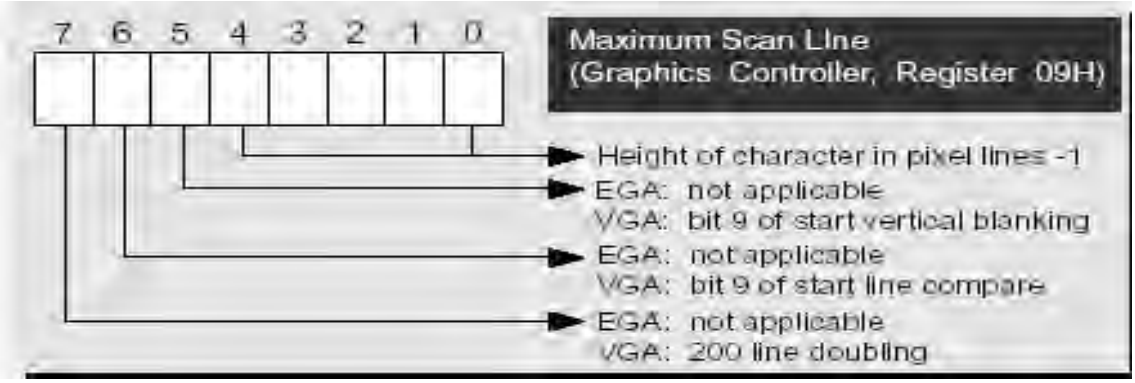
يستخدم لتحريك الصفحة:

B0 –b4 = عدد الأسطر التي تريد تحريكها بالطول

B5 –b7 = عدد الأسطر التي تريد تحريكها بالعرض

09H Maximum Scan Line

Index 9: مسجل التحدد + الفيضان و يستخدم في نمط الكتابة لتحديد إرتفاع الحرف و نمط الرسوما لمضاعفة العرض لكل خط و كمثال:0 نمط العرض عادي، 1 ضعف، 2 ضعفان ، 3 ثلاث أضعاف. ..الخ



الشكل127

0AH Cursor Start

Index A سجل بداية المؤشر

b0-b4= موقع بداية المؤشر
b5= ضبط المؤشر إذا كان موتهيأ

0BH Cursor End

Index B سجل نهاية المؤشر

b0-b4 = موقع نهايه المؤشر
b5-b6 = وقت تأخير إستجابة المؤشر

0CH Start Address High

في بداية تعرف النظام على الشاشة يبدأ Crt Controller يقرأ محتويات الشاشة من الذاكرة الفيديوية من عنوان إزاحة معين، يتم تحديد هذا الإزاحة من خلال هذا المسجل وفي حالة كانت الذاكرة في النمط الزوجي –الفردي يتم التقسيم على 2.يتم تخزين البايت العلوي في هذا المسجل والبايت السفلي في المسجل التالي.

0DH Start Address low

يتم تخزين بايت الازاحة السفلي المتعلق بالمسجل 0CH

0EH Cursor Location High

يحتوي البايت العلوي لموقع المؤشر في الصفحة الحالية

0FH Cursor Location High

يحتوي البايت السفلي لموقع المؤشر في الصفحة الحالية.

10H Start Vertical rescan

إعادة المسح العمودي للشاشة بالجهة العكسية.

11H End Vertical rescan

نهاية المسح العمودي للشاشة بالجهة العكسية

13H Offest

عدد البتات في خط المسح تقسيم النمط 8 أو 16

15H Start Vertical Blank

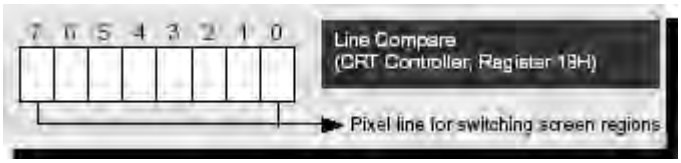
بداية المسح العمودي للشاشة

16H End Vertical Blank

نهاية المسح العمودي للشاشة

18H Line Compare

مسجل المقارنة للخط العمودي



الشكل128

هذا المسجل يمكن أن يستخدم لتقسيم الشاشة غلى منطقتين مختلفتين في الذاكرة الفيديوية، والقيمة المخزنة في هذا المسجل تمثل رقم السطر الفاصل بين المنطقتين (أي نهاية المنطقة الأولى وبداية المنطقة الثانية) و تجدر الملاحظة أنه عادة تأخذ مسجلات CRTC REGISTER الارقام التالية:

0x5F, 0x4F, 0x50, 0x82, 0x54, 0x80, 0x0B, 0x3E,0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,0xEA, 0x0C, 0xDF, 0x28, 0x00, 0xE7, 0x04, 0xE3,0xFF

3 10 11 -متحكم الرسوميات GRAPHICS CONTROLLER

وله 9 مسجلات

جدول26

Number	Register name	Number	Register name
00H	Set/Reset	05H	Graphics Mode
01H	Enable Set/Reset	06H	Miscellaneous
02H	Color Compare	07H	Color Don't Care
03H	Function Select	08H	Bit Mask
04H	Read Map Select		

00H Set/Reset

التهيئة وإعادة التهيئة للوحة الألوان

01H Enable Set/Reset

تمكين التهئية وإعادة التهيئة للوحة الألوان

02H Color Compare

مقارنة الألوان

03H Function Select

عدد البتات لإدارة البيانات قبل الكتابة لذاكرة العرض نمط الكتابة فقط.

05H Graphics Mode

b0-b1= كيفية إرسال البينات من المعالج إلى ذاكرة الفيديو

b3: كيفية قراءته لوحة الألوان

B4: عندما يكون 1 تمكين العنونة الزوجية والفردية

B5: في CGA تمكين اعنونة الزوجية والفردية

b6 : تمكين 256 لون مع تصغير حجم الشاشه

06H MISCELLANEOUS

تحديد عناوين الذاكرة

b0: نمط أحادي اللون =0 -نمط متعدد الألوان=1

b1:Enables Odd/Even mode if set

b2-b3=عنوان بداية ذاكرة العرض

```
:0  use A000h-BFFFh
:1  use A000h-AFFFh   VGA Graphics modes
:2  use B000h-B7FFh   Monochrome modes
:3  use B800h-BFFFh   CGA modes
```

07H Color Don't care

تجاهل مسح لوحات الألوان.

08H Bit Mask

قناع البت ويستخدم لتغيير الأنماط

ملاحظة:عادة تأخذ قيم المسجلات القيم التالية

0x00,0x00,0x00,0x00,0x03,0x00,0x05,0x0F,0xFF

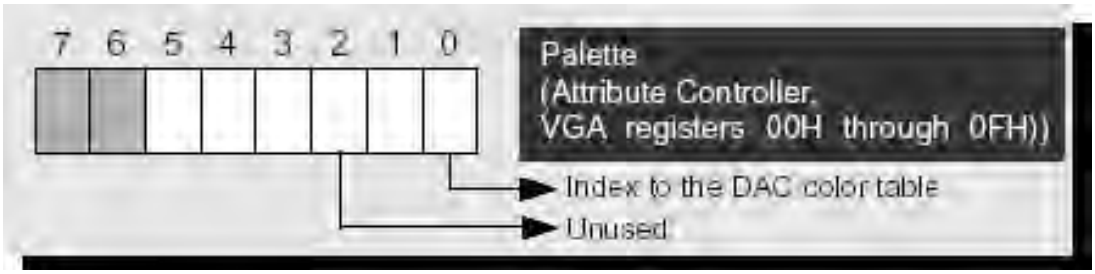
4 10 11 -متحكم الواصفات

ATTRIBUTE CONTROLLER

إن مسجلات Attribute controller تهيء ألوان الاشارات للشاشة , وتدير مسجلات الصفائح palette ومسجلات أخرى مطلوبة لتوليد إشارات الألوان وهي تتألف من 21 مسجل أول 16 منهم لتحديد الألوان هم مسجلات Palette.

Number	Register Name
00H-0FH	Palette Register
10H	Mode Control
11H	Overscan Color
12H	Color Plane Enable
13H	Horizontal Pel Penning
14H	Color Select (VGA only)

الشكل129



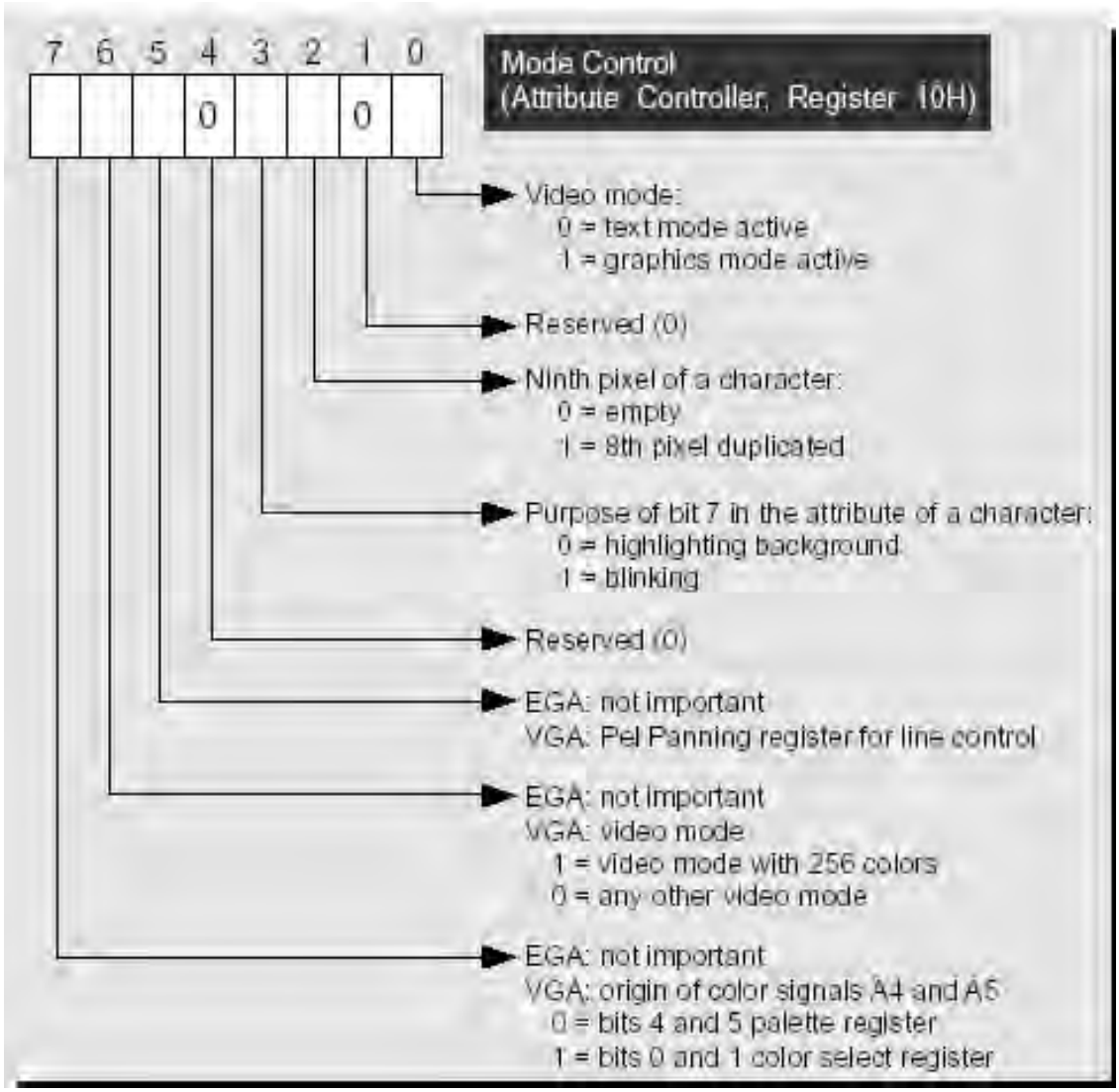
الشكل130

10H Mode Control

b0= تحديد نمط الرسومات.

b1, b2,b3= تحديد خصائص الحروف في نمط الكتابه.

b6= تحديد 8 بت لكل بكسل وهو ل256 لون.



الشكل131

11H Over Scan Color

لتحديد إطار على النافذه

12H Color Plane Enable

تحديد عدد الألوان في لوحه الألوان

13H Horizental pel penning

يحدد عدد البكسلات التي ستمثل الهامش

14H Color Select

تحديد ناخب للوحه الألوان

تجدر الملاحظة أنه عادة تأخذ المسجلات القيم التالية:

```
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
0x01, 0x00 ,  0x0F, 0x00 , 0x00
```

عناوين (بوابات)التعامل مع المسجلات:

جدول27

MISCELLANEOUS REGISTER	
Write index	لا يوجد
Read index	لايوجد
Write data	3C2h
Read data	3CCh
SEQUENCER REGISTER	
Write index	3C4h
Read index	3C4h
Write data	3C5h
Read data	3C5h
CRTC REGISTER	
Write index	3D4h
Read index	3D4h
Write data	3D5h
Read data	3D5h
GRAPHICS CONTROLLER	
Write index	3CEh
Read index	3CEh
Write data	3CFh

Read data	3CFh
ATTRIBUTE CONTROLLER	
Write index	3C0h
Read index	3C0h
Write data	3C1h
Read data	3C1h

11 11- المراجع :

http://www.linuxbios.org/faq/archive/0127.html
http://www.linuxbios.org/faq/archive /
http://www.i18nguy.com/unicode/codepages.html
http://www.bsdg.org/SWAG/EGAVGA/index.html
http://www.bsdg.org/SWAG /

برمجة الحواسيب الشخصة بلغة التجميع
الفصل التاسع مقدمة إلى معالجة الشاشة ولوحة المفاتيح
الفصل العاشر المعالجة المتقدمة للشاشة.

PC intern
الفصل الرابع Closer look at vedio cards

12 -سواقة القرص المرن و القرص الصلب
Diskettes and Hard Drives

مُقدِّمة

إن الهدف الأساسي من إجرائيات الـBIOS هو إنجاز وتنفيذ التوابع ذات المستوى المنخفض نيابة عن الـDOS .فعلى سبيل المثال, تستطيع إجرائيات الـBIOS تهيئة سطح القرص المرن أو الولوج إلى قطاعاتsectors السواقة الصلبة، ولكن يبقى الـDOS هو الموجه الرئيسي لهذه العمليات.تقوم غالبية التطبيقات بإنجاز عمليات القرص الصلب في مستوى الـDOS بدلا من مستوى الـBIOS، وعلى الرغم من ذلك يوجد بعض الاستثناءات. على سبيل المثال هناك بعض البرامج الخدمية للقرص الصلب مثل PC Tools أو Norton , تقوم بالولوج إلى القرص الصلب بواسطة إجرائيات الـBIOS، ولكن بشكل عام هذه البرامج التخصصية نادرة.

سنتناول في هذا الفصل كيفية الولوج إلى سواقات الأقراص بواسطة توابع الـBIOS، ونظرا لأنه لا يتم برمجة جميع متحكمات الأقراص بشكل مماثل, فإننا لا نستطيع برمجة موجه القرص بشكل مباشر.إن جميع التوابع التي يمكنك إنجازها على مستوى موجه القرص يمكنك أيضا أن تنجزها على مستوى الـBIOS, لذلك فإنه من الجدير بالاهتمام استخدام توابع الـBIOS لتجنب البرمجة التي تعتمد بشكل مباشر على موجه القرص.

سوف نناقش أيضا بالإضافة إلى توابع الـBIOS بعض الموضوعات الأخرى ذات الصلة بالسواقة الصلبة , و سنقوم بشرح الأنواع المختلفة لمتحكمات السواقة الصلبة و سنرى أيضا كيف تسجل السواقات الصلبة البيانات،وسوف ننهي الفصل بمناقشة عن كيفية تجزئة القرص الصلب و التي تتيح للمستخدم أن يقسم السواقة الصلبة إلى عدة سواقات منطقية

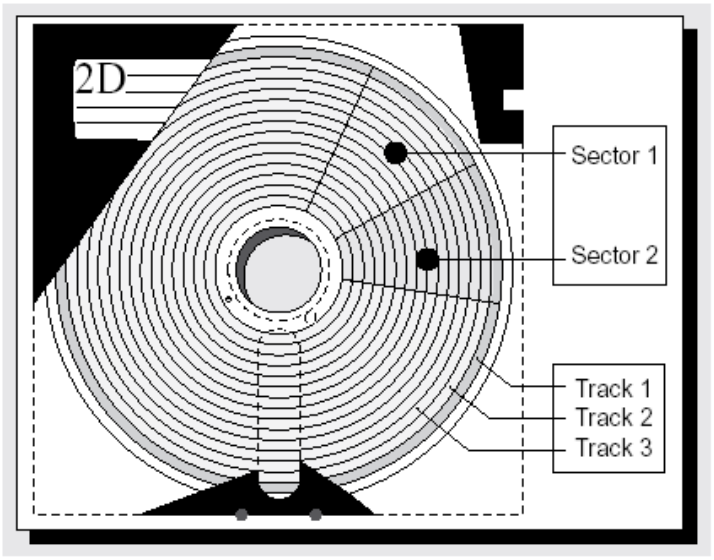
12 -1 - بنية القرص المرن و السواقة الصلبة

بداية لنلقي نظرة على الخصائص المشتركة للأقراص المرنة و السواقات الصلبة. تمتلك الأقراص المرنة و السواقات الصلبة مواصفات مماثلة و التي يعبر عنها بأرقام توابع الـBIOS و التي تطبق عليهما معا، لو تخيلنا و تأملنا أن القرص المرن هو إصدار ثنائي البعد من السواقة الصلبة , فإن التشابهات بينهما ستكون واضحة جلية.

12 -1 -1 - بنية القرص المرن

يتكون القرص المرن من مسارات tracks مستقلة مرتبة كدوائر متحدة المركز وبفواصل متساوية على سطح القرص المغناطيسي. هذه المسارات tracks مصنفة و مرقمة من 0 إلى N، حيث أن N تمثل العدد الكلي للمسارات tracks مطروحا منه 1 و تتنوع اعتمادا على

Structure of a 5.25-inch diskette



الشكل132

يقسم كل مسار track ثانية إلى عدد ثابت من القطاعات sectors , و كل قطاع sector يملك نفس الحجم من المعطيات.ترقم القطاعات من 1 إلى N،حيث أن N تمثل العدد الكلي للقطاعات في المسار.

يعتمد العدد الأعظمي للقطاعات في المسار على نوع سواقة القرص المرن و بنية القرص المرن. يحتوي كل قطاع على 512 bytes و هذه هي أصغر كمية من المعطيات و التي يستطيع أي برنامج الوصول لها و التعامل معها. بمعنى آخر , يجب عليك أن تقرأ أو تكتب قطاعا كاملا في المرة الواحدة. حيث أنه لا توجد إمكانية لقراءة أو كتابة بايت وحيد من القرص المرن،و بشكل مماثل للأقراص المرنة ذات الكثافة المزدوجة.يتم تسجيل المعطيات في كل قطاع باستخدام تقنية FM أو تقنية MFM , و هذه التقنيات هي نفسها المستخدمة في تسجيل المعطيات في السواقات الصلبة، وينبغي عليك كمبرمج أن لا تقلق أبدا حيال تفاصيل التسجيل هذه التي تتم في سواقة الأقراص.

تستعمل المعادلة التالية لحساب سعة القرص المرن, وتذكر أن هذه المعادلة لجهة واحدة فقط من القرص المرن. إذا كانت سواقة القرص المرن رأسي قراءة/ كتابة (مثل غالبية سواقات الأقراص المرنة حاليا) , فإنه ينبغي عليك مضاعفة هذه القيمة. يشير الـDOS إلى جهات القرص المرن و يميزها بـ side 0 و side 1.

المعادلة هي

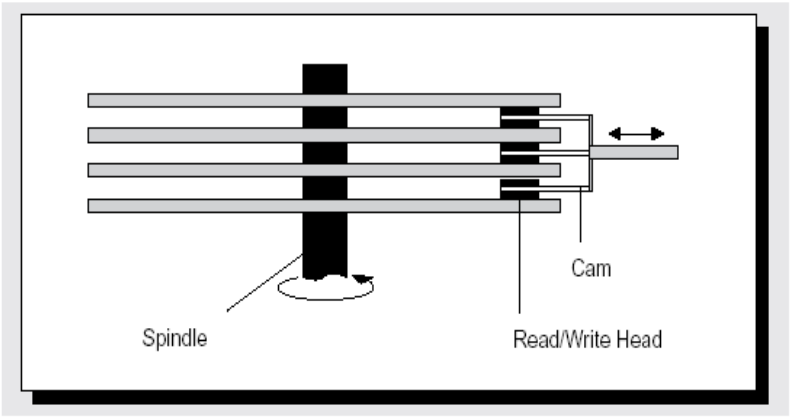
```
Sectors * tracks_per_sector * 512 [bytes per sector]
```

إن عدد القطاعات في المسار يؤثر أيضا في معدل نقل البيانات. يعبر معدل نقل البيانات عن سرعة الكترونيات سواقة القرص المرن وموجهها. مع سرعة دوران ثابتة تقدر بـ300 دورة في الدقيقة , فإن رأس القراءة / الكتابة يقوم بالمرور على بتات أكثر في وحدة الزمن و بالتالي يمكن كتابة قطاعات أكثر في المسار.

12 .1 .2 -بنية السواقة الصلبة

بما أن السواقة الصلبة تدور بشكل أسرع من القرص المرن بعشر مرات , فإن معدل نقل بياناتها سيكون على الأقل أكبر بعشر مرات من معدل نقل البيانات في القرص المرن. يزداد معدل نقل البيانات بمعدل ثانية مرفوعة للقوة 10 , لأن السواقات الصلبة ذات 3.5-inch تستطيع تخزين 100 قطاع لكل مسار. هذه الخصائص لا تغير البنية الأساسية للسواقة الصلبة. بإمكاننا تخيل السواقة الصلبة بأنها مجموعة من الأقراص المغناطيسية المكدسة فوق بعضها البعض. كل قرص مغناطيسي يشبه القرص المرن , و له جهتين , كما أنه مقسم إلى مسارات , و كل مسار يقسم إلى قطاعات . كما أنه يوجد فوق سطح كل جهة من القرص رأس قراءة / كتابة و الذي يقوم بالولوج إلى البيانات. تصطف الأقراص بحيث أن المسار 0 لقرص ما يكون فوق المسار 0 للقرص الآخر تماما و تقوم ذراع القراءة/الكتابة بربط جميع رؤوس القراءة/ الكتابة مع بعضها. للوصول إلى مسار محدد لواحد من الأقراص , تحرك الذراع كل رؤوس القراءة/ الكتابة للمسار المحدد. و نظرا لأن هذا التنظيم يتطلب فقط آلية توضع وحيدة (ذراع القراءة/ الكتابة) , فإنه يبسط التصميم و يقلل من كلفة السواقة الصلبة. ولكن من ناحية أخرى , في هذا التنظيم , يجب أن تتحرك جميع رؤوس القراءة / الكتابة لتصل إلى البيانات في مسار مختلف. لذلك , لقراءة البيانات من المسار 1 لقرص ما , و بعدها البيانات من المسار 50 لقرص آخر و أخيرا البيانات من المسار 1 للقرص الأول مرة أخرى , فإن ذراع القراءة/الكتابة يجب أن تتحرك مرتين. إن توضع الذراع بهذا الشكل كمية هامة من الزمن فيما يخص زمن نقل البيانات.

Hard drive structure



الشكل133

لتقليل الزمن المطلوب للوصول إلى البيانات , يجب عليك أن تمنع البيانات من أن تنتشر عبر مسارات عديدة. هناك طريقة لتحسين زمن الوصول لمجموعة من البيانات ألا و هي كتابة البيانات بشكل متسلسل على المسار. و إذا لم يكف مسار واحد لكتابة هذه البيانات , تتم الكتابة على نفس المسار من قرص آخر. بهذه الطريقة , فإنه لا يوجد حاجة لأن تتحرك ذراع القراءة/الكتابة. حيث أن اختيار (تغيير) الرؤوس أسرع بكثير من التحريك الفيزيائي لذراع القراءة/الكتابة الميكانيكية لتغيير المسارات.

يستخدم مصطلح أسطوانة cylinder للتعبير عن أقراص متعددة متكدسة فوق بعضها البعض. و تشير هنا الأسطوانة cylinder لكل المسارات التي لها نفس الأرقام و لكنها متوضعة على أقراص مختلفة.

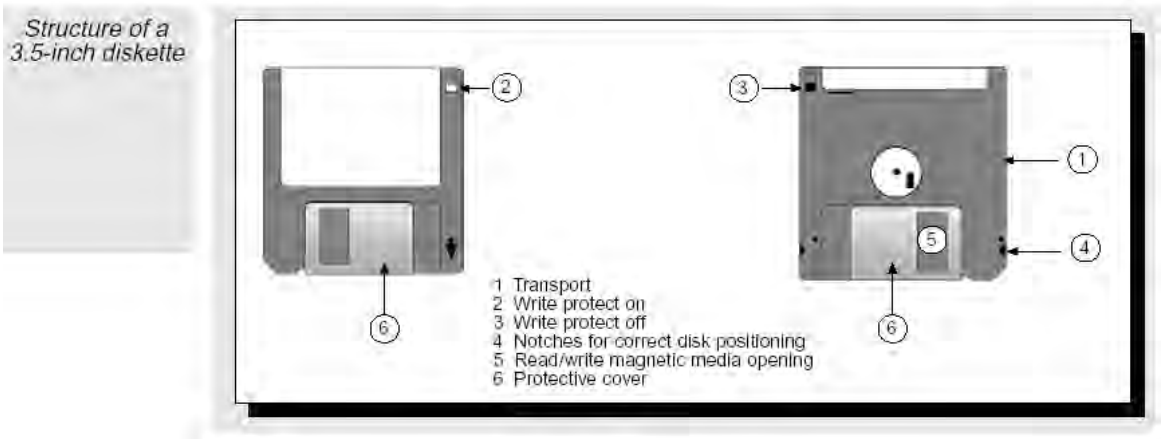
12 2 ـأشكال السواقات الصلبة و الأقراص المرنة

قبل وصف توابع الـBIOS للأقراص المرنة , سنقوم بإلقاء نظرة عامة على أشكال الأقراص المرنة.تمتلك الحواسيب القوية في هذه الأيام على الأقل واحدة من أشكال الأقراص المرنة التالية و حالياً, يعتبر الشكل الأول من أكثر الأشكال الثلاثة شيوعا.

3.5-inch, 1.44 Meg high-density (HD)
3.5-inch Super high density
5.25-inch, 1.2 Meg high-density (HD)

12 2 1- سواقات الأقراص المرنة ذات 3.5-inch

تعبر هذه السواقات الصغيرة الحجم مقارنة بسابقتها أكثر انتشارا في الحواسيب الشخصية و المحمولة. و تعتبر من السواقات المفضلة نظرا لحجمها الملائم و لمتانة الغلاف البلاستيكي الذي يحيطها. تقوم هذه السواقات بتسجيل البيانات إما بطريقة الكثافة المزدوجة أو الكثافة العالية. تكون السعة في الكثافة المزدوجة 720 K , حيث أن عدد القطاعات 9 لكل مسار وعدد المسارات 80 في الجهة الواحدة. و لكن الشكل الأكثر انتشارا من هذا النوع من السواقات هي السواقة 3.5-inch ذات السعة 1.44 Meg , حيث أنها تملك أيضا 80 مسارا في كل جهة و لكن تم مضاعفة عدد القطاعات إلى 18 قطاع لكل مسار. هذا يؤدي إلى مضاعفة معدل نقل البيانات.



الشكل 134

تعمل سواقات الأقراص 3.5-inch ذات الكثافة العالية تماما كالسواقات 5.25-inch من النوع MF حيث أنها يمكن أن تضبط لتستطيع قراءة , كتابة وتهيئة الأقراص مزدوجة الكثافة.يوجد حاليا سواقات 3.5-inch بسعة 2.88 Meg تدعى الأقراص ذات الكثافة العالية الإضافية (ED). الفرق فيها أنه تم مضاعفة عدد القطاعات إلى 36 عن سابقتها و يمكن قراءتها فقط عن طريق سواقات أقراص (ED) الجديدة و هذه السواقات قادرة على معالجة الكثافة المزدوجة و الكثافة العالية. إن سواقات الأقراص المرنة التي تستطيع قراءة وكتابة أشكال مختلفة من الأقراص يجب أن تكون قادرة على تحديد نوع القرص و تقوم باستخلاصها من القرص , بعد ذلك تستطيع تمرير هذه المعلومات للـBIOS قبل الولوج للبيانات على القرص المرن.

إن إيجاد القرص المرن 5.25-inch ليس بالأمر السهل أبدا , لأنه يتم تحديد المعلومات اللازمة فقط عن طريق قراءة البيانات. و بالتالي , يجب أن يكون القرص مهيأ بشكل مسبق. من

ناحية أخرى , يمكن تحديد القرص المرن 3.5-inch عن طريق ثقب صغير, يتوضع هذا الثقب على الجهة الأخرى من جهة التي يوجد بها الحماية للقرص من الكتابة. يوجد ضمن سواقة القرص المرن نفسها حساس يستطيع اكتشاف حضور أو غياب الثقب , و على عكس القرص المرن ذو الكثافة العالية , فإن القرص المرن ذو الكثافة المزدوجة لا يمتلك هذا الثقب و كما أن القرص المرن ذو الكثافة المرتفعة الإضافية يمتلك ثقبا و لكنه موجود في مكان آخر.

12 2 2 -سواقات القرص و متحكماتها

كما رأينا أن السواقة المرنة تتكون من محرك يتولى عملية تحرك القرص بمعدل 300 دورة في الدقيقة , و تقنية لتحريك رأس القراءة و الكتابة بالإضافة إلى مكون الكتروني يدعى فاصل البيانات data separator , و الذي يقوم بتحويل الجهد إلى حزمة من البيانات الثنائية.يتم التحكم و توجيه القرص الصلب عن طريق موجه القرص المرن diskette controller , والذي إما أن يكون جزء من اللوحة الأم للكومبيوتر أو بطاقة دخل/خرج في واحد من شقوق التوسيع expansion slot للكومبيوتر.

12 3 -الوصول إلى سواقات الأقراص المرنة عن طريق BIOS

يوجد مجموعة كاملة من توابع BIOS و التي تمكننا من التعامل و الوصول إلى سواقات الأقراص المرنة. و تستخدم المقاطعة 13H لاستدعاء هذه التوابع. كما أن هذه المقاطعة تعتبر واجهة interface لتخديم السواقات الصلبة. و تتشارك غالبا السواقات الصلبة و المرنة في نفس أرقام التوابع , و للتمييز فيما إذا كانت السواقة التي تقوم باستدعاء التابع هي السواقة الصلبة أو المرنة فإنه ينبغي علينا تمرير نوع السواقة للتابع الموجود في المسجل DL.

بالنسبة للسواقة المرنة يتم استخدام القيمة 0 (للسواقة A) و القيمة 1 (للسواقة B) , و هناك بعض المتحكمات تدعم 4 سواقات مرنة بحيث يقبل الـBIOS القيمة 2 و 3 للسواقات المرنة الأخرى. أما بالنسبة للسواقات الصلبة فإنه يتم تحديدها بالقيم 80H و 81H، وهنا يجب التمييز أيضا بين PC/XT-BIOS و AT-BIOS. , الجدول التالي يعرض توابع القرص المرن للمقاطعة 13H.لاحظ أن التابعان 17H و 18H يقومان بنفس الوظيفة , هذا ليس خطأ , حيث أن التابع 18H تم تقديمه للسواقات المرنة 3.5-inch , لأن التابع الأقدم 17H غير قادر على دعم هذه السواقة , لذلك تم استبداله بالتابع الجديد.

Diskette functions of the BIOS interrupt 13H							
No.	Tasks	PC/XT	AT	No.	Tasks	PC/XT	AT
00H	Reset	Yes	Yes	08H	Request Format	Yes	Yes
01H	Read status	Yes	Yes	15H	Define drive type	No	Yes
02H	Read	Yes	Yes	16H	Detect diskette change	No	Yes
03H	Write	Yes	Yes	17H	Determine diskette format	No	Yes
04H	Verify	Yes	Yes	18H	Determine diskette format	No	Yes
05H	Format	Yes	Yes				

الشكل 135

12 3 1- حالات السواقة

يوجد وظيفة أخرى لتوابع BIOS حيث أنها تقوم بإرجاع الحالات أو شفرة الخطأ, ويتم إرجاع هذا الشفرة للمستدعي في المسجل AH. إن القيمة الصفرية و تفعيل علم الحمل يشيران لوجود خطأ.

Status and error codes of the BIOS diskette functions			
Code	Meaning	Code	Meaning
00H	No error	08H	DMA overflow
01H	Illegal function number	09H	Data transfer past the segment limit
02H	Address marking not found	10H	Read error
03H	Attempt to write to write-protected diskette	20H	Diskette controller error
04H	Addressed sector not found	40H	Track not found
06H	Diskette was changed	80H	Time out error, drives does not respond

الشكل136

بإمكاننا أيضا تحديد حالة القرص المرن عن طريق التابع 01H. و يتم ذلك بأن تمرر رقم التابع 01H إلى المسجل AH و نوع السواقة (صلبة أو مرنة) إلى المسجل DL , و هذا يمثل التهيئة الابتدائية , و من ثم نقوم باستدعاء التابع , و بعد استدعاء التابع فإنه يتم إرجاع حالة السواقة إلى المسجل إليك عن طريق المسجل AH.

12 3 2- إعادة تهيئة Resetting القرص المرن

يجب عليك أن تعيد تهيئة reset سواقة القرص المرن بعد حدوث أي خطأ. للقيام بذلك استعمل التابع 00H. قم بتمرير رقم التابع 00H للمسجل AH و نوع السواقة للمسجل DL. و بعد استدعاء التابع فإنه سوف يتم إرجاع حالة السواقة الحالية في المسجل AH. لا يوجد هناك أي مشكلة إذا مررت 0 أو1 كنوع للسواقة , ذلك لأنه سيتم إعادة تهيئة جميع السواقات المرنة. لكن انتبه جيدا للقيمة التي تمررها للمسجل DL فإنها تؤخذ بعين الاعتبار , حيث إذا مررت قيمة أكثر من 80H " يعطيك العافية " , لأنه سيتم إعادة تهيئة السواقة الصلبة.

12 3 3- نمط و شكل سواقة الأقراص المرنة

إن البرنامج بحاجة إلى أن يعرف نمط و شكل سواقة الأقراص المرنة للتعامل معها. لتحديد هذه المعلومات يجب عليك استخدام التوابع 08H و 15H. حيث أن التابع 08H يستخدم للتمييز بين الأنواع المختلفة للسواقات المرنة. و كما تعلمنا قم بتمرير رقم التابع 08H إلى المسجل AH و نوع السواقة (صلبة أو مرنة) على المسجل DL،أما بالنسبة للمعلومات المرجعة فهي تكون على الشكل،إذا كان يوجد أي خطأ فسيتم إرجاع شيفرته إلى المسجل AH و تفعيل علم الحمل.

سيتم إرجاع قيمة إلى المسجل BL هذه القيمة هامة جدا فهي تبين نوع السواقة فيما إذا كانت 5.25-inch أو 3.5-inch و الأهم من ذلك أنها تبين الكثافة (مزدوجة أو عالية) , و لكن

انتبه فليس من الضروري أن تصف هذه القيمة كثافة القرص الموجود في السواقة , و لكنها تصف أكبر كثافة ممكنة ,معلومات عن عدد الجهات و المسارات و القطاعات في المسجلات CH , CL , DH.مؤشر موجود في زوج المسجلات ES:DI يؤشر إلى DDPT و الذي يمثل جدول لباراميترات سواقة الأقراص.

Information returned by the 08H function	
Register	Information
BL	Drive Type 01H = 5.25-inch, 360K 02H = 5.25-inch, 1.2 Meg 03H = 3.5-inch, 720K 04H = 3.5-inch, 1.44 Meg
DH	Maximum number of sides (always 1)
CH	Maximum number of tracks
CL	Maximum number of sectors
ES:DI	Pointers to DDPT

الشكل137

سنتحدث الآن عن التابع 15H , هذا التابع مدعوم فقط من قبل ATs و سواقاتها التي من النوع FM , فهذه السواقات قادرة على اكتشاف فيما إذا تم تغيير القرص المرن أم لا. هذه الخاصية مهمة جدا للبرامج التي تعتمد على معالجة القرص الحالي و التعامل معه عندما يكون القرص موجودا في السواقة , و ينطبق هذا الكلام خاصة على DOS , حيث أن هذه البرامج تقوم بقراءة جدول FAT قبل الولوج لبيانات القرص و تحدد أي القطاعات في هذا القرص محجوزة و أيها لا يزال غير مستخدم. لذلك إذا تم تغيير القرص من غير أن يعلم DOS بذلك فإن DOS سيتابع باستعمال المعلومات الخاصة بالقرص السابق و يتعامل مع القرص الحالي على هذا الأساس و يمكن أن يقوم و بشكل غير مقصود بالكتابة فوق معلومات سابقة overwrite و إفساد محتويات هذا القرص. لذلك كان لا بد من معرفة DOS إذا تم تغيير القرص أم لا. لنعود على التابع 15H , إن هذا التابع هو من توابع BIOS كما قلنا و يقوم بتحديد فيما إذا تم تبديل القرص أم لا و يتم تهيئة القيم الابتدائية في المسجلات كما في التوابع السابقة و من ثم إرجاع القيمة المطلوبة, و يعرض الجدول التالي المعلومات التي يتم إرجاعها بعد استدعاء هذا التابع.

Drive codes of function 15H	
Code	Meaning
AH = 00H	Drive not present
AH = 01H	Disk drive, does not recognize diskette changes
AH = 02H	Disk drive, recognizes diskette changes
AH = 03H	Hard drive

الشكل138

12 3 4 -قراءة قطاعات القرص المرن

يقوم التابع 02H بقراءة قطاعات القرص. إن هذا التابع يستطيع قراءة أكثر من قطاع بشرط أن تكون هذه القطاعات على نفس المسار و بجانب بعضها البعض، وتجدر الإشارة إلى أن البيانات لا يتم نقلها إلى عنوان ذاكري ثابت , لذلك فإنه يتم تمرير عنوان الـbuffer إلى زوج المسجلات ES:BX , حيث أن المسجل ES يحتوي عنوان القطعة segment للـbuffer , و المسجل BX يحتوي على عنوان إزاحة الـbuffer، بعد استدعاء هذا التابع فإنه سيتم إرجاع حالة الخطأ إلى المسجل AH و عدد القطاعات المقروءة إلى المسجل AL. و إذا تم تفعيل علم الحمل فإن هذا يشير إلى وقوع خطأ والجدول التالي يبين المسجلات التي تتأثر باستدعاء التابع 02H.

Register when calling function 02H	
Register	Information
AL	Number of sectors to be read
DL	Drive specification value
DH	Side (0 or 1)
CL	Sector number (1 to N)
CH	Track number (0 to N-1)
ES:BX	Address of the buffer for the data to be read
AH = 03H	Hard drive

الشكل 139

إذا كنت تستخدم سواقات MF , بإمكانك القيام بحيلة ذكية لتحدد نوع القرص المرن. إذا جربت قراءة قرص ما و استخدمت لذلك قيمة قطاع أكبر من 9 , فعندها تستطيع أن تحدد فيما إذا كان هذا القرص DD أو HD. لأنه و كما نعلم أن العدد الأعظمي للقطاعات في المسار في الأقراص من النوع DD (الكثافة المزدوجة) 9 , و بالتالي سيقوم التابع 15H بإرجاع قيمة تدل على خطأ , طبعا قيمة المسار غير ضرورية و لكن كما نعلم يجب أن تكون أقل من 40.الآن انتبه عند إرجاع التابع لقيمة تدل على وقوع خطأ و لا تقوم مباشرة بإيقاف العملية الحالية التي تقوم بها , و لكن يجب عليك أن تعيد تكرار هذه العملية (سواء أكانت قراءة أو كتابة أو تهيئة) 3 مرات على الأقل قبل أن تستسلم و تجزم بأن هناك خطأ حقيقي قد حدث. عادة , تفشل العملية التي تريد القيام بها في المرة الأولى , و لكنها تنجح في المحاولة الثانية أو الثالثة , و ربما يكون السبب في ذلك أن رأس القراءة /الكتابة لا يكون قد وضع في الموضع المحدد في المرة الأولى , أو أن السواقة المرنة لم تكون قد تزامنت حتى هذه اللحظة مع الالكترونيات. لا تقلق بشأن صحة البيانات عند وقوع خطأ ما لأن السواقة تتأكد عن طريق تساوي و تماثل المعلومات لضمان صحة البيانات في كل قطاع.

12 3 5 -الكتابة على قطاعات القرص المرن

يستخدم التابع 03H للكتابة على قطاعات مستقلة. يبين الجدول التالي الباراميترات التي تمرر لهذا التابع.

Register when calling function 03H	
Register	Information
AL	Number of sectors to be written
DL	Drive specification value
DH	Side (0 or 1)
CL	Sector number (1 to N)
CH	Track number (0 to N-1)
AL	Number of sectors to be read
ES:BX	Pointer to the buffer containing the data

الشكل140

12 3 6 -التأكد من قطاعات القرص المرن

يقوم التابع 04H بفحص و اختبار فيما إذا تم نقل البيانات إلى القرص المرن بشكل صحيح. لا يتم مقارنة البيانات الموجودة بالذاكرة بالبيانات الموجودة على القرص. بل يتم استخدام قيمة CRC التي تحدد فيما إذا تم نقل البيانات بالشكل الصحيح. CRC هي اختصار لـ Cyclical Redundancy Checkو هي عبارة عن إجرائية موثوقة جدا و تقوم بالتأكد من ذلك بدقة كبيرة. حيث أنها تقوم بعمليات و معادلات رياضية معقدة و تقوم بتجميع القيم لكل بايت ضمن القطاع الواحد مع الـcheck sum و تطبق عليها هذه المعادلات الرياضية.

إن الباراميترات للتابع 04H هي نفسها الموجودة في التوابع 02H و 03H باستثناء عنوان الـbuffer فهو لا يلزم. حالياً , تعتبر أغلب السواقات موثوقة , لذلك فإن غالبية المبرمجين يعتبروا أن هذا الإجراء غير ضروري و لا حاجة لاستعماله.

12 3 7 -تهيئة مسارات مستقلة ضمن القرص المرن

يستخدم التابع 07H لتهيئة القرص المرن كاملا. و لكن هناك إمكانية لتهيئة مسارات مستقلة ضمنه. للقيام بذلك , ينبغي عليك أولا أن تستخدم التابع 18H لإخبار BIOS بنوع السواقة. يمرر رقم التابع في المسجل AH , و نوع السواقة في المسجل DL , و عدد المسارات في المسجل CH و عدد القطاعات في المسجل CL. و بعد استدعاء التابع فإن علم الحمل يشير إلى النوع المحدد المدعوم بواسطة السواقة المرنة. في هذه الحالة يكون زوج المسجلات ES:DI مؤشر لـDDPT و الذي تحتاجه توابع التهيئة اللاحقة.

بعد أن يقوم التابع 07H بتحديد النوع المطلوب و بعد أن يتم تمرير مؤشر DDPT لشعاع المقاطعة 1EH , عندها بإمكانك القيام بعملية التهيئة الحقيقية. لتبدأ بالتهيئة , استعمل التابع

05H , هذا التابع يقوم بتهيئة مسار كامل. و على الرغم من أنه بإمكانك تهيئة قطاعات مستقلة بسعة 512,256,128 أو حتى 1024 بايت لكل قطاع , فإنك لا تستطيع أن تهيئ إلا 512 بايت فقط تحت DOS و ذلك لأن DOS يدعم فقط هذه السعة للقطاعات. قم باستخدام التابع 05H ومرر القيم إلى المسجلات كما هو مبين بالجدول التالي:

Register when calling function 05H	
Register	Information
AL	Number of sectors in the track
DL	Number of the drive
CH	Number of the track
DH	Side (0 or 1)
ES:BX	Pointer to format table

الشكل141

يشير زوج المسجلات ES:BX على "جدول التهيئة". إن هذا الجدول يمثل خصائص التهيئة , و كما نرى من الشكل أن هذا الجدول هو عبارة عن مصفوفة من مؤلفة من كيانات أو سجلات لكل قطاع سيتم تهيئته , و السجل الواحد مؤلف من 4 بايتات. و على الرغم من أنه يتم تمرير رقم المسار و جهة القرص المرن إلى التابع 05H , فإنه يجب تكرار ذلك في الجدول.

ES:BX register pair "format table"	
Offset	Meaning
0	Track to be formatted
1	Diskette side (always 0 for one-sided diskettes): 0 = Front side 1 = Back side
2	Number of the sector
3	Number of bytes in this sector: 0 = 128 bytes 1 = 256 bytes 2 = 512 bytes 3 = 1024 bytes

الشكل 142

يتم إنشاء القطاعات فيزيائيا بالتسلسل نفسه لمكونات الجدول. و بالتالي من الممكن تهيئة الكيان الأول على أنه القطاع رقم 1 و الكيان الثاني على أنه القطاع رقم 7. يسجل الرقم المنطقي للقطاع في ترويسة header كل قطاع على القرص المرن. و تجدر الملاحظة أنه ليس من الضروري أن يكون عدد البايتات في كل قطاع هو نفسه , لذلك فإنه يجب أن يتم تحديد أعداد

البايتات بشكل صريح في كل قطاع من الجدول , من هنا نستنتج أنه بإمكاننا أن نغير هذه الأعداد في كل قطاع بمعنى أن نحدد لكل قطاع عدد بايتات يختلف عن القطاع الآخر و ذلك كنوع من حماية النسخ.

جدول باراميترات سواقة القرص Disk Drive Parameter Table (DDPT)

حتى نكون قادرين على برمجة موجه القرص المرن , فإن BIOS بحاجة لمعرفة معلومات التهيئة الفيزيائية و التي وصفناها في الأعلى , بالإضافة لمعلومات أخرى. تحتوي ROM BIOS على جدول يحتوي كل أنواع السواقات و الأقراص المرنة التي يدعمها. و أيضا بإمكانك أن تعرف DDPT خاص بك , وبما أن BIOS يرجع DDPT الحالي عبر مؤشر FAR , و الذي يكون موجودا ضمن المواقع الذاكرية التي يتواجد فيها شعاع المقاطعة 1EH. ونظرا لأن DOS و العتاد الصلب للكومبيوتر الشخصي كلاهما لا يستخدمان المقاطعة 1EH , فإنه بالإمكان تغيير محتوى هذه المواقع الذاكرية.إن DOS يقوم بإنشاء DDPT الخاص به , و مهمته زيادة سرعة الوصول إلى القرص المرن. إن حجم هذا الجدول 11 بايت , و كما نرى من الشكل التالي الذي يوضح الجدول:

Diskette functions of the BIOS interrupt 13H					
Offset	Meaning	Type	Offset	Meaning	Type
*00H	Step rate and head unload time	1 BYTE	06H	DTL (Data Length)	1 BYTE
*01H	Head load time	1 BYTE	07H	Length of GAP3 when formatting	1 BYTE
*02H	Post run-time of diskette motor	1 BYTE	*08H	Fill character for formatting	1 BYTE
03H	Sector size	1 BYTE	*09H	Head settle time	1 BYTE
04H	Sectors per track	1 BYTE	*0AH	Time to run up of diskette motor	1 BYTE
05H	Length of GAP3 when reading/writing	1 BYTE			

الشكل 143

ليس بالإمكان تغيير كل البارامترات , إنما فقط بإمكاننا تغيير البارامترات المعلمة بالإشارة*. إن الحقل الأول من جدول DDPT له حقلين فرعيين , الأول هو معدل الخطوة (bits 4-7) و الثاني هو زمن تفريغ الرأس (bits 0-3). يصف معدل الخطوة الزمن الذي يستغرقه الموجه لتحريك رأس القراءة/الكتابة من مسار إلى آخر , و هذه القيمة تمثل بالملي ثانية , حيث أن القيمة 0FH تمثل 1 ms , 0EH تمثل 2 ms و0DH تمثل 3 ms. أما زمن تفريغ الرأس فهو يصف الزمن الذي يستغرقه رأس القراءة/الكتابة ليرتفع عن سطح القرص المرن , و يعبر عن هذه القيمة كمعاملات لـ 16 ms. و القيمة الافتراضية هي (240 ms) 0FH.أما الحقل الثاني فهو أيضا ينقسم إلى حقلين فرعيين , زمن تحميل الرأس (bits 1-7) و علم DMA (bit 0). بالنسبة لزمن تحميل الرأس فهو الزمن الذي يستغرقه رأس القراءة/ الكتابة ليستقر على سطح المسار , و يعبر عن هذه القيمة كمعاملات لـ 2 ms. أما بالنسبة لعلم DMA فهو يجب أن يأخذ دائما القيمة 0.أما بالنسبة للحقل الثالث فهو يعبر عن الفترة الزمنية التي تنقضي حتى يتم توقف محرك القرص المرن عند عدم وجود أي عملية لها علاقة بالقرص المرن و ينتظر تنفيذها. و نظرا لأن محرك القرص المرن يستغرق فترة من الزمن لا بأس بها حتى يقلع بشكل كامل , فإنه لا ينبغي علينا أن نقوم بإيقافه مباشرة بعد كل عملية وصول للقرص المرن. هذه القيمة تتعلق بالدور الذي يساوي تقريبا 18 ticks per second حيث أن

(1 tick is approximately 55 ms). و القيمة الافتراضية هي 25H و التي تساوي تقريبا 2 ms.الحقل الرابع يحدد عدد البايتات في كل قطاع و التي يمكن استخدامها في عملية القراءة و الكتابة , وهي تتوافق بشكل رئيسي و تطابق قيم تهيئة القطاع , و لذلك فهي عادة تحتوي القيمة 3 من أجل 512 بايت لكل قطاع. للقراءة من أو للكتابة في قطاعات لها سعات أو حجوم مختلفة , ينبغي عليك أولا أن تضع القيمة المناسبة في هذا الحقل ثم تقوم بالعملية المطلوبة.الحقل التالي الموجود عند الإزاحة الذاكرية 04H هو العدد الأعظمي للبايتات في كل قطاع , و الذي يعتمد على نوع القرص المرن.أما بالنسبة للحقول الثلاثة التالية فهي تتعلق بالترميز و فك الترميز لمعلومات القطاع , و التي تخزن على القرص المرن مع البيانات الحقيقية. انتبه جيدا و لا تحاول العبث أو التأثير على هذه القيم أبدا. أما بالنسبة للحقل الموجود عند الإزاحة 08H يمكن تغييره , و يحتوي هذا الحقل على رمز ASCII الذي يعبر عن المحرف الذي نريد أن نملأ به القطاعات بعد عملية التهيئة , حيث أنه عند التهيئة (و التي يتم فيها طبعا خلق القطاعات) تعطى القطاعات محتوى ثابت. و القيمة الافتراضية هي إشارة القسمة (ASCII code 246).

يحتوي الحقل التالي وقت استقرار الرأس. فبعد أن ينتقل رأس القراءة/الكتابة من مسار لآخر , يكون بحاجة لتأخير زمني قصير ليتم تخميد الاهتزازات التي تنتج عن هذه الحركة. و بعد ذلك يستطيع رأس القراءة/الكتابة إنجاز عمليات الولوج للبيانات اللاحقة بشكل مناسب و صحيح. القيمة الافتراضية لهذا الحقل 25 ms.يعبر الحقل الأخير عن الزمن الذي يستغرقه محرك القرص المرن لكي يصل سرعة التشغيل , و هذه القيمة من معاملات و أجزاء 1/8 seconds.

12 4 -استعمال الـ BIOS للوصول إلى الأقراص الصلبة

سوف نصف في هذا البحث توابع Bios الخاصة بالوصول إلى سوا قات القرص الصلب hard drives.مهما يكن,قبل أن نبدأ يجب أن نحذرك بشأن تجريب هذه التوابع.هناك اختلاف بينها وبين سواقة القرص المرن floppy disk drive التي من خلالها يمكنك أن ستعمل قرص غير مرن من أجل التجريب فلا يمكنك اختبار القرص الصلب بهذه الطريقة.سيقودك الاستخدام الغير مبالي لتوابع التهيئة والكتابة إلى فقدان البيانات وسيتعذر عليك إصلاحه.لأن هيكلية نظام التشغيل DOS سوف تفرض نفسها على القرص الصلب,حيث أن هدم قطاع واحد one sector يمكن أن يؤدي إلى اختفاء جميع المسارات و الملفات وذلك لأن نظام التشغيل DOS لم يعد يعرف أين مكانهم على القرص الصلب hard disk.

لذلك إذا أردت أن تختبر توابع الـ BIOS تأكد من تنشأ نسخة احتياطية backup لقرصك الصلب مقدما.أو تستخدم حاسب آخر إذا كان ذلك متاحا لك.تلك هي الطريقة الوحيدة لتتجنب ضياع المعلومات,لأن حتى خدمة القرص الصلب المحكم قد لا تستطيع مساعدتك.

12 4 1- مقاطعة القرص الصلب من أجل الـ BIOS

كما ذكرنا سابقاً تتشارك سواقات القرص الصلب hard drives مع سواقات القرص المرن floppy drive من حلال المقاطعة 13H.على الرغم من أن التوابع الخاصة بسواقة القرص الصلب hard drive والقرص المرن floppy drive هي متطابقة,فإن الـ BIOS تتحكم بسواقة القرص الصلب hard drive بشكل مختلف عن تحكمها بسواقة القرص المرن floppy drive.لهذا السبب تتضمن الـ BIOS نموذجا للتحكم بسواقة القرص الصلب hard drive منفصل عن نموذج التحكم بسواقة القرص المرن.

عندما يتم استدعاء المقاطعة 13H فإن رقم الجهاز في المسجل DL يحدد فيما إذا كان هي عنونة للقرص الصلب أو القرص المرن..تمثل القيمة 80H أن سواقة القرص الصلب الأولى هي المطلوبة في حين تمثل القيمة 81H أن سواقة القرص الصلب الثانية هي المطلوبة.لا يمكن أن نقوم بعنونة أكثر من سواقتي قرص صلب من خلال الـ BIOS. وُجدت توابع الـ BIOS الخاصة بالقرص الصلب منذ مقدمة XT.إن الـ BIOS الخاصة بالحاسب العادية لا تمتلكهم.في عام 1981 لم يفكر أحد في وضع سواقات القرص الصلب في الحواسب المصغرة.عندما تم طرح الحواسب AT & PS/2 من قبل شركة IBM فقد أضيفت بعض التوابع الإضافية كما هو مبين في الجدول التالي:

Function	Task	Origin	Function	Task	Origin
00H	Reset	XT	0CH	Move read/write head	XT
01H	Read status	XT	0DH	Reset	XT
02H	Read	XT	0EH	Controller read test	only PS/2
03H	Write	XT	0FH	Controller write test	only PS/2
04H	Verify	XT	10H	Drive ready?	XT
05H	Format	XT	11H	Recalibrate drive	XT
08H	Check format	XT	12H	Controller RAM test	only PS/2
09H	Adapt to foreign drives	XT	13H	Drive test	only PS/2
0AH	Extended read	XT	14H	Controller diagnostic	XT
0BH	Extended write	XT	15H	Determine drive type	AT

الشكل144

لتتصل البرامج التطبيقية العادية إلى سواقة القرص الصلب من خلال الـ BIOS.سوف نوضح فقط التوابع الأكثر أهمية في هذه البحث.

12 4 2 -شفرة الحالة

تستخدم توابع القرص الصلب علم الحمل carry flag لتشير إلى وجود خطأ.إذا كان علم الحمل يساوي واحد إذا هناك خطأ قد حدث وتُعاد شفرة الحالة للخطأ في المسجل AH.يبين الجدول التالي معاني هذه الشيفرات:

شيفرات الخطأ عندماتستدعي BIOS المقاطعة 13 من أجل الوصول للقرص الصلب			
الشفرة	الوظيفة	الشفرة	الوظيفة
00h	لا يوجد خطأ	10h	خطأ قراءة
01h	رقم التابع أو المشغل غير مسموح	11h	تم تصحيح خطأ قراءة من قبل ECC
02h	لم يجد العنوان	20h	يوجد خلل بالمتحكم
04h	لم يجد عنوان القطاع	40h	فشلت عملية البحث
05h	خطأ في عملية Reset للمتحكم	80h	انتهى الوقت,الوحدة لم تستجيب
07h	خطأ أثناء تهيئة المتحكم	AAh	الوحدة غير جاهزة
09h	خطأ إرسال DMA, تم	CCh	خطأ كتابة

	تجاوز حدود القطعة		
0Ah	يوجد خلل في القطاع		

عندما يحدث أحد هذه الأخطاء(عد الخطأ الأول) فيجب عليك أن تقوم بعملية reset لسواقة القرص الصلب ومن ثم تجرب التابع من جديد.عادة تكون العملية ناجحة.إذا أعيد الخطأ 11H بعد تابع القراءة فإنه ليس من الضروري أن تكون البيانات لاغية.بشكل فعلي فإن شفرة هذه الحالة تشير إلى أن قد تم كشف خطأ قراءة لكن يمكن تصحيحه باستعمال خوارزمية شفرة تصحيح الخطأ ECC(Error Correction Code).هذه الإجرائية شبيهة بإجرائية CRC التي يتم استخدامها من قبل سواقة القرص المرن.تُحسب البايتات الفردية من القطاع sector من خلال صيغة رياضية معقدة.يكتب المجموع الناتج إلى القطاع على القرص الصلب كأربعة بايتات إضافية.إذا تم كشف خطأ قراءة يمكن تصحيحه باستعمال خوارزمية ECC.

12 4 3 -استخدام توابع القرص الصلب

تستخدم توابع القرص الصلب أيضا المسجلات من أجل تمرير البارامترات.يمرر رقم التابع في المسجل AH.عندما يكون من الواجب التعريف عن رقم سواقة القرص الصلب,فإن قيمته تمرر في المسجل DL.إن القيمة 80H تمثل سواقة القرص الصلب الأولى في حين تمثل القيمة 81H سواقة القرص الصلب الثانية.تُمرر عدد رؤوس القراءة والكتابة وأي وجه من القرص(0 أو 1) في المسجل DH. يخصص المسجل CH من أجل عدد الاسطوانات cylinders.بما أنه يمكنك تمثيل فقط 256 اسطوانة بمسجل ذو 8bits و سواقة القرص الصلب لشركة XT تملك أكثر من 306 اسطوانات فإن هذا المسجل غير كافي لتمثيل عدد الاسطوانات في القرص الصلب.لهذا السبب فإن البتين السادس والسابع من المسجل CL يُضافان إلى القيمة الموجودة في المسجل CH لتحدد عدد الاسطوانات.حيث يصبح العدد الأعظمي للاسطوانات الذي يمكن تمثيله 1024 اسطوانة(ترقم من 0 إلى 1023).تحدد البتات الخمسة الأولى من المسجل CL رقم القطاع sector number (من 1 إلى 17 من أجل كل اسطوانة).إذا كان هناك أكثر من قطاع يمكن الوصول إليه في نفس الوقت فإن المسجل AL يحدد عدد القطاعات.يجب أن تحدد في عمليات القراءة والكتابة عنوان الذاكرة المؤقتة Buffer من أجل البيانات التي سيتم كتابتها أو البيانات التي سيتم نقلها.في هذه الحالة يشير المسجل ES إلى عنوان القطعة والمسجل BX إلى عنوان الإزاحة للذاكرة المؤقتة Buffer.

12 4 4 -القيام بعملية Reset لجهاز التحكم بالقرص الصلب

التابع الوحيد الذي لا يتطلب بارامترات هو التابع 00H الذي يشبه التابع 0dH حيث يقوم بعملية Reset لجهاز التحكم controller.مثلا بعد وقوع خطأ فإن هذا التابع يُنجز بشكل دوري قبل الوصول إلى البيانات التالية.إن البارامتر الوحيد الذي يكون بحاجة إليه هو رقم سواقة القرص الصلب الذي يُمرر بدوره في المسجل DL.

12 4 5 -تحديد حالة سواقة القرص الصلب

باستخدام التابع 01H يمكنك أن تحدد حالة سواقة القرص الصلب.أيضا رقم السواقة التي تريد فحص حالتها يُمرر في المسجل DL.

12 4 6 -قراءة قطاعات القرص الصلب

يقوم التابع 02H بقراءة قطاع أو أكثر من القرص الصلب.عند كل استدعاء لهذا التابع يمكنك قراءة 128 قطاع كعدد أعظمي.ربما أنت متعجب لماذا العدد الأعظمي هو 128 بدلا من 256 قطاع.إن جهاز التحكم الخاص بالقرص الصلب يستخدم إمكانيات ذاكرة الوصول المباشر للذاكرة DMA(Direct Memory Access) لنقل البيانات بين ذاكرة الحاسب والقرص الصلب.مهما يكن فإن عناصر DMA تستطيع أن تنقل 64K من البيانات في نفي الوقت كعدد أعظمي.هذا يكافئ 128 قطاع (64k-128 sectors * 512 byte/sector).هناك قيد آخر هو أن عناصر DMA يمكنها أن تنقل البيانات ضممن قطعة الذاكرة الواحدة.لذلك فإن الذاكرة المؤقتة للقراءة أو الكتابة يشير عادة إلى بداية قطعة الذاكرة.تذكر بأن زوج المسجلات ES-BX يحدد الـ Buffer.في هذه الحالة سيشير المسجل ES لإلى بداية هذه القطعة أما المسجل BX فستكون قيمة الإزاحة المخزنة فيه مساوية للصفر.

عندما تستعمل التابع 02H لتقرأ أكثر من قطاع في كل استدعاء له فإنه يتم قراءة القطاعات بالشكل التالي:أولا,القطاعات في الاسطوانة المحددة وأي جانب تقرأ بترتيب متصاعد(بواسطة رقم القطاع).عندما يتم الوصول إلى نهاية الاسطوانة فإن القطاع الأول على نفس الاسطوانة لكن على الرأس التالي يتم قراءته.لن يتم قراءة القطاعات على الاسطوانة التالية حتى بعد أن يتم الوصول إلى آخر رأس في نفس الاسطوانة ولم يبقى هناك أي قاعات أخرى للقراءة.

12 4 7 -كتابة قطاعات على القرص الصلب

يستخدم التابع 03H من أجل كتابة قطاع أو أكثر على القرص الصلب.هذا التابع مشابه للتابع 02H ماعدا أن البيانات تُكتب من الذاكرة المؤقتة Buffer لإلى القرص الصلب.

12 4 8 -التحقق على قطاعات القرص الصلب

يعرف التابع 04H قطاعات كل اسطوانة.مهما يكن فإن البيانات على القرص الصلب تُقارن مع قيمة ECC بدلا من المقارنة مع البيانات الموجودة ف الذاكرة(التي ليس من الضروري أن نحدد عنوان Buffer في زوج المسجلات ES-BX).يحدد عدد القطاعات التي نود التأكد منها ضمن المسجل AL.

12 4 9 -تهيئة اسطوانات القرص الصلب

يجب أن تتم تهيئة القرص الصلب قبل استخدامه.ينجز التابع 05H هذه المهمة.هذا التابع يشبه التابع المستعمل من أجل تهيئة القرص المرن.عنوان الـ Buffer يُمرر إلى زوج المسجلات ES-BX.يجب أن يكون حجم هذا الـ Buffer هو 512Bytes على الرغم من أن فقط أول 34Byte تُستخدم.يتألف الـ Buffer من مدخلين كل منها بحجم 1Byte من أجل كل قطاع من القطاعات الذي يبلغ عددها 17 قطاع لكل اسطوانة لتتم تهيئته.يشير أول بايت فيما إذا كان القطاع جيد أم سيئ.قبل استدعاء هذا التابع نفترض أن كل قطاع هو جيد.لذلك نخزن القيمة 0 هنا.أما البايت الثاني فهو يشير إلى رقم القطاع المنطقي.يستخدم البايت الأول والثاني من الجدول عندما يكون القطاع الفيزيائي الأول من الاسطوانة قد تمت تهيئته. يستخدم البايت الثالث والرابع من الجدول عندما يكون القطاع الفيزيائي الثاني من الاسطوانة قد تمت

تهيئته,وهكذا....بينما التسلسل الفيزيائي ثابت.يُعرف التتالي المنطقي للقطاعات من قبل بايتين من القطاع يُحددا في هذا الجدول.

إن الطريقة الأكثر وضوحا والمستخدمة في عملية التهيئة هي التي تشير إلى رقم القطاع الفيزيائي للقطاع إلى كل قطاع منطقي.مهما يكن فإن تقنية تستدعي sector interleaving تستخدم فعليا لتسريع أداء القرص الصلب.

يحوي البايت الأول من كل table entry على القيمة 00H التي تشير إلى أن القطاع جيد أو القيمة 80H التي تشير إلى أن القطاع سيئ.أثناء عملية التهيئة Formatting يتم نفل هذا البايت إلى الجزء sector marker الذي يشير إلى أن نظام التشغيل DOS لن يستخدم هذا القطاع ليخزن البيانات.

12 4 10 -تحديد بارامترات القرص الصلب

بشكل غير مشابه للأقراص المرنة فإن الأقراص الصلبة لا تمتلك خصائص موحدة.يكون من المهم لبعض البرامج أن تعرف بارامترات القرص الصلب.لفعل هذا استخدم التابع 08H لتمرر رقم القرص الصلب في المسجل DL.بعد استدعاء التابع فإن المسجل DL يحوي عدد الأقراص الصلبة المتصلة بجهاز التحكم.يتم إرجاع القيمة إما 0 أو 1 أو 2.يحوي المسجل DH عدد رؤوس القراءة والكتابة.بما أن هذه القيمة تبدأ من الصفر فإن القيمة 7 تعني وجود ثمانية رؤوس.يتم إرجاع عدد الاسطوانات في المسجل CL و البتين العلويين من المسجل CH.أيضا هذه القيمة تبدأ من الصفر.أخيرا يتم إرجاع عدد القطاعات في كل مسار track في البتات الست الأقل أهمية من المسجل CH لكن هذه القيمة تبدأ من 1 وليس من الصفر.

12 4 11 -تشغيل قرص صلب نوعيته غريبة

إن الـ BIOS في كل حاسب تتضمن مواصفات عدد من الأقراص الصلبة المتنوعة.هذا يجعل من السهل أن تختار مواصفات القرص الصلب أثناء عملية التنصيب setup.مهما يكن افترض أن مواصفات من أجل قرص صلب محدد غير موجودة ضمن الـ BIOS.عندها يوجد طريقة أخرى لجعل الـ BIOS تتعرف على مواصفات هذا القرص.أولا ينشأ جدول يحتوي على المواصفات.ومن ثم يخزن عنوان الجدول عند المقاطعة 41H أو المقاطعة 46H وذلك اعتمادا على فيما إذا كان رقم القرص الصلب المراد تشغيله 0 أو 1.تنجز صيغة الجدول من قبل الـ BIOS ويصف خصائص القرص الصلب.أخيرا يستدعى التابع 09H الذي يشغل جهاز التحكم مع خصائص القرص الصلب الجديد.يُمرر رقم السواقة (80H أو 81H) في المسجل DL.عادة يزود مشغل الجهاز من قبل الشركة المنتجة للقرص الصلب.

12 4 12 - كتابة/قراءة قطاع قرص صلب موسع

إن التوابع 0AH & 0BH شبيه بالتوابع 02H & 03H المخصصة لقراءة وكتابة القطاعات.مهما يكن فهناك اختلاف واحد فقط هو أنه بالإضافة إلى الـ512Bytes المخصصة للمعطيات من أجل كل قطاع ينم نقله فإنه يوجد أربعة بايتات إضافية مخصصة لخوارزمية ECC عند نهاية كل قطاع يتم نقلها أيضا.بما أن كل قطاع أصبح يتكون من 516Bytes بدلا من 512Bytes فإن العدد الأعظمي للقطاعات التي يمكن قراءتها أو كتابتها في نفس الوقت هو 127 قطاع ,بينما كانت التوابع 02H,03H تعالج 128 قطاع.يفحص التابع 10H فيما إذا كان

القرص الصلب الذي مُرر رقمه في المسجل DL هو جاهز لأن ينفذ أوامر.إذا كان علم الحمل يحوي القيمة 1 فذلك يشير إلى أن السواقة غير جاهزة,وعندها فإن المسجل AH سيحوي قيمة شفرة الخطأ.

12 4 13 -معايرة القرص الصلب

يستخدم التابع 0BH لمعايرة القرص الصلب.بعد استدعاء التابع ,يعيد هذا التابع حالة الخطأ متماشيا مع رقم القرص في المسجل DL.

12 4 14 -اختبار جهاز التحكم بالقرص الصلب

يستخدم التابع 14H لعملية الاختبار,إذا اجتاز جهاز التحكم هذا الاختبار فإن علم الحمل سيتضمن القيمة 0.تابع المقاطعة الأخير المخصص للقرص الصلب هو 15H الذي يكون متوفرا فقط على ATs وليس على XTs.تعيد هذه المقاطعة نوع القرص.يمرر رقم القرص (80 or 81) في المسجل DL.إذا لم يكن القرص متوفرا سيتم إرجاع القيمة 0 في المسجل AH.تشير القيمة 2 or1 إلى سواقة القرص المرنة.بينما تشير القيمة 3 لإلى سواقة القرص الصلب.في هذه الحالة تحتوي المسجلات CX and DX على عدد القطاعات لهذا القرص الصلب.يشكل هذا المسجلان 32bit, حيث يحوي المسجل CX البايت الأكثر أهمية بينما يحتوي المسجل DX البايت الأقل أهمية.

12 5 -ميزات سواقات الأقراص الصلبة

من أهم الميزات الحديثة لسواقات الأقراص الصلبة هو أنه أصبح أسرع و أرخص و أصغر حجماً فعلى سبيل المثال:

تم تقليل زمن الوصول في النماذج ذات الأداء الحديث من حوالي 30ms إلى 10ms
زيادة سعة التخزين الأعظمية فقد أصبحت سواقات الأقراص الصلبة ذات السعات أكثر من 1GigaByte كثير الانتشار
أصبحت أسعار سواقات الأقراص الصلبة منخفضة جداً

و سنناقش فيما يلي أهم أسباب تحسين أداء سواقات القرص الصلب.

12 5 1 - التوزيع و عامل عملية التوزيع

حالياً متحكمات سواقة القرص الصلب سريعة جداً بحيث أنها تستطيع مسار كامل حتى لو كانت المعطيات المطلوبة موجودة في قطاع واحد. و هكذا إذا طلبت المعطيات الموجودة في القطاع التالي فلا يقوم المتحكم بالقراءة من القطاع و لكن يتم أخذ المعطيات مع الذاكرة الداخلية للمتحكم و هذا يزيد من سرعة الوصول للمعطيات بشكل كبير. و تملك المتحكمات ST506 ذاكرة داخلية تتسع لقطاع واحد و لكن المتحكمات الحديثة من النوع (SCI ,ESDI ,and IDE) تملك ذاكرة داخلية تتسع لمسار كامل.

Interleaving in the first XT and AT hard drives from IBM			
AT		XT	
Physical sectors	Logical sectors	Physical sectors	Logical sectors
1	1	1	1
2	7	2	4
3	13	3	7
4	2	4	10
5	8	5	13
6	14	6	16
7	3	7	2
8	9	8	5
9	15	9	8
10	4	10	11
11	10	11	14
12	16	12	17
13	5	13	3
14	11	14	6
15	17	15	9
16	6	16	12
17	12	17	15

الشكل145

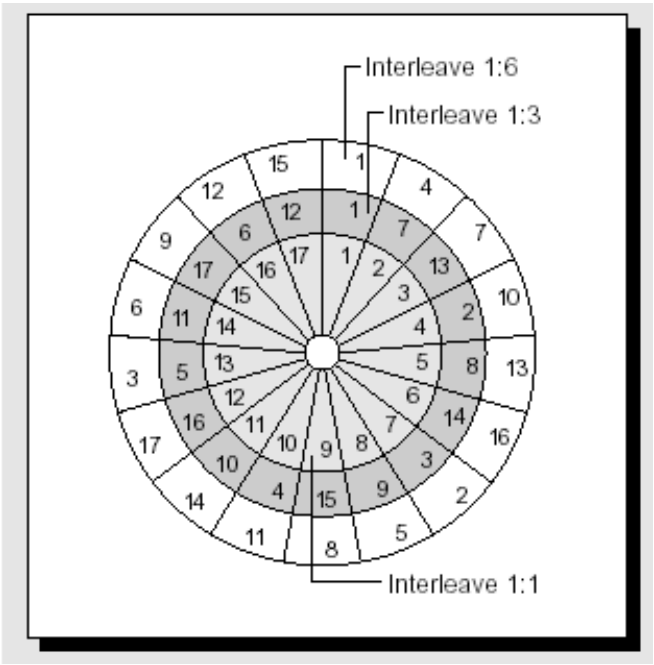
و لا يتم إعادة ملى الذاكرة ذات سعة القطاع إلا بعد نقل آخر بايت إلى CPU. وهذا يتطلب مزيداً من الوقت وهذا الوقت كافي ليمر القطاع التالي من تحت رؤوس القراءة و الكتابة و بالتالي لقراءة المعطيات من القطاع التالي سيتطلب هذا انتظار حوالي دورة كاملة للقرص الصلب و بالتالي إعادة تنفيذ هذه العملية سيودي إلى بطيء ملحوظ في زمن الوصول.

و بالتالي لتقليل هذا التأخير يتم استخدام ما يسمى عملية التوزيعInterleaving لنشر القطاعات في المسار. و بتوزيع القطاعات منطقياً يتاح وقت كافي للمتحكم لمعالجة و إرسال المعطيات من قطاع واحد قبل أن يمر القطاع ذو الترتيب المنطقي التالي تحت رأس القراءة, و بهذا يتم تجنب انتظار دورة كاملة قبل قراءة القطاع التالي. و يتم قياس التوزيع بعامل عملية التوزيع و قيمته هي عدد القطاعات و التي تم إزاحتها برقم القطاع المنطقي عن رقم القطاع الفيزيائي. في سواقة القرص الصلب لجهاز XT يكون عامل عملية التوزيع مساوياً لـ 1:6 أما في أجهزة AT فهو 1:3 و يمكن أن يقلل إلى 1:2 مما يزيد سرعة الوصول. و حالياً القيمة 1:1 لهذا العامل منتشرة جداً مما يعني أنه لا يوجد توزيع على الإطلاق كما في الجدول السابق حيث يوجد 17 قطاع , و بغض النظر عن أهمية التوزيع فإن أفضل عملية التوزيع تكون عندما لا توجد حاجة للتوزيع. و بذلك يمكن قراءة مسار كامل ضمن دورة واحدة للقرص فقط. أما بوجود توزيع فسيكون هناك حاجة لعدد من الدورات 3,2 و ذلك بحسب قيمة عامل عملية التوزيع.

12 5 2 -إسناد عملية التوزيع

يتم ذلك خلال عملية التهيئة ذات المستوى المنخفض لسواقة القرص الصلب

Different interleave factors and their influence on the formatting of a track

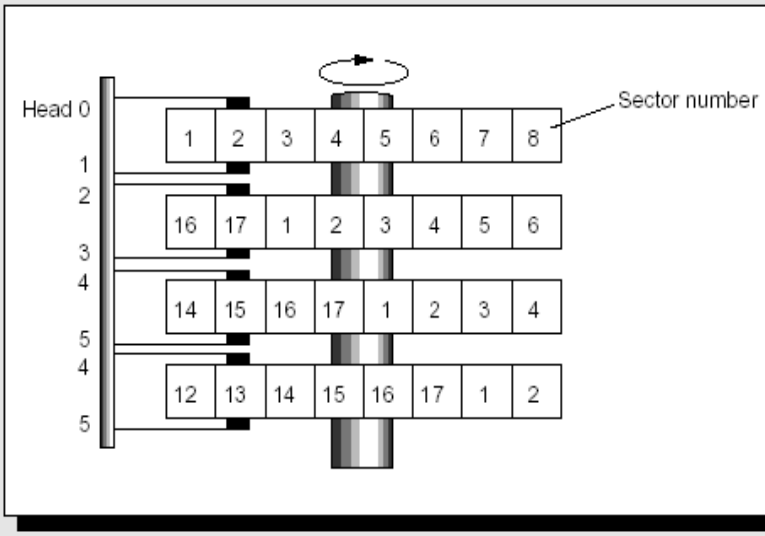


الشكل 146

وبما أن الرقم المنطقي للقطاع يتم تحديده باستخدام برامج التهيئة ذات المستوى المنخفض فمن الممكن إزاحة القطاعات. و معظم البرامج المخدمة لسواقة القرص الصلب ستعطيك عامل عملية التوزيع المناسب. إذا كانت القيمة سيئة سيؤدي هذا إلى بطء شديد في الوصول إلى الملفات الموجودة على سواقة القرص الصلب.

12 5 3 -انحراف الأسطوانة و المسار

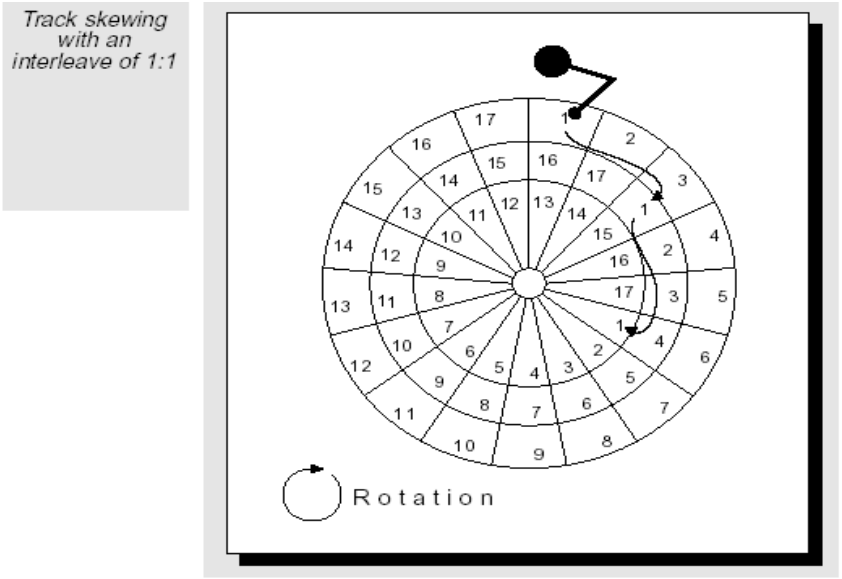
Cylinder skewing



الشكل147

اختيار عامل عملية التوزيع المناسب مهم جداً لتحسين أداء سواقة القرص الصلب. و بعد قراءة المعطيات من مسار معين يتم عادة قراءة المعطيات من على نفس الأسطوانة و لكن من رأس قراءة آخر و بينما يقوم المتحكم بالتبديل بين رؤوس القراءة يستمر القرص الصلب بالدوران و بعد قراءة آخر قطاع من المسار فإن القطاع الأول من الرأس التالي سيكون قد مر من تحت رأس القراءة و لذلك يجب الانتظار حوالي دورة كاملة.

و لمنع هذا من الحدوث يتم استخدام انحراف الاسطوانة حيث يتم إزاحة كل القطاعات على المسار التالي, و بالتالي بعد تبديل رأس القراءة يتم قراءة القطاع الأول بدون تأخير دوراني. ويتم تحديد انحراف الاسطوانة خلال عملية التهيئة ذات المستوى المنخفض.و بالإضافة لعملية انحراف الاسطوانة توجد عملية انحراف المسار و هي مشابهة لعملية انحراف الاسطوانة و لكنها تقوم بالاهتمام بالوقت اللازم لسواقة القرص الصلب لنقل ذراع القراءة إلى المسار التالي



الشكل 148

12 5 4 -تصحيح الخطأ

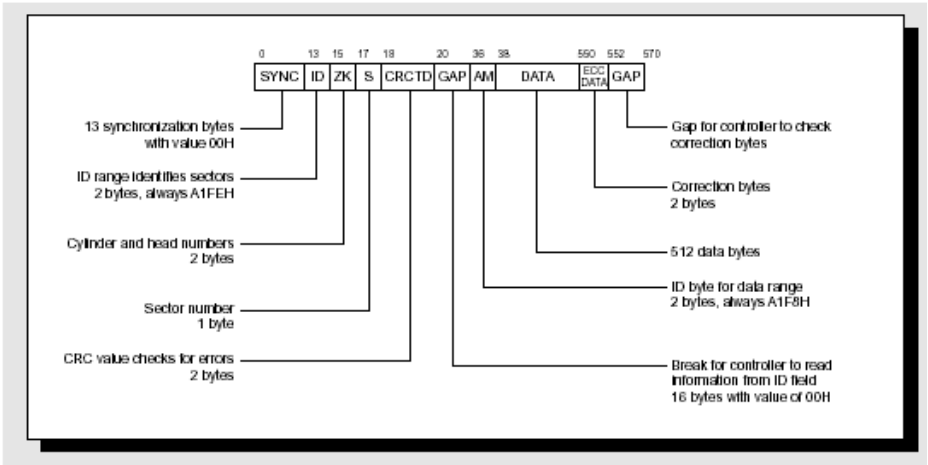
تعتبر الموثوقية من أهم مؤشرات عملية التخزين. إن حدوث التلوث في المادة المغناطيسية شيء لا يمكن تجنبه و يمكن أن يسبب عدم استخدام لبعض الأجزاء من سواقة القرص الصلب.في ما مضى كانت تطبع لائحة بالقطاعات المصابة على حافظة (غلاف) سواقة القرص الصلب كما تم تحديدها من قبل المصنعين و خلال عملية التهيئة ذات المستوى المنخفض يقوم المستخدم بإدخال هذه الأرقام ليتعرف عليها نظام التشغيل و يتم تعليمها على أنها مصابة في جدول توزيع الملفات FAT و يتجنب تخزين المعطيات فيها.

أما النماذج الحديثة لـ IDE ليس لها لائحة قطاعات مصابة و يتم تخزينها على مسار معين منفصل و تتعرف عليها خلال عملية التهيئة ذات المستوى المنخفض. و العديد من السواقات تستطيع تمييز قطاعات مصابة عن طريق عمليات القراءة و الكتابة و هذا هو هدف ECC (شفرة تحديد الخطأ) و التي يتم توليدها أوتوماتيكياً و تخزينها على كل قطاع و هذه الميزة موجودة في سواقة القرص الصلب IDE.

12 5 5 -مكونات سواقة القرص الصلب الأخرى

عندما يتم تهيئة سواقة القرص الصلب فإنه يتم تخزين مجموعة معطيات كاملة على سواقة القرص الصلب و نمط هذه المعطيات يختلف بحسب المتحكم (مثل رقم الأسطوانة, رقم ECC ,....) حيث يبدأ كل قطاع بسلسلة من 13 بايت متزامنة مع القيمة 00H (حقل البدء) و يلي هذه القيمة حقل رقم التعريف ID و الذي يعرف القطاع. بالإضافة إلى أرقام الأسطوانة, الرأس, القطاع(C,H,S) و تبدأ هذه المعطيات ببايت ID مخصص و ينتهي ببايت شفرة الخطأ تستخدم للتأكد من صحة المعطيات (حقل CRCID)، و من ثم منطقة فارغة gap تعطي المتحكم فرصة قبل بدء بحقل المعطيات و يستخدم المتحكم هذه الفرصة لقراءة قيمة ID و التأكد هل هذا هو القطاع المطلوب. و من ثم يبدأ 512 بايت من المعطيات (حقل Data) و من ثم هناك بايتين يستخدمهما المتحكم للتأكد من صحة المعطيات(حقل ECC-Data) و من ثم منطقة فارغة ثانية gap مملوءة بالقيمة 4EH تعطي المتحكم فرصة للتأكد من صحة المعطيات و بذلك يكون الطول الفعلي الكلي لكل قطاع هي 570 بايت.

Sector format created by an ST506-type controller



الشكل 149

مسارات إضافية: و هناك مناطق محجوزة أخرى فهناك مسارات مؤازرة و تستخدمها الدارة الالكترونية لسواقة القرص الصلب لتزامن نفسها مع ساعة المعطيات. و هناك بعض المسارات حتى BIOS لا تراها و هي محجوزة كبديل عن القطاعات المصابة. و أخيراً هناك منطقة الاستراحة (Park area) تستخدم عندما يطفئ جهاز الكمبيوتر حيث تستقر رؤوس القراءة و الكتابة على سطح منطقة الاستراحة. لأنه إذا استقرت رؤوس القراءة و الكتابة على مسار يحتوي معطيات فإنه قد يضر بهذه المعطيات, و حيث أنه لا يوجد معطيات في منطقة الاستراحة.

12 5 6 -زمن الوصول

يقاس أداء سواقة القرص الصلب بسرعة الوصول. و يدعي عادة المصنعون أن سواقة القرص الصلب التي يصنعونها ذات سرعات وصول حوالي 10 أو 30 ميلي ثانية. و تعبر هذه القيمة عن معدل زمن الوصول إلى ملفين مختلفين.و يستخدم أيضاً المصنعون زمن الوصول من مسار إلى مسار و زمن الوصول الأعظمي ليعبران عن سرعة سواقة القرص الصلب. حيث أن زمن الوصول من مسار إلى مسار هو الزمن اللازم لتحريك رأس القراءة و الكتابة من

الأسطوانة إلى أخرى. و زمن الوصول الأعظمي هو الزمن اللازم لتحريك ذراع القراءة و الكتابة من الأسطوانة الأولى إلى الأخيرة للسواقة.

كما أن هذه العوامل لها تأثير مهم على كيفية الوصول إلى سواقة القرص الصلب من مستوى نظام التشغيل و هناك عوامل أخرى فمثلاً: قص بمواصفات سريعة جداً تكون فيه هذه المواصفات عديمة الفائدة ما لم يستطع المتحكم مجاراته. و قد تكون سواقة القرص الصلب مجزئ جداً بحيث أن ذراع القراءة و الكتابة تقفز بشكل دائم للأمام و للخلف بين الأسطوانات المختلفة للوصول إلى المعطيات.لذلك يجب أن يقاس الأداء اعتماداً على الزمن الذي تستغرقه طلبات القراءة و الكتابة في المستويات المختلفة للتطبيق. أي على مستوى نظام التشغيل و مشغلات السواقات و مستوى BIOS و من مستوى سواقة القرص الصلب و متحكماته. و هناك تطبيقات تقيس أداء سواقة القرص الصلب.

12 6 -تجزئة سواقة القرص الصلب Hard Disk Partition

عند تنفيذ تعليمة FDISK من نظام التشغيل DOS فإنها تقوم بتجزئة سواقة القرص الصلب إلى كتل منفصلة منطقياً أو عندما تريد تحميل أكثر من نظام تشغيل على قص واحد. عملية التهيئة لتحضير سواقة القرص الصلب ليتم استخدامها من قبل نظام تشغيل يجب تنفيذ ثلاث تنفيذ مهام. أولاً يجب عليك تنفيذ عملية تهيئة منخفضة المستوى. و عندما تقوم بهذا فأنك تقوم تنظيم سواقة القرص الصلب إلى سواقات و مسارات و قطاعات عن طريق كتابة علامات العنوان المناسب على سطح السواقة. يستخدم المتحكم علامات العنوان فيما بعد لتعيين القطاعات المحددة.

12 6 1 -عملية تجزئة سواقة القرص الصلب

هناك سببان رئيسيان للقيام بعملية التجزئة. أولاً هذا يمكنك من تحميل أكثر من نظام تشغيل على سواقة القرص الصلب. فتجزئة سواقة القرص الصلب إلى عدة مناطق منفصلة يتيح لكل نظام تشغيل أن يدير جزءه الخاص من سواقة القرص الصلب بدون حدوث اضطرابات عن اختلاف هيكلية أنظمة الملفات.

و السبب الآخر للتجزئة هو حتى يصبح من الممكن استخدام سعات إضافية للسواقات ذات السعات الكبيرة. تمتلك حواسب XT قص بسعة 10Meg ثم ظهرت سواقات قرص صلب بسعات 40,80 Meg و كانت نسخ Dos القديمة تستطيع التعامل مع سعات 32 Meg فقط و لكن تم التخلص من هذه المشكلة في نسخة Dos 3.3 حيث أصبحت سعة 32 Meg مرتبطة بجزء واحد. حيث قدمت هذه النسخة من DOS إمكانية تجزئة سواقة القرص الصلب لجزء رئيسي (Primary Portion) و جزء ممتد (Extended Portion) يمكن تجزئته لـ 23 جهاز منطقي سعة كل منها الأعظمية 32 Meg مما يتيح سعة كلية أعظمية 768 Meg لسواقة القرص الصلب. و نسخة Dos 4.0 كانت تدعم أقراص صلبة بسعة 2 Gigabyte و لكن المستخدمين استمروا بعملية التجزئة لأنهم كانوا يفضلون التعامل مع عدة أقراص منطقية على التعامل مع قرص واحد كبير السعة.

12 6 2 ـ قطاع التجزئة

و هو عبارة عن البنية التي تستخدمها جميع نسخ DOS لتعريف تجزيئات سواقة قرصها الصلب. و عندما تقوم بتنفيذ برنامج FDISK لأول مرة فإنه يقوم بإنشاء قطاع التجزئة و ذلك على القطاع الأول لسواقة القرص الصلب المعرف بالأرقام (Cylinder0,Track0,Sector1) و يقوم BIOS في عند بدء التشغيل بتحميل قطاع التجزئة بدلاً من قطاع الإقلاع لـ DOS و بعد أن ينتهي إتمام تشغيل نظام الحاسب يتم تحميل قطاع التجزئة إلى الذاكرة و ذلك بدءاً من العنوان 0000:7C00 و هذا إذا لم يكن هناك أقراص مرنة في السواقة :A.و إذا وجد BIOS القيمتين 55H وتليها AAH في البايتين الأخيرين من قطاع التجزئة المؤلف من 512 بايت فإنه يعتبر هذا القطاع قابلاً للتنفيذ و يبدأ بتنفيذ البرنامج بدءاً من البايت الأول من القطاع نفسه و إلا يقوم BIOS بإظهار رسالة خطأ و من ثم يبدأ تنفيذ ROM-BASIC أو يدخل في حلقة لا نهائية و ذلك بحسب نسخة BIOS و مصنعيها.و يقوم هذا البرنامج بتنظيم و تشغيل الجزء الفعال من نظام التشغيل و يتطلب القيام بذلك تحميل قطاع الإقلاع لنظام التشغيل و تمرير التحكم للبرنامج الموجود ضمن هذا القطاع و بما أن البرنامج الجديد يجب تحميله إلى الذاكرة أيضاً على العنوان 0000:7C00 فإن برنامج قطاع التجزئة يتم نقله للعنوان 0000:0600 أولاً لإفساح المجال لقطاع الإقلاع بالعمل.

Structure of the partition sector of a hard drive

Address	Contents	Type
+000h	Partition code	Code
+1BEh	1.Entry in the partition table	16 BYTE
+1CEh	2.Entry in the partition table	16 BYTE
+1DEh	3.Entry in the partition table	16 BYTE
+1EEh	4.Entry in the partition table	16 BYTE
+1FEh	IDcode(AA55h), which identifies the partition sector as such	2 BYTE

Length: 1EH (30) bytes

الشكل150

12 6 3 ـ جدول التجزئة

البرنامج المكتوب في قطاع التجزئة يجب أن يكون قادراً على إيجاد للجزء الفعال و للقيام بذلك فإنه يستخدم جدول التجزئة و يتم تحميل هذا الجدول في الإزاحة 1BEH من قطاع التجزئة.

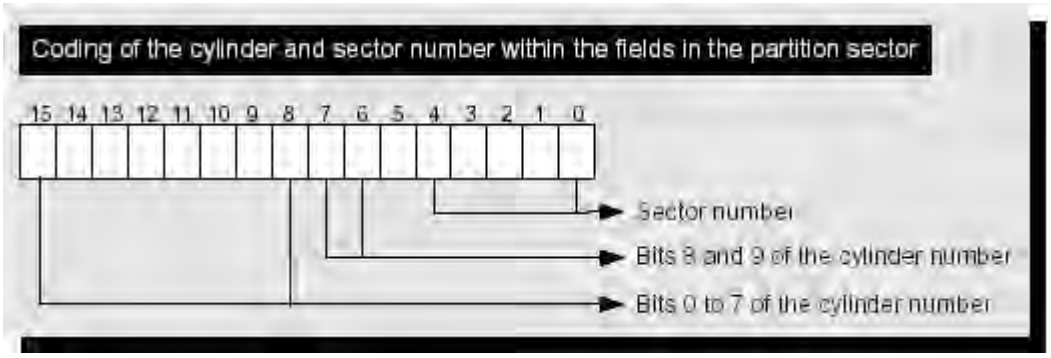
Structure of an entry in the partition table		
Address	Contents	Type
+00h	Partition status 00h = inactive 80h = Boot-Partition	1 BYTE
+01h	Read/write head, with which the partition begins	1 BYTE
+02h	Sector and cylinder, with which the partition begins	1 WORD
+04h	Partition type * 00h = Entry not allocated 01h = DOS with 12-Bit-FAT (primary Part.) 02h = XENIX 03h = XENIX 04h = DOS with 16-Bit-FAT (primary Part.) 05h = extended DOS-Partition (DOS 3.3) 06h = DOS-4.0 partition with more than 32 Meg DBh = Concurrent DOS	1 BYTE
+05h	Read/write head, with which the partition ends	1 BYTE
+06h	Sector and cylinder, with which the partition ends	1 WORD
+08h	Removal of first sector of the partition (Boot-sector) of partition sector in sectors	1 DWORD
+0Ch	Number of sectors in this partition	1 DWORD
Length: 10h (16 Bytes)		
* Other codes possible combined with other operating systems or special driver software.		

الشكل151

كل مدخل Entry في جدول التجزئة مكون من 16 بايت و هذا الجدول موجود في نهاية قطاع التجزئة متيحاً المجال لـ 4 مداخل لذلك عدد التجزئات محدود بـ 4 أجزاء. و لإنشاء أكثر من 4 أجزاء يلجأ بعض مصنعي سواقات الأقراص الصلبة إلى استخدام برنامج إعداد خاص.في بعض الأحيان يتم تعديل شفرة قطاع التجزئة مما يتيح لك الإقلاع إلى أي من أنظمة التشغيل التي لديك و هذا يجعل الاختيار أسهل بين أنظمة التشغيل عندما يبدأ الحاسب بالعمل. و يتألف جدول التجزئة للمدخل الواحد كما في الجدول السابق.

12 4 6 4 -بدء تشغيل قطاع الإقلاع

الحقل الأول من كل مدخل جدول تجزئة يشير إذا ما كان الجزء فعالاً أم لا. القيمة 00H تعني أن الجزء غير فعال و القيمة 80H تعني أن الجزء فعال و يجب الإقلاع منه. و إذا وجد برنامج قطاع التجزئة أكثر من جزء واحد فعال أو أنه لا يوجد أي منها فعال فإنه لا يتابع عملية الإقلاع و يظهر رسالة خطأ و يدخل في حلقة لا نهائية لا يمكن الخروج منها إلا بإعادة الإقلاع، عندما يحدد برنامج قطاع التجزئة جزء فعال واحد فإنه يستخدم الحقلين التاليين لتحديد مكان هذا الجزء على القرص الصلب. و يتم التعبير عن رقم الأسطوانة و القطاع تماماً كما في المقاطعة رقم 13 و متضمناً البت السادس و السابع من رقم القطاع.و الذي يمثل البت الثامن و التاسع من رقم الأسطوانة (أي هذين البايتين لهما الشكل الذي يمكننا مباشرة استدعاء مقاطعة BIOS رقم 13 بتحديد رقم الأسطوانة و رقم القطاع لها).



الشكل 152

في هذا الوقت تكون مقاطعة BIOS رقم 13 و توابعها هي الطريقة الوحيدة للوصول لسواقة القرص الصلب لأنه من الواضح أن توابع DOS غير متاحة في هذه اللحظة.

12 7 -المراجع

28 جدول

PC Intern (Chapter 14)	written by: Michael Tescher and Bruno Jennrich DATA BECKER Edition 1996
Information Technology AT Attachment with Packet Interface Extension (ATA/ATAPI4)	written by: T13, a Technical Committee of Accredited Standards Committee, NCTTS Revision 18 19 August 1998

الباب . التعريب

13 -التعريب

13 1 -أساسيات عملية التعريب ومشاكلها

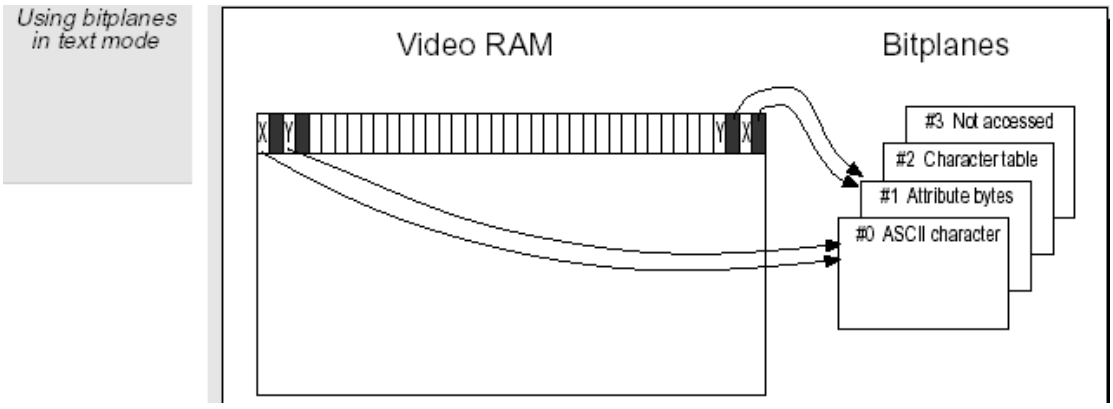
إن فكرة تحميل الأحرف العربية هي أن نحمل الأحرف العربية مكان الأحرف الموسعة لـ ASCII (128 - 255) واستعمال جدول حروف عربية في مكان الحروف ASCII الموسعة الموجودة، والمشاكل التي ستواجهنا:

تبديل الحروف 128 الأخيرة من حروف ASCII بالحروف العربية وتحميلها بالذاكرة.
صناعة خط عربي يناسب الVGA .
الكتابة من اليمين إلى اليسار (RightToLeft بدلا من LeftToRight).
مشكلة التوصيل في الأحرف العربية فمعظم الأحرف لها أربعة حالات (باستثناء أحرف القطع) فحرف الخاء له أربعة أشكال - خ ـخـ ـخ – فعند الضغط على لوحة المفاتيح على حرف الخاء(المقابل لحرف O الإنكليزي) يجب تحديد أي نوع من الأحرف التي سيتم تحديدها بناء على الحرف الذي قبله والحرف الذي بعده لتحديد شفرة المسح (سيتم شرحها لاحقاً).
العمل في النمط المحمي (Pmode) والنمط الحقيقي (Real mode).
العمل في النمط النصي (Text Mode) والنمط الصوري (Graphics Mode).

13 2 -علاقة ذاكرة العرض بالتعريب

لقد قمنا بشرح قسم الذاكرة بشكل سابق ولكن نشرح هنا ما يهمنا من أجل عملية التعريب:

تقسم الذاكرة الفيديوية إلى أربع أقسام (مناطق) كبيرة تدعى bitplanes, وهذه المناطق تستخدم لأهداف وغايات مختلفة في النمط النصي والنمط البياني, معظم كروت الشاشة لديها 256 K يتم حجز 64 K لكل bitplane.

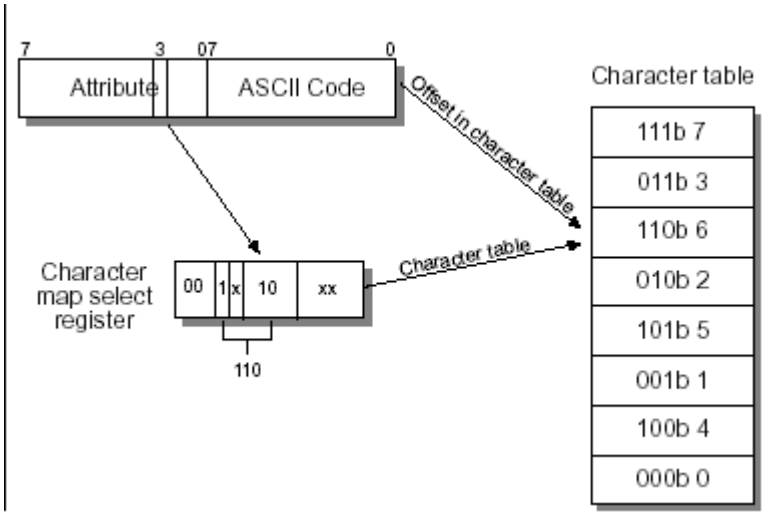


الشكل 153

Bitplane 0: تحتوي على رمز المحرف ASCII
Bitplane 1: مواصفات الحرف.

وكما ذكرنا فإن الحرف له بايتان, البايت الأول يحتوي على ASCII والثاني على مواصفات الحرف (لونه ولون الخلفية له), في النمط النصي يتم تقسيم الذاكرة إلى نموذج زوجي/ فردي أي أنه يبدأ عنوان ذاكرة B800، وبالتالي مع إزاحة زوجية لهذا العنوان نصل مباشرة لعناوين Bitplane0ومع إزاحة فردية نصل لعناوينBitplane1 ، وفي EGA , VGA عادة تستخدم Bitplane2 من أجل جدول الأحرف ولتحميل جداول أحرف الـ ASCII التي يتم تحميلها من ذاكرة ROM إلى هنا للتعامل معها.

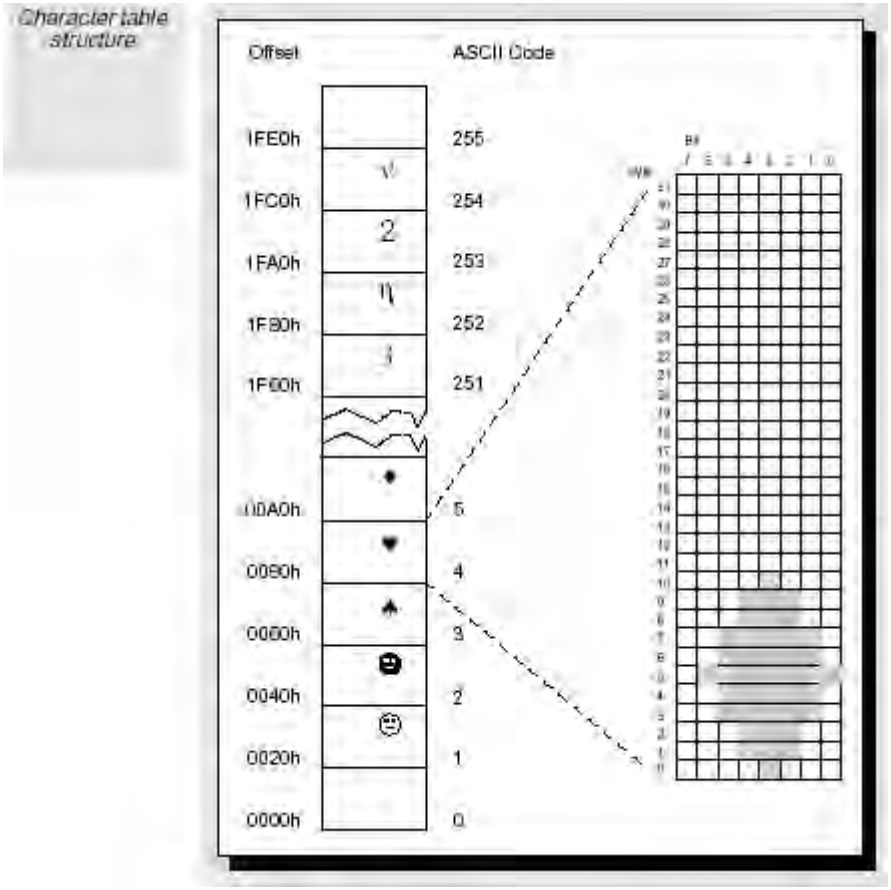
كل Bitplaneتأخذ 64 KB من الرام, حيث كل جدول أحرف يأخذ 8 K ولذلك يمكن أن نستعمل 8 جداول أحرف مختلفة K8 8X = K64 ولا اختيار جدول معين يتم ذلك عن طريق بايت المواصفات ومسجل Character map register وقد تم شرحها سابقا.



الشكل154

كل جدول أحرف لديه K8 حيث يتم حجز 32 بايت لكل حرف وبالتالي يمكن للجدول أن يتسع إلى 256 حرف وهكذا نحصل على 32 * 256 = 8192 = 8 KB، حيث تمثل 32 بكسل ارتفاع الحرف الأعظم ويمكن أن لا نستخدم النصف العلوي من الحرف ويبقى فارغ وللوصول إلى حرف ما ضمن الجدول أو إلى سطر ما ضمن الجدول.

عنوان بداية الجدول * (الحرف ASCII)(Index) + السطر المطلوب



الشكل155

وقد قامت الـ BOIS بتغليف العمل السابق من خلال نصف دزينة من التوابع المتوفرة من خلال الخدمة 11 للمقاطعة 10 الفيديوية INT 10.

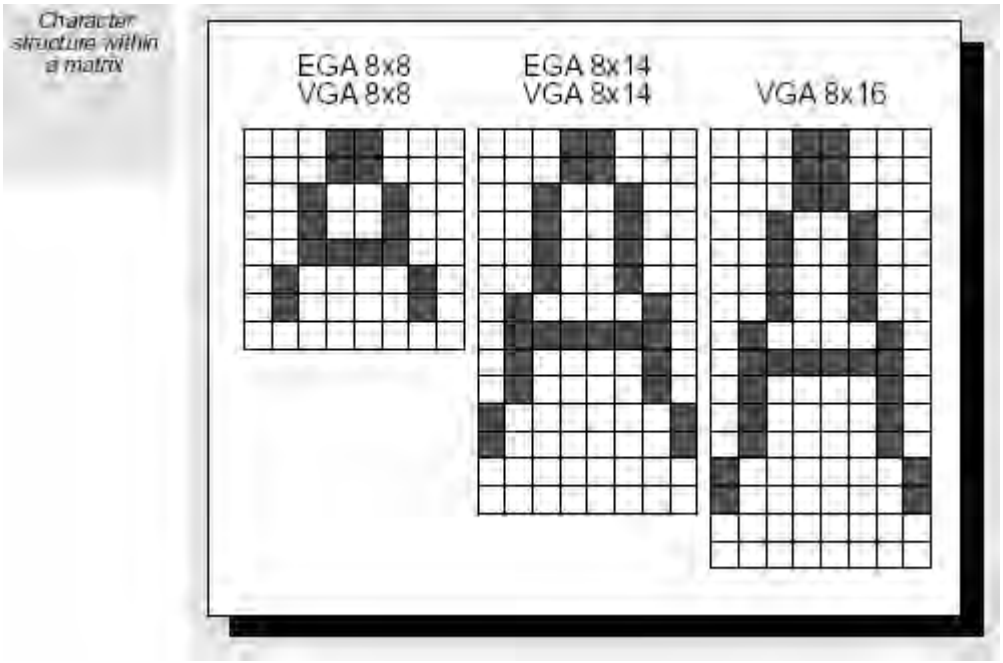
13 3 -الخطوط في بطاقات العرض

قبل الدخول في تفاصيل الخدمة 11 لا بد من إلقاء النظرة عن الخطوط في النمط النصي.إن بنية خطوط الفيديو تختلف عن الخطوط المستخدمة للطباعة, وينشأ هذا الاختلاف بسبب حجم الأحرف (المصفوفة التي تمثل عرض الأحرف)، حيث يستخدم أل EGAعادة مصفوفة من 14 X 8بكسل لخط الروم الأول ويستخدم مصفوفة أصغر لحجم الروم الثاني 8 X 8 حيث حجم بكسل يبقى ثابتا نفسه لا يتغير ولكن الذي يتغير عدد البكسلات في الارتفاع فقط (حيث البكسل في النمط النصي يكون بت واحد فقط)، وبالخلاصة يوجد ثلاثة خطوط قياسية لل VGA:

نوع الخط	الارتفاع X العرض
صغير	8 X 8
متوسط	8 X 14
كبير	8 X 16

وخطين قياسيين لل EGA

نوع الخط	الارتفاع X العرض
صغير	8 X 8
متوسط	8 X 14



الشكل156

13 4 ـدقة الشاشة

13 4 1 ـعلاقة ارتفاع الشاشة مع ارتفاع الخط

إن الخطوط الصغيرة تشغل مساحة أصغر على الشاشة من حيث الأرتفاع مثلا بالنسبة لل EGA الارتفاع هو 350 بكسل، والخط الصغير ارتفاع الحرف فيه 8 بكسل:

عدد الأحرف بالارتفاع = عدد الأسطر = 350/8 = 43.75
أي حوالي 43 وبالتالي يكون النمط 80 X 43.
الخط المتوسط لل VGA:
14/350 = 25 سطر => 80 X 25
بالنسبة لل VGA (حيث يحتوي 400 بكسل بالارتفاع)
الخط الكبير 400/16 = 25 سطر
الخط المتوسط 400/14 = 28 سطر
الخط الكبير 400/16= 25 سطر

13 4 2 ـعلاقة عرض الشاشة مع ارتفاع الخط

إن العرض يبقى ثابت لأنوع الخطوط الثلاثة:

EGA: عرض الحرف 8 بكسل في EGA , والعرض الأفقي للشاشة في EGA هو 640 بكسل وبالتالي:

عدد الأحرف الأفقية = 640/8 = 80 حرف.

VGA: يملك كرت VGA 720 بكسل في العرض الأفقي للشاشة ولكن الحرف الواحد عرضه 9 بكسل وليس 8 بكسل (يستخدم البت الأخير لأغراض خاصة وللمسافة بين الكلمات..) وبالتالي يكون عندنا 80 حرف بالسطر.

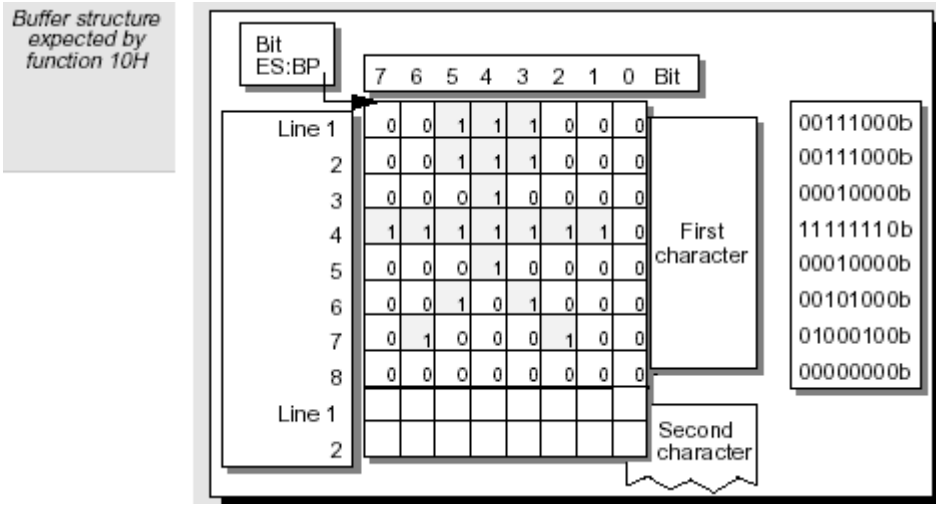
عدد الأحرف الأفقية = 720/9 = 80 حرف.

الخط	نوع كرت الشاشة	الارتفاع	العرض	الدقة
صغير	EGA	350/8 = 43	640/8 = 80	80 X 43
متوسط	EGA	350/14 = 25	640/8 = 80	80 X 25
صغير	VGA	400/8 = 50	720/9 = 80	80 X 50
متوسط	VGA	400/14 = 28	720/9 = 80	80 X 28
كبير	VGA	400/16 = 25	720/9 = 80	80 X 25

13 5 -الخدمة 11 INT 10 (استحداث محارف جديدة) في النمط الحقيقي

فالخدمة 11 تفيد في الأنظمة EGA VGA MCGA في تغيير مجموعة المحارف المستخدمة في النمط النصي لاستحداث محارف جديدة , كما يمكن استخدامها لقراءة معلومات عن المحارف الحالية وتسمى هذه الخدمة Character Generator

يستعيد البيوس نموذج الأحرف من بفر(Buffer) , وهذا البفر(Buffer) يجب أن يخلق ويهيئ من قبل توابع أخرى , وعنوان هذا البفر يمثل مؤشر FAR للمسجلين ES: Bp حيث ومنه نجد أن يمكن الوصول لجدول الحروف من خلال المؤشر السابق



الشكل157

13 5 1 -تحميل الأحرف في النمط Real mode

إذا عندنا مكان بالذاكرة وهو Bitplane2 مخصص لوضع الأحرف التي نريد استحداثها وبالتالي عندما تكون قيمة البت 1 عندها يضيء مكانه وعندما يكون صفر لا يضيء ولاستحداث حرف العربي الشين (ش) نقوم بما يلي:

```
HARF_SHIN
      db 00000000b
      db 00000000b
      db 00001100b
      db 00001100b
      db 00010010b
      db 00000000b
      db 00000000b
      db 10011111b
      db 10010000b
      db 10010000b
      db 10010000b
      db 01100000b
      db 00000000b
      db 00000000b
      db 00000000b
```

يأخذ هذا الحرف 16 بايت وهو من نمط الخط الكبير يتم تحميل هذا الحرف بالجدول ثم الحرف الذي يليه وهكذا حتى يتم تحميل كافة الأحرف التي نريد تحميلها.نضع في المسجل AH رقم الخدمة و في المسجل AL رقم الخدمة الفرعية فالخدمة 11 لها العديد من الوظائف الفرعية و في المسجل BH عدد البايتات في كل حرف وفي المسجل BL الجدول المراد تحميله في الرام (الأحرف العربية)

```
AH = 11h
AL = 00  تحميل محارف المستخدم
BH = 0x10// عدد البايتات في كل حرف 16 بايت
BL =  الجدوال المراد تحميله في الرام (0-7)
CX =  عدد الأحرف في كل جدول (حوالي 255)
DX =  رمز الاسكي لأول حرف في الجدول
ES:BP =  مؤشر يؤشر إلى بداية الجدول الذي سنقوم بتحميله في الرام
```

جدول 29

AH	Sub functions of function 11 from INT 10 Character generator
0	Load Custom Text Character set
1	Load 8*14 Text Character Set
2	Load 8*8 Text Character Set
3	Select Loaded Text Character Set
4	Load 8*16 Text Character Set
10	Load Custom Text Character Set and adjust Height
11	Load 8*14 Text Character Set and adjust eight
12	Load 8*8 Text Character Set and adjust Height
14	Load 8*16 Text Character Set and adjust Height

20	Load 8*8 Graphic Character Set ES:BP->table of bit patterns (hi-bit only)
21	Load Custom Graphic Character Set BL= 0: DL=no. of rows = 1: 14 rows = 2: 25 rows = 3: 43 rows ES:BP->table of bit patterns
22	Load 8*14 Graphic Character Set BL= 0: DL=no. of rows = 1: 14 rows = 2: 25 rows = 3: 43 rows
23	Load 8*8 Graphic Character Set BL= 0: DL=no. of rows = 1: 14 rows = 2: 25 rows = 3: 43 rows
24	Load 8*16 Graphic Character Set BL= 0: DL=no. of rows = 1: 14 rows = 2: 25 rows = 3: 43 rows

30	Get Character Set Information ونحصل من هذه الوظيفة على معلومات الجدول المحمل وفقاً لقيم BH, حيث ياخذ القيمة من 0 إلى7 (أي أول ثلاث بتات) ولهذه الوظيفة ES:BP الفرعية وعلى سبيل المثال عند وضع القيمة 2 يتم وضع المؤشر 14 وبالتالي نستطيع X ذات الخط ROM8على الجدول المحمل بالروم INT 1F الحصول على معلوماته كاملةً , ولهذه الخدمة علاقة بالمقاطعة 43 والمقاطعة INT. BH = information desired: = 0 ~INT 1F~ pointer = 1 INT 43h pointer = 2 ROM 8x14 pointer = 3 ROM 8x8 double dot pointer (base) = 4 ROM 8x8 double dot pointer (top) = 5 ROM 9x14 alpha alternate pointer = 6 ROM 8x16 character table pointer = 7 ROM 9x16 alternate character table pointer on return: CX = bytes per character (points)(8/14/16) DL = rows (less 1) ES:BP = pointer to table

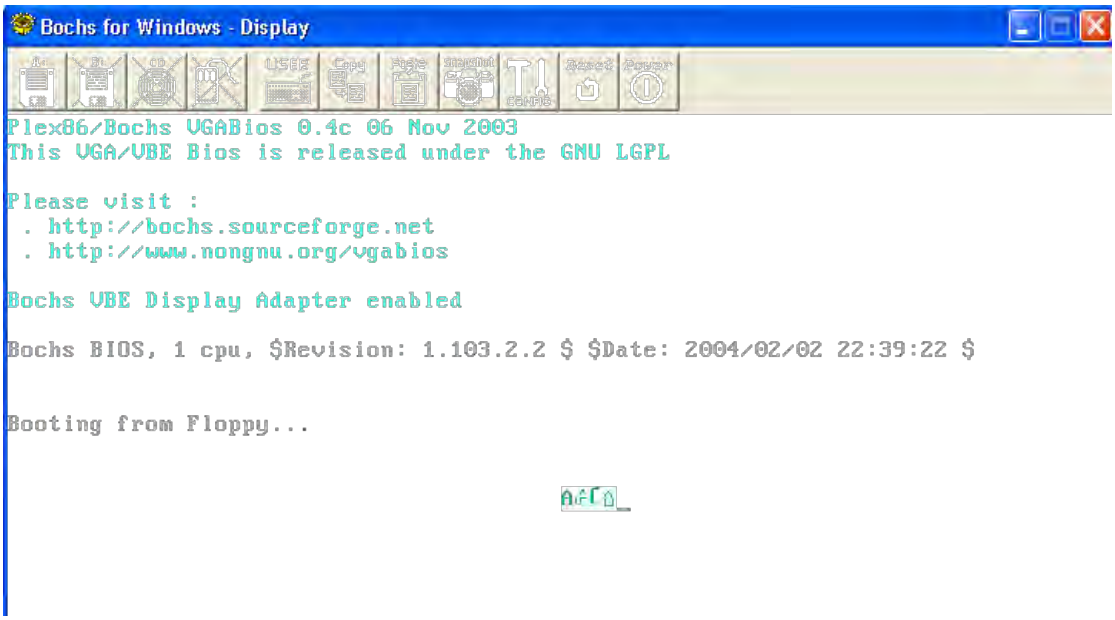
الآن الذي يهمنا من هذه الخدمات الفرعية الخدمة 0 وهي التي سوف نستخدمها لتحميل الأحرف العربية التي نريد تحميلها في شفرة ASCII الموسعة (128-255).

```
; SETTING GLYPH KHA IN THE VGA FONT TABLE

  MOV   BP,HARF_KHA  ; [ES:BP] عنوان بداية الجدول
  MOV   AH,0X11      ; 11 الوظيفة
  MOV   AL,0X00      ; 0 الوظيفة الفرعية
  MOV   BH,0X10      ; عدد البايتات في كل حرف 16 بايت
  MOV   BL,0X00      ;block 0 رقم الجدول المراد التحميل فيه ضمن
 MOV   CX    ,28     ; عدد الأحرف المراد تحميلها وهي 28 حرف ولكني لم أضع هنا غير حرف واحد فقط فيجب
                        وضع القيمة1 هنا
 MOV   DX,128        ; أول رقم اسكي يتم تحميله بالجدول أي أول حرف سيأخذ هذا الاسكي والحرف الذي
.                       يليه يأخذ الاسكي التالي وهكذا...
  INT   10H         ; استدعاء مقاطعة الفيديوية العاشرة
;           المكان المراد الإظهار فيه
MOV    DL, 80     ; العمود الحالي
MOV    DH, 10     ; السطر الحالي.
MOV    SI,NEW_LINE
INT   10H

NEXT_ROW:
  MOV    DL,80
  INC    DH
```

هذا الكود لإدخال حرف عادي وإظهاره أي ليس الأحرف التي فوق 128

```
MOV     BH, 0    ; page.
MOV     BP, msg  ; offset.
MOV     BL, 0F3h ; default attribute.
MOV     CX, 4    ; char number.
MOV     DL, 40   ; col.
MOV     DH, 15   ; row.
MOV     AH, 13h  ; function.
MOV     AL, 1    ; sub-function.
INT     10h

K_LOOP:
  MOV     AH,10H
  INT     16H
  MOV     AH, 02H
  INT     10H
  MOV     CX,1
  MOV     AH, 09H
  INT     10H
  DEC     DL
  CMP     DL,0 ; 255
  JE      NEXT_ROW

  MOV   AH,0EH
  MOV   BX,07H
  INT   10H
  JMP K_LOOP

NEW_LINE   DB 13,10,65,100,128,67
 msg db 65,128,131,129 ;  65 هنا يتم استدعاء الأحرف التي فوق 128وحرف  65=A 128=shin
131=miem                ;                                  129=thaa
HARF_Shin
       db 00000000b
       db 00000000b
       db 00001100b
       db 00001100b
       db 00010010b
       db 00000000b
       db 00000000b
       db 10011111b
       db 10010000b
       db 10010000b
       db 10010000b
       db 01100000b
       db 00000000b
       db 00000000b
       db 00000000b
```

في هذا البرنامج يتم تحميل الأحرف العربية فوق 128 ويتم إظهار الأحرف العربية والانكليزية معا....ولكن تبقى مشكلة إدخال الأحرف فوق 128 والوصول إليها والتي سنتكلم عليها لاحقا، أما بالنسبة لتشغيل البرنامج سنتكلم عنه بالملحق وسنتكلم عن المحاكي Bochs وpartcopy وnasmw.



الشكل158

وبهذه الطريقة نكون قد حملنا الأحرف العربية في النمط النصي وضمن الـ Real Mode.

أما في حالة النمط المحمي Pmode فلا يجب أن نستخدم مقاطعة البيوس BOIS المقاطعة العاشرة Int 10 فلذلك يتوجب علينا تغليف المقاطعة السابقة كلها وأن نؤمن وظائفها كاملة وذلك بالوصول المباشر إلى الذاكرة ولذلك يتوجب علينا التعامل مباشرة مع الذاكرة الفيديوية ومع مسجلات كرت الشاشة المختلفة التي تم شرحها سابقاً.

13 6 -تحميل الأحرف المستحدثة في النمط المحمي Pmode

سنتكلم عن تحميل الخط في النمط النصي(Text mode) ومن ثم النمط الصوري(graphics mode). يجب أن يتم الوصول مباشرة إلى bitplane2 يجب تغير طريقة عنونة الذاكرة ويتم ذلك من خلال المسجل Map Mask Register حيث يكون له بالأحوال العادية القيمة 3 وهذا يعني أن ذاكرة الرام B800 تكون معنونة بالطريقة الزوجية/ الفردية ولكن عند العمل مع bitplane2 تكون قيمته 7 , والمسجل Read Map select يتم من خلاله تحديد bitplane الفعالة، ومسجل Graphics Mode Register يحتاج القيمة 0

Register	Bitplane	Video RAM
Map mask register (sequencer controller)	04H	03 H
Memory mode register (sequencer controller)	07H	03H
Read Map select register (Graphics controller)	02H	00H
Graphics mode registrer (Graphics controller)	00H	10H
Miscellaneous register (Graphics controller)	04H	0EH

يظهر الجدول قيم المسجلات للوصول bitplane2 ومن ثم الوصول إلى الذاكرة الرام الاظهارية، إن كل ما يسبق يتم من خلا ل التابع

```
void put_new_character(BYTE ascii,BYTE *glyph)
}
        int i,c=0;
        char *screen=(char *)(0xD0000000+0xB8000)
        int address=(((ascii*2)*16)
        */Removing all what we have Done/*
        make_screen_ready();
        for (i=address;i<((address)+(16));screen[i++]=glyph[c++];
        restore_screen();

{
```

إذا للوصول Bitplane2 نعدل المسجلات كما رأينا ومن ثم للوصول إلى الذاكرة الفيديوية مرة ثانية يجب استعادة حالة المسجلات الأصلية.

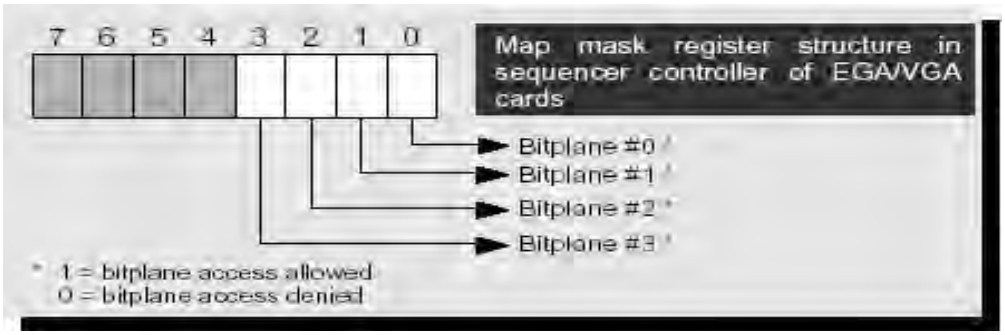
ونحن نقوم من خلال هذا التابع بتحميل حرف واحد فقط وهذا الحرف مخزن بصورة ست عشرية 16 بايت

```
x00,0x00,0x00,0x10,0x00,0x38,0x47,0x18,0xE0,0x00,0x00,0x00,0x00,0x00,
0x00,0x00]0  glaph[16]=[
```

نقوم بداية بوضع عنوان الجدول المراد التحميل فيه في Screen ومن ثم نصل إلى Index الحرف حيث ارتفاع الحرف كما ذكرنا 32 بايت * رقم ASCII المراد الوضع مكانه مثلا 129 , ومن ثم نقوم من خلال التابع make_screen_ready() بتجهيز المسجلات للكتابة يالـ Bitplane2 وحفظ حالة المسجلات الباقية ونقوم بداخل هذا التابع بتفعيل Bitplane2 من خلال التابع setplane(2) ومن ثم نقوم بتحميل الحرف العربي ضمن ASCII الذي نريده له حيث تم إعطاء اسكي لكل حرف عربي سيتم شرحهم بالملحق , وبعد تحميل الحرف يتم استعادة المسجلات والرجوع للكتابة بالذاكرة الفيديوية من خلال التابع restor_screen().

```
void make_screen_ready()
{
	int i;

	/* sequencer حفظ مسجلات */
	for (i=0;i<0x05;i++) {
		outb(VGA_SEQ_A,i);
		temp_seq[i]=inb(VGA_SEQ_D);
	}

	/* graphics حفظ مسجلات*/
	for (i=0;i<9;i++) {
		outb(VGA_GFX_A,i);
		temp_gra[i]=inb(VGA_GFX_D);
	}
	/*  جعل العنونة الذاكرية خطية بدلا من العنونة- زوجية /فردية-  */
	outb(VGA_SEQ_A,4);
	outb(VGA_SEQ_D,inb(VGA_SEQ_D)|0x04);//&&&& &1&&
	outb(VGA_GFX_A,5);
	outb(VGA_GFX_D,inb(VGA_GFX_D) & ~0x10);//&&&0 &&&&
	outb(VGA_GFX_A,6);
	outb(VGA_GFX_D,inb(VGA_GFX_D) & ~0x02);//&&&& &&0&

    /* تفعيلbitplane2الثانية */
	set_plane(2);
}
```

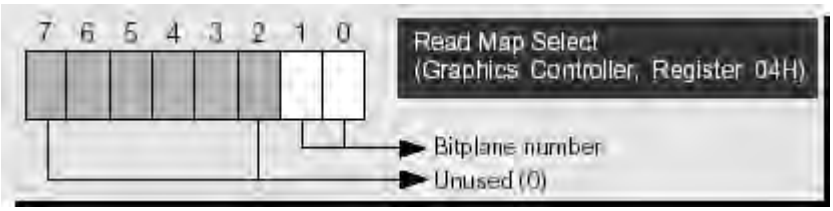
بعد تهيئة المسجلات يتم تفعيل Bitplane الثانية والعمل مع المسجل الثاني للـ Sequ Map mask register لتحديده ونضع فيه القيمة 04 = 00000100



الشكل 159

والرابع من لمسجلات Read Map select register ونضع فيه القيمة:

00000010 = 02



الشكل 160

0+1 For read access in read mode 0, these two bits store the number of the latch register (which is the same as the number of the associated bitplane) that will be copied to the CPU. Read accesses in read mode 1 and chain4 mode are not affected.

```
void set_plane(BYTE p)
{
        BYTE pmask;

        p &= 3;//  0000 0010 & 0000 0011 = 0000 0010= 02 لتحديد
        pmask = 1 << p;// (0000 0001<<00000010)=0000 0100= 04
/* set read plane */
        vga_GFX_outb(4,p);
/* set write plane */
        vga_SEQ_outb(2,pmask);// تفعيل البتبلان الثانية

);
```

وبعد الانتهاء من التحميل يتم ارجاع المسجلات لحالتها الاصلية restor_screen()

```
void restore_screen()
{       /* حذف كل العمل السابق وإرجاع المسجلات لحالتها الاصلية */
        int i;
        for (i=0;i<0x05;i++)
          vga_SEQ_outb(i,temp_seq[i]);
        for (i=0;i<9;i++)
                vga_GFX_outb(i,temp_gra[i]);
}
```

وهكذا نحمل باقي الأحرف.

ملاحظة هامة: يتألف مجموعة محارف ASCII من 256 محرف، تكون مخزنة على شريحة ROM لكرت الشاشة الذي يقوم بتحميلها إلى ذاكرة الرام (Video RAM) للتعامل معها،وبعد ذلك التابع مولد الأحرف يصل إلى المعلومات المطلوبة عن مظهر الأحرف , وبالتالي عند التعديل مع المحارف من 128 إلى 255 التي تم تحميلها في ذاكرة RAM يطلب من المبرمج تحميل الأحرف إلى المكان المخصص لها كلما تمت عملية الإقلاع .

13 7 -الشفرة الموحدة "يونِكود"

كما نعلم تتعامل الحواسب فقط مع الأرقام، وتقوم بتخزين الأحرف والمحارف الأخرى بعد أن تُعطي رقما معينا لكل واحد منها. وقبل اختراع "يونِكود"، كان هناك مئات الأنظمة

للتشفير وتخصيص هذه الأرقام للمحارف، ولم يوجد نظام تشفير واحد يحتوي على جميع المحارف الضرورية. وعلى سبيل المثال، فإن الاتحاد الأوروبي لوحده، احتوى العديد من الشفرات المختلفة ليغطي جميع اللغات المستخدمة في الاتحاد. وحتى لو اعتبرنا لغة واحدة، كاللغة الإنجليزية، فإن جدول شفرة واحد لم يكف لاستيعاب جميع الأحرف وعلامات الترقيم والرموز الفنية والعلمية الشائعة الاستعمال.

ومن الجدير بالملاحظة أن أنظمة التشفير المختلفة تتعارض مع بعضها البعض. وبعبارة أخرى، يمكن أن يستخدِم جدولي شفرة نفس الرقم لتمثيل محرفين مختلفين، أو رقمين مختلفين لتمثيل نفس المحرف. ولو أخذنا أي جهاز حاسب، وبخاصة جهاز النادل (server)، فيجب أن تكون لديه القدرة على التعامل مع عدد كبير من الشفرات المختلفة، ويتم تصميمه على هذا الأساس. ومع ذلك، فعندما تمر البيانات عبر أنظمة مختلفة، توجد هناك خطورة لضياع أو تحريف بعض هذه البيانات.

13 7 1 - "يونِكود" تغير هذا كليا !

تخصص الشفرة الموحدة "يونِكود" رقما وحيدا لكل محرف في جميع اللغات العالمية، وذلك بغض النظر عن نوع الحاسب أو البرامج المستخدمة. وقد تم تبني مواصفة "يونِكود" من قبل قادة الصانعين لأنظمة الحواسب في العالم، مثل شركات آي.بي.إم. (IBM)، أبل (APPLE)، هِيْولِت باكرد (Hewlett-Packard)، مايكروسوفت (Microsoft)، أوراكِل (Oracle)، صن (Sun) وغيرها. كما أن المواصفات والمقاييس الحديثة (مثل لغة البرمجة "جافا" "JAVA" ولغة "إكس إم إل" "XML" التي تستخدم لبرمجة الانترنيت) تتطلب استخدام "يونِكود". علاوة على ذلك، فإن "يونِكود" هي الطريقة الرسمية لتطبيق المقياس العالمي إيزو ١٠٦٤٦ (ISO 10646).

إن بزوغ مواصفة "يونِكود" وتوفُر الأنظمة التي تستخدمه وتدعمه، يعتبر من أهم الاختراعات الحديثة في عولمة البرمجيات لجميع اللغات في العالم. وإن استخدام "يونِكود" في عالم الانترنيت سيؤدي إلى توفير كبير مقارنة مع استخدام المجموعات التقليدية للمحارف المشفرة. كما أن استخدام "يونِكود" سيُمكِّن المبرمج من كتابة البرنامج مرة واحدة، واستخدامه على أي نوع من الأجهزة أو الأنظمة، ولأي لغة أو دولة في العالم أينما كانت، دون الحاجة لإعادة البرمجة أو إجراء أي تعديل. وأخيرا، فإن استخدام "يونِكود" سيمكن البيانات من الانتقال عبر الأنظمة والأجهزة المختلفة دون أي خطورة لتحريفها، مهما تعددت الشركات الصانعة للأنظمة واللغات، والدول التي تمر من خلالها هذه البيانات.

وله العديد من الأنواع والتطورات وهي:

UTF-32
UCS-4
UTF-16
UCS-2
UTF-8
UTF-EBCDIC
UTF-7
وقد اعتمدنا على الترميز UTF-8

UTF-8- 2 7 13

هي اختصار للجملة (bit Unicode Transformation Format-8) وترجمتها (صيغة تحويل نظام الحروف الدولي الموحد بقوة 8 بت)، هذا الترميز وضع من قبل كل من روب بايك و كين تومسن لتمثيل معيار نظام الحروف الدولي الموحد للحروف الأبجدية لأغلب دول العالم، ويتم تشفير الرموز فيها في حجم يتراوح بين بايت واحد و4 بايت للرمز الواحد.فالرمز الإنكليزي يحتاج إلى بايت واحد بينما 1920 الحرف التالية تحتاج إلى بايتتن والحرف العربي يحتاج إلى ثلاث بايتات وغير أحرف تحتاج إلى أكثر من ذلك ويتم تشفير الأحرف بالـ UTF-8 بالطريقة التالية:U+0080 أو U+07FF.

وللـ utf8 ثلاث نماذج:

نماذج utf-8

نموذج utf-8	مجال الأحرف العربية
عادي	u+600 - u+6FF
نموذج A	u+FB50 - u+FDFF
نموذج B	u+FE70 - u+FEFF

إذا وبسبب ترميز الحرف بعدة بايتات تم فصل الأحرف إلى مجموعات وكل مجموعة لها لاحقة محددة:

مجال الرمز(بالشكل الست عشري)	UTF-8 (بالشكل الثنائي)	ملاحظات
000000 - 00007F	0xxxxxxx سبعx	الذي يرمز ببايت تكون لاحقته 0 أي يبدأ البايت بالقيمة 0
000080 - 0007FF	110xxxxx 10xxxxxx ستةx خمسة x	الذي يرمز ببايتين تكون لاحقته 110 للبايت الأول و 10 للبايت الثاني
000800 - 00FFFF	1110xxxx 10xxxxxx 10xxxxxx خمسةx خمسة x أربعة x	الذي يرمز بثلاث بايتات تكون لاحقته 1110 للبايت الأول و 10 للبايت الثاني و 10 للبايت الثالث
010000 - 10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xx xxxx خمسةx خمسة x خمسة x ثلاثة x	الذي يرمز أربع بايتات تكون لاحقته 11110 للبايت الأول و 10 للبايت الثاني و 10 للبايت الثالث 10 للبايت الرابع

إن حرف الياء(ياء بداية الكلمة) ترميزه (encoding) ضمن الينيكود هو U+FEFC ويكون ترميزه في utf-8 بالطريقة التالية:

إن هذا الترميز ينتمي إلى صنف ثلاث بايتات وفقاً للجدول السابق - 000800 00FFFF)) أي سيكون من الشكل 1110xxxx 10xxxxxx 10xxxxxx.

إن قيمة FEFC الستة عشرية تقابلها القيمة الثنائية

1111-111011-111100

يتم وضع البتات الستة عشر ضمن ترتيبهم مكان الـ x

11101111 10111011 10111100

بتحويلهم إلى ست عشري أصبح الثلاث بايتات

EF BB BC

13 8 -القسم التنفيذي لعملية تحميل الأحرف بالنمط الصوري و في النمط المحمي للنظام

إن الحرف االسابق هو حرف الياء ضمن utf-8 وفقا لًلنموذج B الذي اعتمدناه.إن التابع get_utf_char() هو الذي يقوم بتحويل الرمز العربي ذو الثلاث بايتات إلى رمزه الموافق ضمن يونيكود

unsigned get_utf_char(char **str_p)

يأخذ هذا التابع سلسلة من بايت أوأكثر ويقوم بعمليات مقارنة

إذا كان البايت الأول 1110 هومن صنف 3 بايتات وتتم هذه المقارنة if ((c&0xF0)==0xE0)

حيث c هي قيمة البايت الأول

إذا كان البايت الأول(* 110) فهو من صنف 2 بايتات

وإذا كان البايت الأول(0) فهو بايت

وبعد عمليات المقارنة يتم حذف اللواحق من كل بايت ومن ثم دمج البتات سوية للحصول على الرمز الحرف ضمن اليونيكود.

إن اليونيكود يكون عادة ضمن ملف لذلك للحصول على ترميز لابد من فهم الملف قليلاً لنستطيع تحميل ترميز الحرف المطلوب ضمن الملف ومعرفة عرض الحرف وارتفاعه فإن الأحرف لها عرض مختلف وكذلك ارتفاعاتها مختلفة , بالإضافة لمعرفة عرض الخط وارتفاع الخط. والتابع الذي يقوم بتحميل الخط والحصول على المعلومات المطلوبة من الملف هوload_bdf_font()

int load_bdf_font(font_t *font, const char* file_name)

يأخذ بارامترين الاول fontوهو عبارة عن struct سيتم تعبئة معلومات الملف فيه, فهو يحتوي على عرض الخط وارتفاعه

وعدد الأحرف وbitmaps (الصور النقطية للأحرف) حيث سيتم حجزها في الذاكرة , البارامتر الثاني هو اسم الملف الذي يحتوي ترميز اليونيكود

مهمة التابع تحميل الملف ووضع المعلومات المطلوبة في font

حيث يستدعي التابع load_bdf_help() الذي يحجز لكل حرف بحرفه.

وإن كان هدفنا هو التعامل مع الأحرف العربي والإنكليزي فيمكن تحميل الأحرف الإنكليزي من الملف ولكن كما قلنا إن الأحرف الإنكليزية يتم تحميلها من الروم إلى الذاكرة كما تحدثنا سابقا, ويكفي أن نضع مؤشر على الذاكرة (ES:BP) من خلال الوظيفة الفرعية 30 للوظيفة 11 للمقاطعة العاشرة INT 10 ضمن مسجل BH للحصول على أحد الجدوال للحصول على الأحرف الإنكليزية ويتم ذلك من خلال التابع load_rom_font

int load_rom_font(font_t *font, unsigned font_num)

له بارامتران الأول هو font تحدثنا عنه سابقا والثاني هو الجدول المراد التحميل منه

أي الرقم الذي سنضعه ضمن مسجل BH (2-7) - راجع الخدمة 11 –

regs.ax = 0x1130

regs.bx = font_num << 8

تحميل في المسجلAH=0x11

والمسجل AL=0x30

المسجل AH=0x(font_number)

(00000111 – 00000010) 7 -2 = font_number

وبعد تحميل الأحرف العربية والإنكليزية لابد من وضع النمط ضمن نمط الصوري (graphics mode) فعنوان الذاكرة في النمط النصي B8000 وعنوان الذاكرة ضمن النمط الصوري هو A0000 ويتم ذلك من خلال التابع set_graphics_mode()

set_graphics_mode(unsigned depth)

بناء على بارامتر الممر depth يتم وضع النمط أو الدقة

```
Depth:
:1 /* 640x480x1 */
  لكل بت 16 لون 21

:4 /* 640x480x4 */
  لكل بت 16 لون 24

:8 /* 320x200x8 */
  لكل بت 256 لون 28
```

INT 10 وذلك من خلال الوظيفة الفرعية 13 أو12 للوظيفة 00 للمقاطعة العاشرة
(graphics mode)وذلك لجعل النمط صوري

```
regs.ax = 0x0012;
```

بعد ذلك يتم حجز المؤشر عند الذاكرة A0000

```
g_fb.raster = (unsigned char HUGE *)0xA0000L
```

ويمكن جعل النمط نصي (text mode) وذلك من خلال الوظيفة الفرعية 3 للوظيفة 0 في المقاطعة الفيديوية INT 10

```
regs.ax = 0x0003;
```

الآن من أجل كتابة حرف بالعربي أصبح عندنا الأحرف بالإنكليزية alfont والعربية بالـ.afont قبل الخوض في عملية الطباعة والكتابة لابد من الكلام قليلا عن img_t struct وهو عبارة عن الصورة image التي سيتم الكتابة إليها في الذاكرة, ففي النمط الصوري سيتم الكتابة بكسل بكسل(pixel by pixel) وفقا للعمق لكل بت depth , يحتوي هذا الـstruct تابعيين هاميين جدا هما read_pixel وwrite_pixel .

فالتابع write_pexil يقوم بكتابة الحرف بكسل بكسل بالذاكرة في موقع معين ومحدد.read_pexil يقوم بقراءة الحرف بكسل بكسل بالذاكرة من موقع معين ومحدد.لدينا التابع و()bputs والذي يشابه تابع printf و يتم تمرير ثلاث بارامترات له البارامتر الأول هو struct img_t التي تحدثنا عنها سابقا والثاني الجملة المراد طبعتها والثالث هو الـ font حيث يتم تمرير الخط العربي عند كتابة جملة عربية والخط الإنكليزي عند كتابة الجملة الإنكليزية.وعند كتابة الجملة الإنكليزية يتم كتابتها بين تنصيص ولا يوجد مشكلة

```
bputs(&g_fb, "hello", &lfont);
```

وذلك لأن اسكي المحرر(ascii) الذي سيتم الكتابة فيه يوافق نفس اسكي (ascii) الجدول المحمل من الروم وبالتالي سيتم الوصول إلى الحرف مباشرة.ولكن عند الكتابة بالحرف العربي سيتم تحميل الأحرف العربية من الملف وبالتالي عند كتابة الحرف العربي ليكن (خ) فالمحرر سيفهمها رمز ما وسيمرر لها اسكي معين مختلف.وبالتالي لابد من كتابة الأحرف العربية بشكله المرمز بالـ utf-8وذلك بشكل ثلاث بايتات فحرف الميم أول الكلمة هو EF BB B3 وبعد ذلك يتم تحويله إلى شكل اليونيكود من خلال التابع get_utf_char() للحصول ل على .U+FEFC

وبالتالي لكتابة كلمة(يوم)

```
                */ya          waw          mim  /*
bputs(&g_fb, "\xEF\xBB\xB3""\xEF\xBB\xAE""\xEF\xBB\xA1", &afont)
```

ملاحظة(1):

إنني أعتقد أننا إذا كتبنا الكود في محرر يدعم utf-8 النموذج B سنستغني عن هذه المشكلة وعن التابع get_utf_char() وذلك لأن المحرر عندها سيمرر الترميز نفس ترميز utf-8ولكنني لم أجربها

ملاحظة(2)

في نظام تشغيلنا لم ندخل النمط الصوري فيه وهو من أحد الأهداف التطويرية حيث سنقوم بإدخال الواجهات والقوائم والأمور المتعلقة بها, ولكن قمنا بتوفير عملية التعريب لكي لا تشكل عائقا للواجهات وتوفير عملية التعريب على المستوى المتدني في النمط الصوري.

ملاحظة (3)

يوجد برنامج مستقل عن أي نظام تشغيل موجود في cd لتأمين عملية التعريب في النمط الصوري (graphics mode) يمكن إضافته إلى أي نظام

ملاحظة(4)

إن التوابع السابقة التي شرحناها قياسية وعالمية وهذه الأكواد هي الخلاصة النهائية للعديد من العقول وأينما بحثت في الانترنت في هذا المجال ستجد هذه التوابع المفيدة لذلك من الضروري جدا فهمها لمن يريد المتابعة في هذا المجال وإذا أردنا تخصيصها فقط للغة العربية قد نستغني عن الكثير منها ,ولكن بهذه الطريقة بإمكاننا أن نظهر أي لغة بالعالم (ضمن تمثيل اليونيكود) وبسهولة وبدون إي تغيير بالكود فقط تغيير الاستدعاءات.

ملاحظة(5)

عند استخدامنا ترميز utf-8 أصبح ترميزنا عالمي وقياسي فإي جهاز أو نظام يتوافق مع يونيكود يمكن أن يظهر أحرفنا. بعد الانتهاء من مشكلة تحميل الخط العربي بالذاكرة سنتكلم عن المشكلة الثانية صناعة الخط العربي.

13 9 -صناعة الخط العربي

13 9 1 -النمط الصوري (graphicmode)

إن الخط العربي والأحرف العربية مؤمنة في النمط الصوري (graphics mode) بشكل كبير وذلك من خلال الترميزات الكثيرة والكبيرة وقد اعتمدنا منها utf-8

كما ذكرنا سابقاً وتحثنا عنها.

13 9 2 -النمط النصي (textmode)

المشكلة في النمط النصي إن عرض الحرف لا يتجاوز(8أو9) بتات وعرض الحرف في النمط الصوري قد يكون بآيتين أوأكثر.لنأخذ المثال التالي: حرف (الواو) هو باليونيكود u+feee وbitmap له هي:

الثنائي	الســـتة عشري
0000000000000000	0000
0000000000000000	0000
0000000011000000	00C0
0000000111100000	01E0
0000000111100000	01E0
0000000100010000	0110
0000000110010000	0190
0000000111110000	01F0
0000000011110000	00F0
0000000000110000	0030
0000000001100000	0060
1000000001100000	8060
0100000111000000	41C0
0011111110	3F80
	0000

	000000 0000000000 000000

ونلاحظ كيف أن عرض الحرف هو 2 بايت ونحن بالنمط النصي لا نملك سوى بايت أو 9 خانات ولذلك قمنا برسم الخطوط بأنفسنا من خلال برنامج يساعد في عملية الرسم وهو موجود في سيدية المشروع (DFE.EXE) حيث نقوم برسم كل حرف بشكل ثنائي ويحوله إلى شكله الست عشري.

الستة عشري	الثنائي
00	00000000
00	00000000
00	00000000
00	00000000
00	00000000
0C	00000000
12	00000000
0F	00000000
02	00001100
04	00010010
18	

0000111 1 0000001 0 0000010 0 0001100 0 0110000 0 0000000 0 0000000 0 0000000 0	60 00 00 00

وقد رسمنا حوالي 128 حرف وفقا للصفوف (classes) التي سيتم شرحها لاحقاً.

13 10 -كتابة الأحرف الموصلة

معظم الأحرف تحتاج إلى أربعة حالات (ماعدا أحرف القطع تحتاج لحالتين)، إن أحرف القطع هي: (ا, ر, ز, د, ذ, و, ' ')

أحرف القطع	بداية الكلمة	وسط الكلمة	آخر الكلمة	آخر الكلمة موصول
الألف	ا	ـا	ا	ـا
الراء	ر	ـر	ر	ـر
الزاء	ز	ـز	ز	ـز
الذال	ذ	ـذ	ذ	ـذ
الواو	و	ـو	و	ـو

لنأخذ مثال على حرف غير قطع وليكن حرف النون:

أحرف الوصل	بداية الكلمة	وسط الكلمة	آخر الكلمة	آخر الكلمة موصولة
الميم	مـ	ـمـ	م	ـم

النون	نـ	ـنـ	ن	ـن
الهاء	هـ	ـهـ	ه	ـه
الياء	يـ	ـيـ	ي	ـي

فحرف الميم نلاحظ كيف أنه يأخذ الحالات الأربعة بالكلمات (مازن- الماء- علوم - علم).وبالنسبة لباقي الأحرف تم رسمها بالملحق.ولسهولة تحويل الحرف من حرف بداية إلى متوسط إلى نهاية مقطوع قبله أو موصول قبله

تم تقسيم الأحرف إلى أربعة صفوف وبما أننا اعتمدنا العمل على Text mode وقمنا بتحميل الأحرف مكان أحرف الاسكي الموسعة أي (128 - 255). ف أسكي الأحرف العربية سيكون ضمن هذا المجال أي 1xxxxxxx ونحن عندنا كل صف حوالي 28 حرف أي يحتاج 5 بتات لتمثيل الحرف 2^{5} = 32 فتم تخصيص أول خمسة بتات لتمثيل الحرف 1xxxxxxx بقي عندنا البتان الخامس والسادس تم تخصيصهم للصفوف الأربعة (00 – 10 – 01 – 11)

لصف	المجـــــــال (الثنائي)	المجا ل (العشري)	نوع الحرف	أمثلة
0	10000000 10011111 -	128 159 -	نهاية قبله حـــرف وصل	حرف الباء نهاية الكلمة (ـب) 10000001 129
1	10100000 10111111 -	160 191 -	حرف بداية	حرف الباء بداية الكلمة (بـ) 10100001 161
0	11000000 11011111 -	192 223 -	حرف متوسط	حرف الباء بداية الكلمة (ـبـ) 11000001 193

1	11100000 11111111 -	223 255 -	ح رف نهاية قبله قطع	حرف الباء بداية الكلمة (ب) 11000001 224

وكما نعلم أن اللغة العربية من اللغات التوصيلية ذات الأحرف المتصلة مع بعضها وبالتالي عند الكتابة باللغة العربية فالحرف المطبوع سوف يتغير وفقاً للحرف الذي قبله والذي بعده فعند كتابة كلمة محمد تتم بالمراحل التالية:

(م مح محم محمد)

الحرف	التعديل
م	م (ميم نهاية وقبلها حرف قطع)
مح	م(حرف بداية), ـح(حرف نهاية قبله موصول)
محم	م(حرف بداية), ـحـ(حاء متوسط) ـم(ميم نهاية قبلها موصول)
محمد	م(بداية),ـحـ(حاء متوسط),ـمـ(ميم متوسط),ـد(دال نهاية قبلها موصول)

إذن فعند كتابة الحرف نقوم بتعديل الحرف الذي قبله و فقاً للخوارزمية التالية:

بداية تتم طباعة الحرف بشكل افتراضي كحرف نهائي مقطوع

ولكن قبل ذلك يتم فحص الحرف الذي قبله فإذا كان حرف قطع يتم كتابة الحرف بشكل نهائي قطع ولا يتم تعديل الحرف الذي قبله.

وآلا إذا كان الحرف الذي قبله حرف وصل فنحن هنا نحن أمام حالتين:

إذا كان الحرف الذي قبل الحرف المراد طبعه هو حرف قطع سيتم تغير الحرف القبل الحرف المراد طبعه إلى حرف بداية والحرف المراد طباعته كحرف نهاية قبله موصول.

مثال: كلمة ذكر

لنفرض أننا كتبنا (ذك) ونريد كتابة الراء بداية ننظر للحرف السابق وهو كاف نجد أنه حرف وصل وبالتالي يجب تعديله وبالتالي لكي نعدله يجب أن ننظر للحرف الذي قبله وهو في مثالنا حرف قطع ولذلك سيتم تعديل حرف الكاف من (حرف نهاية قبله قطع (ك)إلى حرف بداية(كـ)) وسيتم طباعة الحرف المراد طباعته وهوالراء في مثالنا كحرف نهاية قبله موصول (ـر) ,وبالتالي ستصبح الكلمة (ذك) -----> (ذكر).

إذا كان الحرف الذي قبل الحرف المراد طبعه هو حرف وصل سيتم تغير الحرف القبل الحرف المراد طبعه إلى حرف متوسط والحرف المراد طباعته سيكتب كحرف نهاية قبله موصول.

مثال: سمر

لنفرض أننا كتبنا (سم) ونريد كتابة الراء بداية سينظر للحرف الذي قبله فيجده حرف وصل (م) وبالتالي سينظر للحرف الذي قبل ما قبله فيجده حرف وصل وهو (سـ) وبالتالي سيطبع حرف الميم كحرف متوسط(ـسـ)ويطبع حرف الراء المراد طباعته كحرف نهاية قبله موصول(ـر), وبالتالي ستصبح الكلمة (سم) -----> (سمر).

وكذلك تطبق هذه الخوارزمية في عملية الحذف.

ملاحظة:

عند تعديل الحرف السابق يتم حذفه أولاً أي وضع مكانه فارغ بالذاكرة ومن ثم طباعة الحرف المعدل في نفس المكان

13 11- الكتابة من اليمين إلى اليسار (LeftToRight بدلا من RightToLeft)

وهي مشكلة بسيطة تمت كتابتها لتوضيح الكود.إن الكتابة العربية تكتب من اليمين إلى اليسار بدلا من اليسار إلى اليمين وهنا سيتم اللع25, لمؤشر فعند الكتابة بالنمط النصي 25 x 80, فبداية المؤشر تكون عادة عند القيمة (0, 0) ومن ثم يزداد واحد تلو الواحد.

((79 ,2)...(79 , ,)....(9(1 ,0).....(10 ,0).....(0 ,0)

فبدل من وضع المؤشر عند البداية (0, 0) وزيادة المؤشر سنضع المؤشر عند الـ(0, 79) وسيتم إنقاص القيمة واحد واحد وعند الوصول إلى العامود الأول سنضيف القيمة 160 وذلك للوصول إلى السطر التالي من الطرف اليمين ومن ثم نعاود الإنقاص حتى الوصول إلى العامود الأول وهكذا.... وعند التحويل إلى اللغة الإنكليزية يتم إزاحة الحرف الإنكليزي وذلك بإعادة طباعة كل الكلمة الإنكليزية بإزاحة إلى اليسار كما يفعل (الورد).....! و بالخلاصة:

التعديل سيكون بحركة المؤشر

وعند الحذف

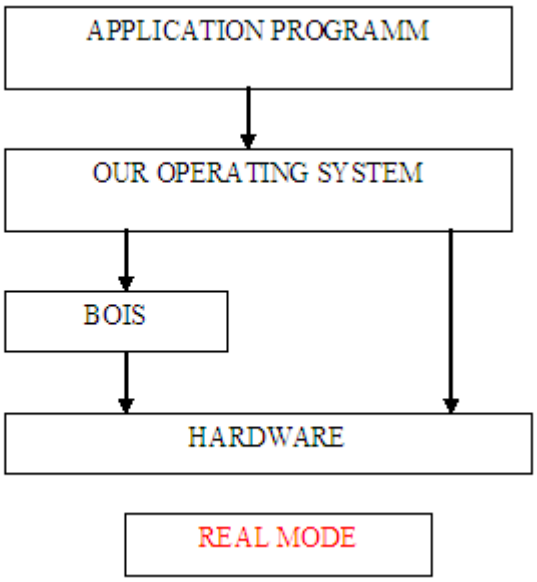
وعند ScrollUp.

13 12 -قسم التنفيذ

الآن ننتقل لتطبيق الدراسة النظرية لما سبق عن الاظهار والشاشة لتطبيقه في نظام تشغيلنا. أرجو قراءة قسم التعريب قبل البدء في هذا القسم, فهز يحتوي القسم النظري والقسم العملي الخاص به معاً والان سنتحدث عن قسم الاظهار وتطبيقه العملي.

لقد اعتمدنا العمل في النمط النصي (TextMode) ضمن النمط المحمي للنظام (Pmode)

ولذلك وكما تحدثنا سابقا فإننا لن نستطيع استخدام مقاطعات البيوس أبدا بل يتوجب علينا تغليفها والعمل مباشرة مع القسم الصلب Hardware.



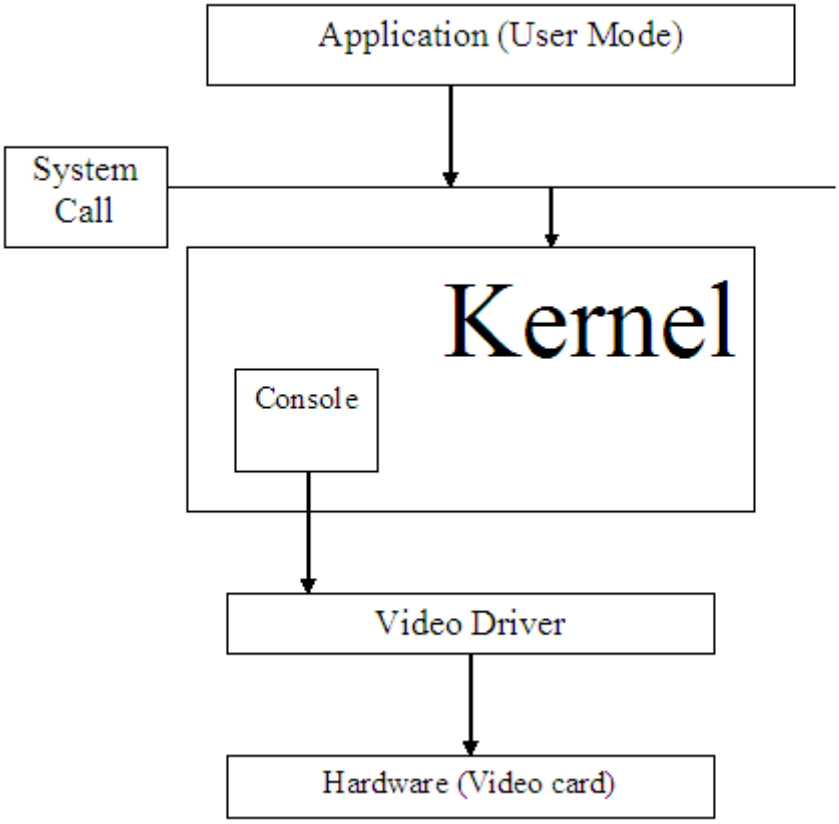
الشكل 161

فكما نلاحظ من الشكل فالعمل بالنمط الحقيقي يسمح لنا التعامل مع البيوس BOIS مما يسهل الكثير من الأعمال وأحياناً نتعامل مباشرة مع Hardware ولكن هذا النمط له سيئات، كما وجدنا في قسم النمط المحمي.

بل يجب BOIS على مستوى البيوس Pmodeكما نلاحظ أنه لا يمكن العمل في

العمل مباشرة على مستوى الهاردوير (Hardware) والتعامل مباشرة مع الذاكرة مما يتطلب من المبرمج صعوبة أكبر ولكنه أفضل للنظام ككل.كما وجدنا سابقاً فإننا فعند التعامل مع الخرج هناك مستويات لابد من المرور خلالها والتعامل معها فعندنا نحن مستويين رئيسين مستوى الكونسول (concole) والذي هو في مستوى النواة

(kernel) وهو الذي يغلف الـ video driver ويتعامل معه وهو الوحيد الذي يتعامل معه وباقي النظام يتعامل مع الكونسول (concole).



الشكل163

13 12 1- العمل على مستوى الكونسول (concole)

الهدف من الكونسول هو التهئية لعمل نظامنا متعدد المستخدمين وقد قمنا بالعمل على تهيئة الكونسول للعمل على 10 مستخدمين , وبالتالي ممكن أن يكون عندنا 10 مستخدمين ولكل مستخدم الكونسول الخاص به , مما يتطلب منا تغليف توابع الفيديو السابقة كلها حتى لا يتمكن بالوصول إلى هذه التوابع سوى العمليات الشغالة الفعالة (Current process)

قبل الخوض في توابع الكونسول لا بد من التعريف ببنية الكونسول

13 12 1 1- بنية الكونسول (console)

```
//! Virtual console structure.
typedef struct console
{
	// --- VIDEO ---                                        //
	word  *vid_buffer;        //!< Video buffer.
	word  cur_pos;            //!< Cursor position.
	byte  cur_color;          //!< Current text color.

	// --- KEYBOARD ---                                     //
	byte  num_lock,           //!< NUM-LOCK flag.
	      caps_lock,          //!< CAPS-LOCK flag.
	      scroll_lock;        //!< SCROLL-LOCK flag.

	word  keyb_buf_read,      //!< The first character to read from the
buffer.
	      keyb_buf_write,     //!< The last character insered into the
buffer.
	keyb_buf_count;//How many characters are present into the buffer.

	//! The keyboard circular buffer.
	word  keyb_buffer[KEYB_BUF_DIM];
} console_t;
```

فالكونسول عبارة عن struct والذي يهمنا فيه ما يخص الفيديو والباقي تابع للوحة المفاتيح

vid_buffer: هو عبارة عن مؤشر لبفر الفيديو المؤقت الذي تتم الكتابة عليه ومن ثم عندما الكونسول فعال (Current Concole) (أي الكونسول الذي تتم فيه الكتابة أو الإظهار)يتم الكتابة مباشرة إلى العنوان الفيزيائي.

cur_pos: هو موقع المؤشر وهو من النمط Word يحتوي x. والـ y معاً.

cur_color: وهو بايت الذي يحدد الألوان (لون النص – ولون الخلفية) ,

البتات الأربعة الدنيا للنص والأربعة العليا للخلفية

13 12 1 2 -توابع الكونسول (console)

نقوم بتعريف مصفوفة من الكونسولات تتسع لعشر كونسولات بالإضافة للكونسول الرئيسي main concole (0)

```
console_t vir_cons[K_TOT_VIR_CONS+1]// K_TOT_VIR_CONS = 10
```

get_console_addr

) يعيد لك بنية الكونسول كاملةًindexوهو عبارة عن تابع تعطيه رقم الكونسول (

```
console_t *get_console_addr(int c)
```

get_curr_console

)Current_concole(وهو تابع يعطي الكونسول الفعال الحالي (

```
get_curr_console()
```

set_curr_console

هو الفعال cجعل الكونسول ذو الرقم

```
set_curr_console(int c)
```

switch_to_console

```
bool switch_to_console(int c);
```

وهو تابع للتبديل بين الكونسولات.

بدايةً يفحص إذا كان الكونسول ذو الرقم c هو الكونسول الفعال (Console Current) يعيد true ويخرج ,

وإذا لم يكن الكونسول الممر هو الكونسول الفعال (Console Current) عندها يقوم بنسخ محتويات الكونسول القديم الموجودة بذاكرة الفيديو الفيزيائية التي تظهر على الشاشة (الكونسول الرئيسي ذو الرقم 0) في الكونسول الفعال (Console Current)- والذي ليس c- ومن ثم يجعل الكونسول ذو الرقم c هو الكونسول الفعال (Console Current) وبعد ذلك ينسخ محتويات الكونسول الجديد ذو الرقم C والذي أصبح كونسال فعال Console (Current) إلى الذاكرة الفيديوية ومن ثم بتحديث مسجلات لوحة المفاتيح.

init_boot_console

```
init_boot_console();
```

يتم فيه تهيئة الكونسول الرئيسي ذو القيمة (0) بالذاكرة الفيزيائية المباشرة وقبل عملية تحويل الذاكرة إلى ذاكرة تعمل على مبدأ الصفحات (paging) بحيث يصبح مؤشر البفر عنده يؤشر على الموقع الفيزيائي للذاكرة مباشرة عند القيمة 0XB8000 وبدون إضافة عنوان بداية المقطع (راجع قسم الذاكرة), ويتم تهيئة المؤشر أيضا وتهيئة اللون الافتراضي.

init_main_console

```
init_main_console()
```

تهيئة الكونسول الرئيسي (0) هذه التابع نفس التابع (init_boot_console() ولكن يختلف عنه بأنه هنا تم تهئية الذاكرة من ذاكرة خطية إلى ذاكرة تعمل بطريقة الصفحات (pages) وبالتالي سيتم وضع مؤشر البفر عند العنوان الفيزيائي ولكن باستدعاء التابع PHYSICAL(0xB8000) أي سيصبح العنوان

0xE0000000 + 0xb8000 = 0xE00b8000

وبالتالي أصبح عندنا الذاكرة الفيزيائية هي الكونسول (0) فأي كونسول آخر ينقل محتوياته إلى هذا الكونسول فكأنه يكتب مباشرة على الذاكرة الفيزيائية.

create_virtual_console

```
Void create_virtual_console()
```

تهيئة الكونسولات الافتراضية

يتم تهيئة كل كونسولات وحجز لكل بفر كونسول حجم بالذاكرة بمقدار حجم الشاشة (80*25*2) ومن ثم تصفير قيمه بالذاكرة, وتهيئة المؤشر وتهيئة ألوانه.

ومن ثم يتم نسخ محتويات الكونسول الرئيسي (الفعال) إلى الكونسول الأول لجعله الكونسول الفعال ومن ثم نقوم بجعل الكونسول الأول هو الكونسول الفعال.

Clrscr()

```
void clrscr()
```

هذا التابع هدفه تغليف التابع video_clrscr الموجود في مستوى driver الفيديو الذي لايتم استدعائه إلا من قبل الكونسول .

إذالا يوجد عملية فعالة (Current Process) يتم استدعاء video_clrscr للكونسول الرئيسية (0)(أي سيتم الكتابة مباشرة على الذاكرة الفيزيائية),

إذا كان كونسول العملية الفعالة هو الكونسول الفعال أيضاً يتم استدعاء video_clrscr للكونسول الرئيسية (0) (أي سيتم الكتابة مباشرة على الذاكرة الفيزيائية),

وإلا إذا كانت العملية الفعالة لا تعمل على الكونسول الفعال سيتم الكتابة على الذاكرة المؤقته لكونسول العملية الفعالة (وهو ليس الكونسول الفعال).

كل التوابع التالية تقوم بنفس عمل تابع clrscr() من حيث تغليف التوابع الخاص بها بحيث لايتم استدعائها إلا للعملية الفعالة وفي حال الكونسو ل التابع للعملية الفعالة هو الفعال عندها تتم الكتابة مباشرة على الذاكرة الفيزيائية وإلا تتم الكتابة على البفر الخاص للكونسول التابع للعملية الفعالة.

وهذه هي أسماء التوابع في مستوى الكونسول وأسماء التوابع في مستوى الـ Video Driver التي تقوم بتغليفها.

التابع في مستوى الكونسول console	التابع في مستوىdriver video
get_color();	k_get_color(console_t *console)
set_color(byte attrib);	k_set_color(console_t *console, byte attribe)
scroll_up();	k_scroll_up(console_t *console)
gotoxy(int x, int y);	k_gotoxy(console_t *console, int x, int y)
kputchar(int c)	video_putchar(console_t *console, byte c);
get_color();	k_get_color(console_t *console)
set_color(byte attrib);	k_set_color(console_t *console, byte attrib)
clrscr();	video_clrscr(console_t *console);

سنخوض بتفاصيل هذه التوابع عند الشرح على driver الفيديو.

13 12 2 -العمل على مستوى الفيديو (Video Driver)

بداية بما أننا نعمل بالنمط النصي (Textmode) فيكون عنوان الذاكري الفيزيائي

هو 0xB8000

```
//! The physical address of the video memory buffer.

#define VIDEO_MEM_ADDRESS   PHYSICAL(0xB8000).
```

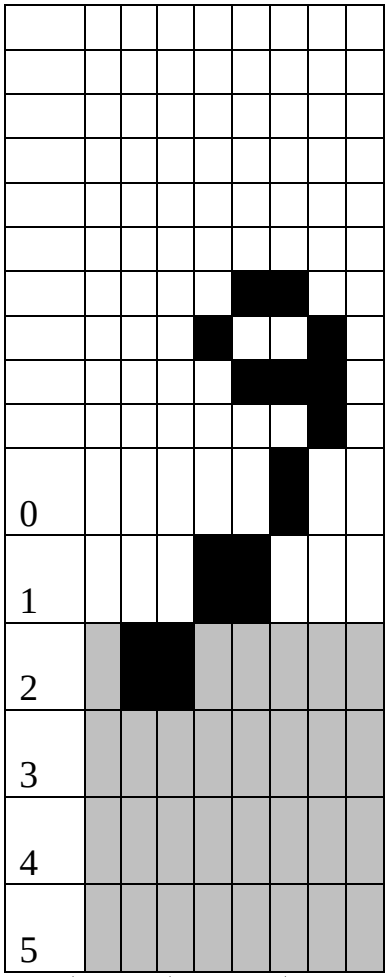
13 12 2 1- توابع مشغل الفيديو Vedio Deriver

video_set_cur_size

الهدف من هذا التابع تحديد حجم المؤشر (cursor) فالمؤشر يجب أن يكون حجمه متناسب مع الأحرف

ليكن عندنا حرف مثل حرف الواو.

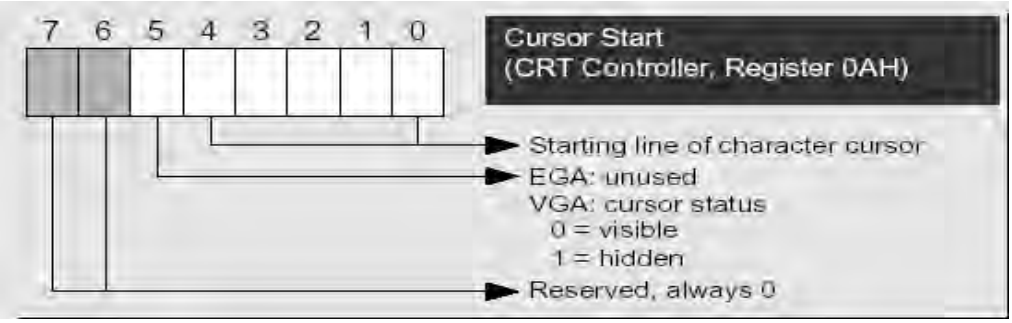
نلا حظ أن المؤشر له خط بداية (start Line) وله خط نهاية (end Line) ونستطيع عن طريق المسجلان 0a, 0b تحديد خطي البداية والنهاية وهذا مانفعله بالتابع video_set_cur_size() والذي يقوم بتحديد حجم المؤشر.



المؤشر لا يكون تحت الحرف مباشرة ولكن عند الحرف الذي سيسبقه أي الحرف الحالي الذي سوف يطبع.

وكما ذكرنا في قسم التعريب قد يكون الحرف 32 بايت نصفه فارغ لذلك أحيانا تكون قيمة بداية المؤشر عند القيمة 31 والنهاية عند القيمة 15 مما يمسح القسم الفارغ كله.

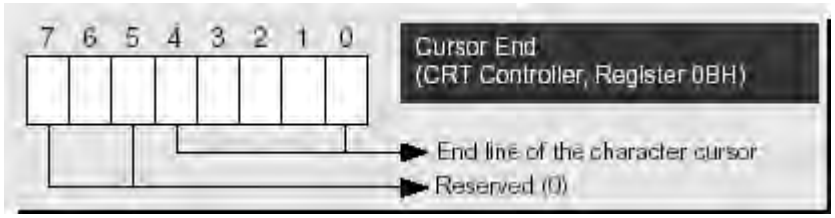
إذن كما قلنا المسجل 0A يحدد قيمة سطر البداية (Start Line)



الشكل 164

البتات الخمسة الأولى من هذا المسجل تحدد قيمة سطر البداية (0 – 31).

البت السادس يستعمل فقط في VGA وذلك عندما نريد إظهار المؤشر أولا نريد(0 يظهر 1 لايظهر).المسجل 0B يستعمل لتحديد السطر النهائي للمؤشر(end cursor).



الشكل 165

void move_cur( )

يتم تحريك المؤشر عن طريق المسجلين 14 – 15 لمسجلات CRT Controller

0EH Cursor Location High

يحتوي البايت العلوي لموقع المؤشر في الصفحة الحالية (Y)

0FH Cursor Location High

يحتوي البايت السفلي لموقع المؤشر في الصفحة الحالية (X)

ولذلك فعندنا المتحول Pos يحتوي على قيمة الـ X والـ Y فعندما يحتوي القيمة

Pos عشري	Pos(ثنائي)	X=pos & 0xff (البايت السفلي)	Y=pos >> 8 (البايت العلوي)
8	00000000 00001000	8 = 00001000	0 = 00000000
88	00000000 10001000	88 = 10001000	0 = 00000000

video_clrscr(console_t *console)

وهو تابع لتنظييف شاشة الإظهار حيث يتم كتابة BLANK - وهو ثابت له لون الخلفية (الافتراضي أسود)- على الذاكرة المؤقته الفيديوية (buffer) التابعة للكونسول (console) ابتداء من عنوان البداية حتى عنوان البداية + حجم الإظهار(50 x 80) وطبعا ذلك بعد ايقاف المقاطعات ومن ثم استرجاعها بعد عملية الكتابة

k_set_cur_pos(console_t *console, word pos)

إن التابع Move_cur لا يتم استدعائه من المكتبات الخارجية , ولكن هومغلف بالتابع k_set_cur_pos()

والذي يأخذ تابعين الكونسول والموقع المراد نقل المؤشر إليه. حيث يقوم هذا التابع بالانتظار إلى أن يصبح الكونسول الممر هو الكونسول الفعال (current) يتم استدعاء عند ذلك التابع Move_cur .

```
If ( console==get_console_addr(0))
    move_cur( console->cur_pos);
```

k_scroll_up

عملية scroll_up وهي عند وصول الكتابة إلى اخرشاشة الاظهار فيجب انزاح شاشة الإظهار كلها للأعلى بمقدار سطر حيث يصبح السطر الأخير فارغ حيث سنقوم بنسخ كامل المحتويات ابتداءكن السطر الثاني حتى حجم كامل الشاشة مع بايت الواصف 80 X 25 X (2)

إلى الذاكرة المؤقتة الفيديوية (buffer).

```
To  = (void *)(console->vid_buffer);
From = ((void *)(console->vid_buffer))+crt_width*2);
Size= crt_width*(crt_height-1)*2
```

Memcpy (to,from ,size)
وهكذا نكون جعلنا السطر الأخير فارغ وجاهز للكتابة فيه ومن ثم نحدث مكان المؤشر

k_gotoxy

جعل المؤشر عند القيمة X,Y بدل من تمريرهم كـ Pos

video_putchar(console_t *console, uint8_t c)

وهو من أهم التوابع يأخذ بارامتران الأول الكونسول والثاني ASCII الحرف المراد طباعته ويتم فيه فحص الحرف الممر ومن ثم اختبار حالة الحرف فإذا كان حرف عادي

(أي ليس حرف تحكم) يتم كتابته مباشرة على الذاكرة المؤقتة الفيديوية (buffer) التابعة للكونسول وذلك بعد إيقاف المقاطعات ومن ثم بعد الكتابة يتم تفعيل المقاطعات وبعد ذلك إنقاص المؤشر بمقدار 1 (بسبب الكتابة بالاتجاه العربي)

(console->vid_buffer)[console->cur_pos] = ((console->cur_color) << 8) | c;

(console->cur_pos)--;

وطبعا يتم فحص حالات الـ Scrollup

init_video()

يتم فيه تهيئة الحالات الافتراضية لملف الفيديو

حيث يتم حجز العنوان الفيزيائي للفيديو, وكذلك تهيئة المؤشر, وحجم المؤشر

13 13 -المراجع :

http://www.hostileencounter.com

http://www.praxagora.com/lunde/j_tools.html

http://www.unicode.org/charts/PDF/U0600.pdf

http://www.lanpoint.com/develop/devlanfactpro/int10/index10.asp

http://www.acm.uiuc.edu/sigops/rsrc /

http://www.inp.nsk.su/~bolkhov/files/fonts/univga/index.html

http://www.langbox.com/arabic/ara_e.html

http://www.arabeye.org

http://www.alkhawarezmi.com

http://www.htl-steyr.ac.at/~morg/pcinfo/hardware/interrupts/inte6re8.htm

http://www.arabteam2000.com

الباب . تنفيذ البرامج على مستوى المستخدم

14 -استدعاءات النظام

مُقدِّمة

توفر أنظمة التشغيل للمعالجات التي تعمل في وضع المستخدم(User Mode) مجموعة من الواجهات لتتفاعل مع أجهزة الكيان الصلب كالمعالج, الأقراص, الطابعات وغيرها، و إن وضع طبقة إضافية بين التطبيقات والكيان الصلب لديه العديد فوائد.

تجعل البرمجة أسهل, محرر المستخدم من دراسة البرمجيات منخفضة المستوى لأجهزة الكيان الصلب.

تزيد من أمان النظام بشكل كبير، طالما أن النواة تختبر صحة الطلب عند الواجهة قبل أن تخدّمها.

تجعل هذه الواجهات النظام أكثر قابلية للحمل طالما أنه يمكن ترجمته وتنفيذه بشكل صحيح على أي نواة توفر نفس المجموعة من الواجهات.

ينفذ نظامنا معظم الواجهات بين معالجات على مستوى المستخدم مع أجهزة الكيان الصلب بما يسمى استدعاءات النظام "system calls" الموجهة إلى النواة, وسوف ندرس الآن آلية عمل استدعاءات النظام مع نواتنا.

14 -1 POSIX APIs و استدعاءات النظام

لنبدأ بتوضيح الفرق بين الـ API (واجهة تطبيقات المبرمج)واستدعاءات النظام، الأول هو تعريف لتابع يحدد كيف يتم الحصول على خدمة متاحة, بينما الأخير هو طلب إلى النواة بواسطة مقاطعة مرنة.إن معيار POSIX يشير إلى الـ API وليس إلى استدعاءات النظام.إن النظام يمكن أن يكون مرخص بهذا المعيار إذا قدم مجموعة معينة من توابع المجيبة I. لبرامج التطبيق, دون أن يهمنا كيف تنفذ التوابع المجيبة.

- من وجهة نظر المبرمجين, هناك فرق بين الـ API و استدعاءات النظام والشيء الجيد هو اسم التابع, أنواع البارامترات, ومعنى الكود المرجع.
- من وجهة نظر مصممي النواة, الفرق أن استدعاءات النظام تنتمي للنواة بينما المكتبات على مستوى المستخدم لا تنتمي للنواة.

14 2 -مقابض استدعاءات النظام و إجرائيات الخدمة

عندما تقوم باستدعاء معالجة على مستوى المستخدم استدعاء نظام, يتحول المعالج إلى نمط النواة ويبدأ بتنفيذ تابع النواة. في نظامنا يجب استدعاء استدعاءات النظام بتنفيذ المقاطعة int $0x80 , وطالما أن النواة تنفذ العديد من استدعاءات النظام, يجب على المعالجة أن تمرر بارامتر يدعى رقم استدعاء النظام "system call number" ليعرف استدعاء النظام المطلوب, يستخدم المسجل eax لهذا الغرض.

ترجع كل استدعاءات النظام قيما صحيحة, وتعبّر القيم الموجبة والصفرية عن إنهاء ناجح لاستدعاء النظام, بينما تعبّر القيم السالبة عن حدوث خطأ، وفي الحالة الأخيرة, إن القيم هي تعبير عن كود الخطأ الذي يجب إرجاعه إلى برنامج التطبيق.

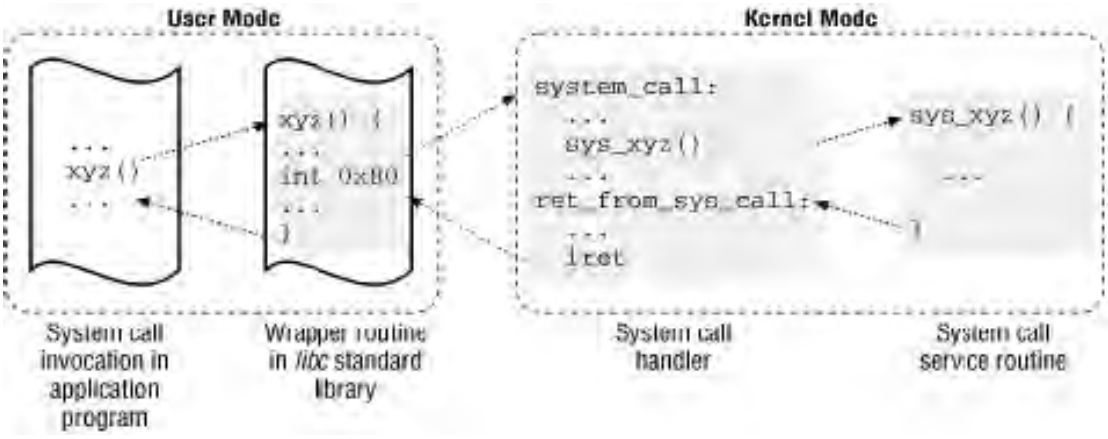
إن مقبض استدعاء النظام والذي يمتلك بنية شبيهة بتلك التي تملكها مقابض الاستثناءات, تقوم بالأعمال التالية:

تحفظ محتوى معظم مسجلات في مكدس نمط النواة (هذه العملية عامة على جميع استدعاءات النظام ومبرمجة بلغة المجمع).

تعالج استدعاء النظام باستدعاء تابع الـ C الملائم والذي يسمى إجراء خدمة استدعاء النظام "system call service routine".

الخروج من المقبض بواسطة التابع ret_from_sys_call() (هذا التابع مبرمج بلغة المجمع).

إن اسم إجراء الخدمة المرتبط باستدعاء النظام xyz() هو غالبا sys_xyz() ويوضح الشكل التالي العلاقة بين تطبيق البرنامج الذي يستدعي استدعاء النظام , الإجراء المطاط المناسب ,مقبض استدعاء النظام , وإجراء خدمة استدعاء النظام وتوضح الأسهم تدفق التنفيذ بين التوابع.



الشكل 166

من أجل ربط كل استدعاء نظام بإجراء الخدمة المناسب , تستخدم النواة جدول ربط استدعاء النظام "system call dispatch table" ويخزن هذا الجدول في المصفوفة

sys_call_table ولديها NR_syscalls مدخل (عادة 256) , ويحتوي المدخل ذو الترتيب n عنوان إجراء الخدمة لاستدعاء النظام ذو الرتبة n.

إن الماكرو NR_systemcalls هو مجرد حد ثابت على الرقم الأعظمي لاستدعاءات النظام القابلة للتنفيذ و هو لا يشير إلى الرقم استدعاء النظام الذي تم تنفيذه حقيقة و إنما قد يحتوي أي مدخل في جدول التوزيع على عنوان التابع sys_ni_syscll() والذي هو إجراء خدمة لاستدعاءات النظام التي لم يتم تنفيذها إنها ترجع كود الخطأ ENOSYS–.

14 3 -تهيئة استدعاء النظام

يهيئ التابع trap_init() المستدعى عند تهيئة النواة مدخل الـ IDT (جدول توزيع المقاطعات) الناسب للشعاع 128 كما يلي:

```
set_system_gate(0x80, &system_call);
```

14 3 1- التابع system_call()

ينفذ التابع system_call() مقبض استدعاء النظام وهو يبدأ برقم استدعاء النظام وجميع مسجلات المعالج التي يمكن أن تستخدم في معالج الاستثناءات على المكدس , ما عدا eflags , cs, eip , ss, esp , والتي تكون قد تم حفظها تلقائيا بواسطة وحدة التحكم , إن الماكرو SAVE_ALL أيضا ناخب القطعة لقطعة معطيات النواة في ds و es:

```
system_call:
pushl %eax
SAVE_ALL
movl %esp, %ebx
andl $0xffffe000, %ebx
```

يحفظ التابع أيضا في ebx عنوان واصف المعالجة الحالية current, يحدث ذلك بأخذ القيمة من مؤشر مكدس النواة وتدويرها بجدائها إلى 8 كيلوبايت، بعدها يحدث تحقق من الصحة على رقم استدعاء النظام الممرر من قبل المعالجة على مستوى المستخدم. إذا أكبر من أو تساوي NR_syscalls يتم إنهاء استدعاء النظام:

```
cmpl $(NR_syscalls), %eax
jb nobadsys
movl $(-ENOSYS), 24(%esp)
jmp ret_from_sys_call
nobadsys:
```

إذا كان رقم استدعاء النظام غير متوافق, يخزن التابع قيمة الـ ENOSYS- في موقع المكدس حيث تم تخزين المسجل eax. ثم يقفز إلى ret_form_sys)call(). بهذه الطريقة, عندما تتابع المعالجة تنفيذها على مستوى المستخدم, ستجد قيمة سالبة مرجعة في الـ eax.

14 4 -تمرير البارامترات

كما في التوابع العادية, غالبا ما تتطلب إستدعاءات النظام بعض بارامترات الدخل والخرج , والتي يمكن أن تحتوي على قيم حقيقية أو عناوين توابع ومتحولات في مجال ذاكرة المعالجة على مستوى المستخدم user mod process , بما أن التابع system_call() هو نقطة الدخول الوحيدة إلى جميع إستدعاءات النظام , فكل واحدة منها لديها على الأقل بارامتر واحد وهو رقم استدعاء النظام الممرر في المسجل eax , على سبيل المثال إذا استدعى التطبيق الإجراء fork() فإن المسجل eax سيحمل بالرقم 12 قبل تنفيذ المقاطعة int $0x80.

إن استدعاء النظام fork() لا يتطلب بارامترات أخرى. لكن تحتاج إستدعاءات أخرى إلى أن نمرر بارامترات إضافية من التطبيق. عل سبيل المثال استدعاء النظام nanosleep يتطلب بارامترين بالإضافة إلى الرقم (رقم استدعاء النظام).تمرر بارمترات التوابع العادية بكتابة قيمها في مكدس البرنامج الفعال (سواء في المكدس على مستوى النواة أو المكدس على مستوى المستخدم). أما بارامترات استدعاء النظام فتمرر إلى برنامج تخديم استدعاء النظام من خلال مسجلات المعالج. تنسخ إلى المكدس على مستوى النواة.لماذا لا يتم نسخ البارامترات مباشرة إلى ألمكدس على مستوى النواة ؟ أولا كان, العمل على مكدسين في نفس الوقت أمر معقد, وأكثر من ذلك استخدام المسجلات يجعل بنية برنامج تخديم استدعاء النظام مشابه إلى برامج تخديم الاستثناءات (exception handlers).أيا كان, من أجل تمرير البارامترات في المسجلات يجب تحقيق شرطين:

طول البارامتر لا يتجاوز طول المسجل والذي 32 بت.

ألا يتجاوز عدد البارامترات الست بارامترات (بما في ذلك رقم استدعاء النظام الممرر في eax) , طالما أن عدد مسجلات المعالج محدود جداً.

إن الشرط الأول محقق دوماً طالما أنه تبعا لمعيار POSIX, فإن البارامترات ذات الطول الأكبر من 32بت يجب أن يمرر بتحديد عناوينهم.أما في حالة أن عدد البارامترات أكبر من 6, عندئذ يستخدم مسجل للإشارة إلى منطقة من الذاكرة في مجال عنونة المعالجة والتي تحتوي على قيم البارامترات.

إن المسجلات الستة المستخدمة في تخزين بارامترات استدعاءات النظام هي على الترتيب: eax(لتخزين رقم استدعاء النظام),ebx,ecx,edx,esi و edi , إن التابع system_call() , يقوم بتخزين قيم المسجلات في المكدس على مستوى النواة بتطبيق متكرر لتعليمة الأسمبلي push , عندما ينتقل إجراء استدعاء النظام إلى المكدس فإنه يجد العنوان المرجع من التابع system_call() متبوعا بالبارامتر المخزن في المسجل ebx (والذي هو البارامتر الأول لاستدعاء النظام) , ثم البارامتر المخزن في ecx وهكذا.

14 5 -قسم التنفيذ

سنشمل في دراستنا العملية على أجزاء من كل مكتبة استخدمناها في نظامنا تتعلق باستدعاءات النظام وهذه الأجزاء ستساعد القارئ على فهم الأجزاء الأخرى.

أولا تتألف المكتبات بشكل رئيسي من المكتبة sys.h والمكتبة syscall.c والمكتبة syscall.s وسندرسها الآن بالتفصيل:

14 5 1- المكتبة sys.h

تحتوي هذه المكتبة في البداية على اسنادات تقابل استدعاءات النظام مع أرقامها كالمثال التالي:

```
#//! Go back to the system.
#define SYS_exit                0
//! Get a char from stdin.
#define SYS_getchar             1
//! Put a character on stdout.
#define SYS_putchar             2
//! Allocate a memory area.
#define SYS_malloc      3
//! Free a memory area.
#define SYS_free                4
//! Administer the system log book.
#define SYS_syslog      5
//! Get the system ticks.
#define SYS_times       6
//! Create a new thread.
#define SYS_thread_spawn        7
//! Returns the process ID of the current process.
#define SYS_getpid      8
//! Returns the process ID of the parent of the current process.
#define SYS_getppid             9
//! Get the UNIX timestamp.
#define SYS_time                10
//! Fork a new task.
#define SYS_fork                11
//! Sleep some micro-seconds.
#define SYS_nanosleep           12
```

وهكذا بعدد استدعاءات النظام وسنرى كيف ستفيدنا هذه الاسنادات بعد قليل في تمرير رقم استدعاء النظام إلى المسجل eax. ثم نعرف التابع __syscall_return() والذي كما ذكرنا في القسم النظري يرجع إما قيمة موجبة تعني أن الاستدعاء تم بنجاح أو يرجع قيمة سالبة تعبر عن رقم الخطأ في جدول معالجة الأخطاء.

ثم نعرف التابع syscall_والذي له واحد من ستة أشكال مختلفة تبعا لعدد بارامترات استدعاء النظام...وسندرس واحد منها وهو syscall2_ والذي يمرر بارامترين.

تجدر الملاحظة إلى أنه تم تعريف التوابع بهذا الشكل لأن استدعاءات النظام مهما اختلفت فإنها واحدة من هذه الأشكال لذلك تجنباً لتكرار الكود فقد كتبنا هذه الماكروات والتي يمرر لها اسم الاستدعاء كبارامتر يستخدمه في تنفيذ الاستدعاء المناسب من خلال الشكل SYS_##nameوالذي كما رأينا في الاسنادات في الأعلى يعبر عن رقم استدعاءات النظام.ويكون شكل الماكرو كما يلي:

```
#define _syscall2(type, name, type1, arg1, type2, arg2) \
type name( type1 arg1, type2 arg2 )    \
{                                      \
        long __res;                    \
        __asm__ __volatile__ (         \
                "int $0x80"            \
                : "=a"(__res)          \
                : "0"(SYS_##name),     \
                "b"((long)(arg1)),     \
                "c"((long)(arg2)) ); \
        __syscall_return(type, __res);    \
}
```

أما شرح الكود فهو كما يلي:

ننتقل إلى لغة الأسمبلي ثم نستدع المقاطعة int$0x80 التي سنشرحها في الوحدة syscall.s , ثم نستخدم المسجل a.الذي هو نفسه المسجل eax كدخل وخرج فهو كدخل نضع فيه رقم استدعاء النظام من خلال السطر:,"0"(SYS_##name) فإذا كان المتحول name مثلاً fork() يصبح السطر:,"0"(SYS_FORK) والذي هو معرف في الأعلى ومسند له الرقم 12 كما نستخدمه كخرج في السطر:"=a"(__res) أي أن نتيجة المقاطعة الموجودة في المسجل a (eax) نسندها في المتحول __res والذي نمرره للتابع __syscall_return() كي يقوم باختباره.ثم نسند البارامترات تباعا في المسجلات اللاحقة ,ونرجع ال __res.بقي لنا أن نعرف كيف مررنا الاستدعاء إلى الماكرو والذي تم بالشكل التالي:

```
static __INLINE__ _syscall1(int, exit, int, exitcode);
static __INLINE__ _syscall0(int, getchar);
static __INLINE__ _syscall1(int, putchar, int, c);
>>>>>
>>>>>
>>>>>
static __INLINE__ _syscall2(int, nanosleep, const struct timespec *, req,
struct timespec *, rem);
```

14 5 2 -المكتبة syscall.s

تحتوي هذه الوحدة في البداية على متحول يحدد عدد استدعاءات النظام أو بمعنى آخر طول الجدول الذي يحوي استدعاءات النظام هذا المتحول هو SYS_CALL_NR ,ثم نجد الجدول الذي يحوي الاجراءات التي يجب استدعاءها عندما نقوم باستدعاء نظام ما, والذي تقوم المقاطعة int $0x80 بطلب الإجراء منه تبعا للرقم (رقم الاستدعاء) الممرر في المسجل eax كما رأينا سابقاً.

هذا الجدول هو بالشكل:

```
syscall_table:
        .long auto_kill
        .long kgetchar
        .long kputchar
        .long umalloc
        .long ufree
        .long sys_syslog
        .long sys_times
        .long sys_thread_spawn
        .long sys_getpid
        .long sys_getppid
        .long sys_time
        .long sys_fork
        .long sys_nanosleep
```

ثم نجد تابع تخديم المقاطعة $0x80 int والمصرح عنه في مكتبة المقاطعات interribt.h كما يلي:

```
#define MINIRIGHI_INT     0x80
```

ويتابع التصريح في المكتبة interrubt.c كما يلي

```
set_user_gate( MINIRIGHI_INT, &_system_call_entry );
```

أما برنامج التخديم فهو يكون في مكتبتنا هذه بالشكل:

```
_system_call_entry:
        cmpl $(SYS_CALL_NR), %eax
        ja      bad_syscall
        cld
        push   %ds
        push   %es
        ; // These are both the register of the task and
        ; // the arguments of the system call.
        pushl  %ebp
        pushl  %edi
        pushl  %esi
        pushl  %edx
        pushl  %ecx
        pushl  %ebx
        ; // Update segment registers with kernel data descriptor.
        movw $KERNEL_DATA_SEL, %bx
        movw %bx, %ds
        movw %bx, %es
        ; // Execute the system call.
        call    *syscall_table(,%eax,4)
```

وقد كنا قد شرحنا مبدأ عمله في القسم النظري حيث يخزن المسجلات المطلوبة ثم يستدع الإجراء المطلوب من الجدول تبعا لرقم الاستدعاء الممرر في المسجل eax كما في السطر call *syscall_table(,%eax,4). على سبيل المثال:

إذا كان مررنا الرقم 12 في المسجل eax فإننا سنستدعي الإجراء sys_fork تبعا للجدول في الأعلى.

14 5 3 -المكتبة syscall.c

تحتوي هذه المكتبة على تنفيذ الإجراءات المذكورة في الجدول السابق والتي لا يتم تنفيذها في مكتبات النواة الأخرى..مثلا الإجراء kgetchar موجود في المكتبة keyboard.c..أما الاستدعاءات التي لا يتم تنفيذها في مكتبات النواة بشكل مباشر فإننا نعرفها في هذه المكتبة ,على سبيل المثال , الاجراء sys_fork نجده في هذه المكتبة كما يلي:

```
pid_t sys_fork(syscall_context_t *regs)
{
	pid_t __ret;
	__ret = do_fork(regs);
	return __ret;
}
```

حيث يقوم باستدعاء التابع() do_fork الموجود في المكتبة task.c.

14 5 4 -الخلاصة

وهكذا نجد أن عملية إضافة استدعاء نظام جديد لتخديم تطبيقات جديدة هو أمر غاية في الدقة والبساطة فكل ما علينا هو إتباع الإجراءات التالية:

إسناد الرقم المناسب له في المكتبة sys.h.

تحديد عدد البارامترات التي يحتاجها ثم تحديد نوع الماكرو المطلوب وإضافة التصريح في آخر المكتبة sys.h.

إضافة التابع إلى جدول استدعاءات النظام في المكتبة syscall.s.

إما أن يكون هذا الإجراء معرفا في إحدى مكتبات النواة أو أن نضيف تنفيذه في المكتبة syscall.c.

15 -محرر الأوامر "Shell"

مُقَدِّمَة

يعتبر محرر الأوامر الواجهة التي سوف يتعامل معها المستخدم النهائي لنظام التشغيل , لذلك لا بد لهذا المحرر من التعامل مع طرفين مستقلين , المستخدم من جهة و النواة من جهة أخرى.إن هذا التعامل المزدوج محفوف بالعديد من التحديات , إذ لا بد لمحرر الأوامر من التعامل مع توابع النواة الهامة من جهة والتي بمقدورها أن توصلنا إلى داخل النواة والتعامل مع موارد الحاسب براحة تامة , كما عليه أن يأخذ التعليمات من المستخدم بحساسية نسبية فهو عليه أن يكون دقيقا في التعليمات المتاحة مع ترك هامش من الحرية له كي يتنبأ بالتعليمة المناسبة،إذاً الصعوبة كامنة في المزج بين الأداء الدقيق على مستوى النواة , والأداء السهل على مستوى المستخدم.على محرر الأوامر أن يتيح كل ما يستطيع المستخدم استخدامه في الحاسب لذلك هناك دعابة تقول أنه علينا أن نبني محرر الأوامر أولا ثم نقوم ببناء نظامنا وهذا الكلام ليس بعيداً عن الصحة فنحن بمعنى آخر يجب أن نحدد ماذا نريد من نظامنا أن يعمل ثم نقوم ببنائه.

30 جدول

شرح التعليمة	التعليمة في النسخة العربية	التعليمة في النسخة الانكليزية
جعل العملية تعمل في الخلفية	للخلف	bg
عرض محتويات ملف (ضمن نظام Fat12)	فتح	cat
عرض محتويات ملف (ضمن نظام Ext2)	فتح2	cat2
تغيير المسار الحالي (ضمن نظام Fat12)	مسار	cd
تغيير المسار الحالي (ضمن نظام Ext2)	مسار2	cd2
تقوم بخلق مهمة في الخلفية تفحص تكاملية إطار المكدس	فحص	checkmem
مسح محتويات الشاشة	مسح	clear
عرض معلومات مختلفة عن معالج الجهاز	معالج	cupid
تفريغ مسجلات المعالج	تفريغ	dump
تتأكد إذا ما كان الملف قابل للتنفيذ بصيغة elf	تنفيذي	elf
تقوم بتنفيذ العملية أو البرنامج في المقدمة	للامام	fg
طباعة العناوين الذاكرية	ذاكرة	free

الفارغة		
إيقاف تشغيل الجهاز	ايقاف	halt
عرض مساعدة وشرح عن تعليمة ما	مساعدة	help
عرض معلومات كاملة عن القرص الصلب	قرص	idedeviceinfo
قراءة كتلة ما من قناة ide محددة	قراءةق	ideread
كتابة كتلة ما من قناة ide محددة	كتابةق	idewrite
تهيئة بطاقة الشبكة	تشبيك	ifconfig
قتل عملية من خلال الرقم التعريفي الخاص بها	قتل	kill
طباعة معلومات عن الكتل المحجوزة في الذاكرة الافتراضية	ذاكرةا	kmem
عرض محتويات مجلد على شكل قائمة (ضمن نظام Fat12)	عرض	ls
عرض محتويات مجلد على شكل قائمة (ضمن نظام Ext2)	عرض2	ls2
عرض عناوين الذاكرة الفارغة	ذاكرةف	lsframe
عرض عناوين الذاكرة التي تم تغييرها	ذاكرةغ	lspage
عرض البطاقات الموصولة على pci	بطاقات	lspci
تحميل نظام الملفات FAT12 من القرص المرن	تحميل	mount
إلغاء تحمبل نظام الملفات FAT12 إلى القرص المرن	رفع	umount
عرض حالة جميع العمليات	عمليات	ps
عرض مسار الفهرس الحالي	هنا	pwd
قراءة كتلة من القرص المرن	قراءة	read
كتابة كتلة إلى القرص المرن	كتابة	write
إعادة إقلاع الجهاز	اقلاع	reboot
حذف ملف من القرص المرن	حذف	rm
تفريغ ذاكرة القرص إلى القرص المرن	مزامنة	sync
طباعة معلومات نواة النظام	نواة	uname
طباعة التاريخ والوقت الحالي	توقيت	date

الدخول إلى محرر الملفات	محرر	editor
تحديد لون الخط	لون	color
إنشاء فهرس جديد على القرص المرن	جديدف	create
إنشاء حساب مستخدم جديد	جديدم	add_user
نسخ محتويات ملف مصدر إلى ملف جديد	نسخ	copy
قص ملف مصدر من موقعه إلى ملف جديد	قص	cut
إنشاء مخدم للدردشة مع برنامج على نظام windows	دردشة	chat

16 -البرامج التنفيذية

16 -1 صيغة الربط والتنفيذ
Executable and Linking Format (ELF)

تم تطوير صيغة الربط و التنفيذ The Executable and Linking Format (ELF) في مختبرات أنظمة يونكس UNIXو أصبح معياريا بسرعة كصيغة معتمدة للملفات التنفيذية المترجمة تحت بيئة يونكس و مثيلاتها من الأنظمة.

بدأت معيارية هذا النوع بالانتشار بسرعة فور إصداره لأنه تميز بقوة أكبر و مرونة أكثر من ملفات الخرج التقليدية a.out وباقي الأنواع من الملفات الثنائي.نعتبر صيغة ELF الصيغة الثنائية القياسية في أنظمة التشغيل المبنية وفق فلسفة يونكس مثل Linux, Solaris ,SVR4 ,2.x.تتميز صيغة ELF بتحقيق الربط الديناميكي dynamic Linking، التحميل الديناميكي dynamic loading، كما يتميز بطريقة محسنة لخلق المكتبات المشتركة.يتميز تمثيل هذه الصيغة بأن التحكم بالمعطيات في ملف الغرض Object File يتم في منصة مستقلة بالإضافة لميزات تحسينية أخرى تميزه عن الملفات الثنائية العادية.

يوجد ثلاث أنواع رئيسة لصيغة ELF هي التنفيذية executable و إعادة التوضع المسجل المتنوع MISCELLANEOUS REGISTER:

و ملفات الغرض المتشاركة shared object files , و تقوم هذه الأنواع الثلاثة بحفظ الكود و المعطيات و المعلومات المتعلقة ببرنامج ما و التي يحتاجها نظام التشغيل لانجاز العمليات المختلفة على هذه الملفات.و يمكن تلخيص الأنواع الثلاثة السابقة بمايلي:

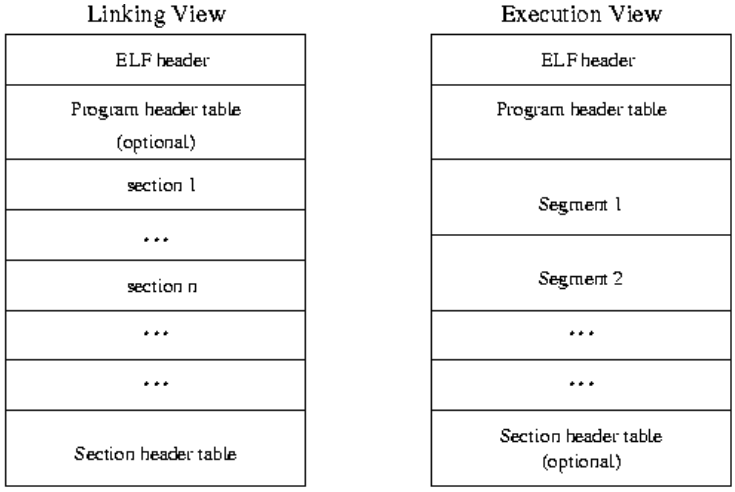
التنفيذي: يؤمن المعلومات اللازمة لنظام التشغيل لخلق العمليات Processes اللازمة لتنفيذ الكود و الوصول للمعطيات المحتواة في الملفات اللازم تشغيلها.
إعادة التوضع: يصف ملف إعادة التوضع كيف يجب ربط الملف مع ملفا الغرض الأخرى other object files لخلق ملف تنفيذي أو مكتبة مشتركة.
ملفات الغرض المتشاركة: يحتوي ملف الغرض المشترك على المعلومات اللازمة في كل من الربط الديناميكي و الستاتيكي. المسجل المتنوع MISCELLANEOUS REGISTER:

سنتكلم عن صيغة ELF متضمنة وصف هذا التصنيف و أقسامه و ترويسته حيث سنتطرق لأقسامه الخمسة و هي:

ترويسة ELF (ELF header)
جدول ترويسة البرنامج (the program header table)
جدول ترويسة القسم (the section header table)
أقسام ELF (the ELF sections)
مقاطع ELF(the ELF segments)

16 2 -بنية ELF

يوجد رؤيتين لكل من الأنواع الثلاثة التي قمنا بالمرور عليها فيما سبق و تدعم هاتين الرؤيتين كل من ربط و تنفيذ البرامج.و يمكن تلخيص هاتين الرؤيتين بالشكل التالي حيث يوضح الشكل اليساري الرؤية الخاصة بالربط link view بينما الشكل اليميني يوضح الرؤية المتعلقة بالتنفيذ execution view.



الشكل 167

تقسم الرؤية الخاصة بالربط الملف إلى أقسام sections بينما الرؤية الخاصة بالتنفيذ تقسمه إلى مقاطع segments .

يهتم المبرمج الحصول على معلومات القسم حول محتويات البرنامج مثل جدول الرموز symbol tables , إعادة التوضع relocation, شفرات تنفيذية خاصة أو معلومات الربط الديناميكية التي سوف يتم استخدامها في الربط

يهتم المبرمج بالحصول على معلومات المقطع مثل توضع نص المقطع text segment أو مقطع المعطيات data segment التي سوف يتم استخدامها في التنفيذ.

تؤمن مكتبة الوصول ل ELF المعروفة باسم libelf أدوات تمكن المستخدم من الوصول و معالجة ملف الغرض بصيغه ELF في كلا الرؤيتين الربط و التنفيذ.

يمكن تلخيص بنية ELF كالتالي:

تصف ترويسة ELF ملف الغرض object file ل ELF بشكل عام و يحتوي المعلومات اللازمة للوصول لباقي الأقسام كالتالي:
يعطي جدول ترويسة القسم Section Header موقع و وصف الأقسام و التي تستخدم في الربط.

يؤمن جدول ترويسة البرنامج Program Header موقع و وصف المقاطع و التي تستخدم في خلق عملية البرنامج و هي خطوة قبل تنفيذه.
تحافظ كل من المقاطع و الأقسام على المعطيات في ملف الغرض و التي تتضمن التعليمات، المعطيات، جدول الرموز، معلومات إعادة التوضع، معلومات الربط الديناميكي.

16 3 -ترويسة ELF

تتميز ترويسة ELF بأنها القسم الوحيد الذي له مكان ثابت لا يتغير ضمن البنية و هذا المكان هو القسم الأول من الملف بينما لا يشترط بباقي الأقسام وجود ترتيب معين أو حتى وجودها بالأصل.تصف الترويسة نوع ملف الغرض و طبعا يكون أحد الأنواع التالية:
relocatable, executable, shared، كما يصف هيكليته و إصدار ELF التي يستخدمها، وبالإضافة لما سبق يحتوي ملف الترويسة هذا على موقع جدول ترويسة البرنامج، جدول ترويسة القسم، جدول النصوص و الأرقام المقابلة لها و أحجام مداخل كل جدول.أخيرا تحتوي الترويسة على موقع تعليمة التنفيذ الأولى.يوضح الشكل التالي ترويسة ELF :

```
#define EI'NIDENT    16

typedef struct -
    unsigned char     e'ident[EI'NIDENT];    //  file ID, interpretation
    Elf32'Half        e'type;                //  object file type
    Elf32'Half        e'machine;             //  target architecture
    Elf32'Word         e'version;            //  ELF version
    Elf32'Addr        e'entry;               //  starting virtual address
    Elf32'Off         e'phoff;               //  file offset to program hdr
    Elf32'Off         e'shoff;               //  file offset to section hdr
    Elf32'Word         e'flags;              //  processor-specific flags
    Elf32'Half        e'ehsize;              //  the ELF header's size
    Elf32'Half        e'phentsize;           //  program hdr entry size
    Elf32'Half        e'phnum;               //  program hdr entry number
    Elf32'Half        e'shentsize;           //  section hdr entry size
    Elf32'Half        e'shnum;               //  section hdr entry number
    Elf32'Half        e'shstrndx;            //  section hdr index for strings
" Elf32'Ehdr;
```

الشكل168

16 4 -جدول ترويسة البرنامج

إن ترويسات البرنامج مهمة جدا في ملفات الأغراض المشتركة و التنفيذية , فجدول ترويسة البرنامج هو عبارة عن مصفوفة من المداخل حيث كل مدخل هو عبارة عن سجل يصف مقطع في ملف الغرض أو معلومات أخرى ضرورية لخلق الملف التنفيذي. يكون عدد المداخل في الجدول و حجم كل مدخل محددا في ترويسة ELF كما ذكرنا سابقا أما كل مدخل في جدول ترويسة البرنامج فهو يحتوي:

النوع type
إزاحة الملف file offset
العنوان الفيزيائي physical address
العنوان الوهمي virtual address
حجم الملف file size
حجم الذاكرة memory image size
إزاحة المقطع في البرنامج alignment for a segment

يقوم نظام التشغيل بنسخ المقطع إذا كان محملا و يكون ذلك عندما p_type is PT_LOAD إلى الذاكرة وفقا لموقع و حجم المعلومات و يمكنك رؤية الحقل في الشكل كأول عنصر في السجل حيث يبين الشكل التالي جدول ترويسة البرنامج:

```
typedef struct -
    Elf32'Word        p'type;       //  type of the segment
    Elf32'Off         p'offset;    //  file offset to segment
    Elf32'Addr        p'vaddr;     //  virtual address of first byte
    Elf32'Addr        p'paddr;     //  segments' physical address, i
    Elf32'Word        p'filesz;    //  size of file image of segment
    Elf32'Word        p'memsz;     //  size of memory image of se
    Elf32'Word        p'flags;     //  segment-specific flags
    Elf32'Word        p'align;     //  alignment requirements
* Elf32'Phdr;
```

الشكل169

16 5 -جدول ترويسة القسم

يتم تعريف جميع الأقسام في ملف الغرض بواسطة جدول ترويسة القسم و بشكل مشابه لترويسة البرنامج فإن ترويسة القسم هو عبارة عن مصفوفة من السجلات حيث أن كل سجل يصف قسم ضمن الملف و يشمل ذلك الوصف:

الاسم
النوع
عنوان الذاكرة البدائي في حال كونه قابل للتحميل loadable.
إزاحة الملف
حجم القسم بالبايتات
الإزاحة
كيفية تفسير المعلومات في المقطع

يوضح الشكل التالي الحقول التخصصية لهذه الترويسة.

```
typedef struct -
    Elf32'Word        sh'name;          //  name of section
    Elf32'Word        sh'type;          //  type of the section
    Elf32'Word        sh'flags;         //  section-specific attributes
    Elf32'Addr        sh'addr;          //  memory location of sectio
    Elf32'Off         sh'offset;        //  file offset to section
    Elf32'Word        sh'size;          //  size of section
    Elf32'Word        sh'link;          //  section type dependent
    Elf32'Word        sh'info;          //  extra information
    Elf32'Word        sh'addralign;     //  address alignment constrai
    Elf32'Word        sh'entsize;       //  size of an entry in section
" Elf32'Shdr;
```

الشكل170

16 6 -أقسام ELF

يوجد عدة أنواع من الأقسام الموصفة بمداخل ضمن جدول ترويسة القسم و يمكن أن تحفظ الأقسام:

شفرة تنفيذية
معطيات
معلومات الربط الديناميكي
معلومات التنقيح
جداول الرموز
معلومات إعادة التوضع
التعليقات
جداول النصوص

الملاحظات

يتم تحميل بعض الأقسام لعملية التنفيذ و أخرى تحتوي معلومات ضرورية للتنفيذ و أخرى تستخدم للربط و يوضح الشكل التالي لائحة ببعض المقاطع الخاصة مع وصف مختصر لكل منها.

Names of sections	Description of the section
.bss	Uninitialized Data present in process image
.comment	Version control information
.data and .data1	Initialized data present in process image
.debug	Information for symbolic debugging
.dynamic	Dynamic linking information
.dynstr	Strings needed for dynamic linking
.dynsym	Dynamic linking symbol table
.fini	Process termination code
.got	Global offset table
.hash	Symbol hash table
.init	Process initialization code
.interp	Path name for a program interpreter
.line	Line number information for symbolic debugging
.note	File notes
.plt	Procedure linkage table
.relname and .relaname	Relocation Information
.rodata and .rodata1	Read-only data
.shstrtab	Section names
.strtab	Usually names associated with symbol table entries
.symtab	Symbol Table
.text	Executable instructions

الشكل171

16 .7 -مقاطع ELF

المقاطع هي طريقة لتجميع الأقسام المتعلقة ببعضها. و كمثال يقوم المقطع text بتجميع الشفرات التنفيذية معا و المقطع data يقوم بتجميع معطيات البرنامج أما مقطع dynamic فيقوم بتجميع المعلومات المتعلقة بالربط الديناميكي.يتألف كل مقطع من قسم أو أكثر و يتم خلق عملية التنفيذ بتحميل و ترجمة المقاطع. يقوم نظام التشغيل بنسخ مقطع الملف لمقطع الذاكرة الظاهرية وفق المعلومات في جدول ترويسة البرنامج و يمكن أن يستخدم نظام التشغيل المقاطع لخلق مورد ذاكري مشترك.

16 8 -التنفيذ Implementation

16 8 1- ترويسة ELF

بالاعتماد على المعلومات السابقة أصبح كتابة ملف الترويسة الخاص ب ELF و الذي له الاسم elf32.h على النحو التالي:

```
//bit ELF address types//----------------------------------------

typedef uint32_t    Elf32_Addr;
typedef uint16_t    Elf32_Half;
typedef uint32_t    Elf32_Off;
typedef int32_t     Elf32_Sword;
typedef uint32_t  Elf32_Word;
```

و أما بالنسبة للترويسة فهي:

```
//ELF32 header//----

#define EI_NIDENT  16      >!//Size of e_ident.[]

-32 !//bit ELF header.
typedef struct elf32_hdr
{
        unsigned char       e_ident[EI_NIDENT;[
        Elf32_Half    e_type;
        Elf32_Half    e_machine;
        Elf32_Word    e_version;
        Elf32_Addr    e_entry;
        Elf32_Off     e_phoff;
        Elf32_Off     e_shoff;
        Elf32_Word    e_flags;
        Elf32_Half    e_ehsize;
        Elf32_Half    e_phentsize;
        Elf32_Half    phnum;
        Elf32_Half    e_shentsize;
        Elf32_Half    e_shnum;
        Elf32_Half    e_shstrndx;
} elf32_hdr_t;
```

أما تعريف المقاطع فهو على النحو التالي:

```
--- //ELF32 Sections----------------------------------------------
//
```

```
!//ELF32 Section header.
typedef struct elf32_shdr
{
        Elf32_Word   sh_name;
        Elf32_Word   sh_type;
        Elf32_Word   sh_flags;
        Elf32_Addr   sh_addr;
        Elf32_Off    sh_offset;
        Elf32_Word   sh_size;
        Elf32_Word   sh_link;
        Elf32_Word   sh_info;
        Elf32_Word   sh_addralign;
        Elf32_Word   sh_entsize;
} elf32_shdr_t;
```

و يبقى تعريف الكثير من الثوابت الأخرى و التي تجدها ضمن الملف المذكور، كما يلزمنا تعريف التوابع التالية للتعامل مع ملفات الربط و التنفيذ ELF:

elf32_copy_sections
elf32_get_entry_point
elf32_load_file

16 8 2 2- تابع فحص بنية الملف elf32_check

يقوم التابع التالي بفحص الملف و التأكد أنه ملف ELF صالح و يتم ذلك عبر التأكد من الأرقام السحرية magic ضمنه و التأكد من بنيته و يوضح لك الكود التالي ذلك:

```
{
#ifdef ELF32_DEBUG
	kprintf("\n\rFile header:");
	kprintf("\n\rmagic[]=%c%c%c%c class=%u data=%u",
		f->e_ident[EI_MAG0],
		f->e_ident[EI_MAG1],
		f->e_ident[EI_MAG2],
		f->e_ident[EI_MAG3],
		f->e_ident[EI_CLASS],
		f->e_ident[EI_DATA]
	);
#endif
	// Magic number check                                   //
	if (
		(f->e_ident[EI_MAG0] != ELF_MAG0) ||
		(f->e_ident[EI_MAG1] != ELF_MAG1) ||
		(f->e_ident[EI_MAG2] != ELF_MAG2) ||
		(f->e_ident[EI_MAG3] != ELF_MAG3)
	)
		return( NULL );

	// Class check
	if ( f->e_ident[EI_CLASS] != ELF_CLASS32 )
		return( NULL );

	// Data encoding check
	if ( f->e_ident[EI_DATA] != ELF_DATA2LSB )
		return( NULL );

	// Check OK!                                            //
	return( f->e_entry );
}
```

16 8 3 -تابع الحصول على موقع الذاكرة لتحميل ملف ELF

get_entry_point( )

```
{
	size_t ret;

	// Check if it is a valid ELF file.
	ret = elf32_check( (elf32_hdr_t *)file_buffer );
	if ( ret == NULL )
		return( ret );

	// Return the entry point.
	return( ret );
}
```

16 8 4 -تابع تحميل ملف elf32_load_file

أولا لدينا النمط التالي:

```
typedef unsigned int size_t
```

يتم تعريف التابع بالترويسة التالية:

```
size_t elf32_load_file  (  uint8_t *    file_buffer,
  int    size,
  int    dir,
  char *    file_name
 )

   Load an ELF32 file into a buffer.

Parameters:
file_name  The name of the file to be loaded.
file_buffer  The buffer where the file will be loaded.

Returns:
The entry point of the file or NULL if the file is not a valid elf32 file.
```

و يكون خوارزمية التابع ببساطة هي:

size_t ret;

```
        // Load the file into the buffer.
        if ( fat12_load_file(file_buffer, size, dir, file_name) < 0 )
                return( NULL );
        // Check if it is a valid ELF file.
        ret = elf32_get_entry_point( file_buffer );
        // The ELF32 file is successfully loaded.
        return( ret );
 }
```

16 8 5 -تابع نسخ مقطع للذاكرة elf32_copy_sections

يتألف هذا التابع من كثير من الكود و لعل ما يهمنا هو ذلك المقطع الذي يوضح عملية النسخ:

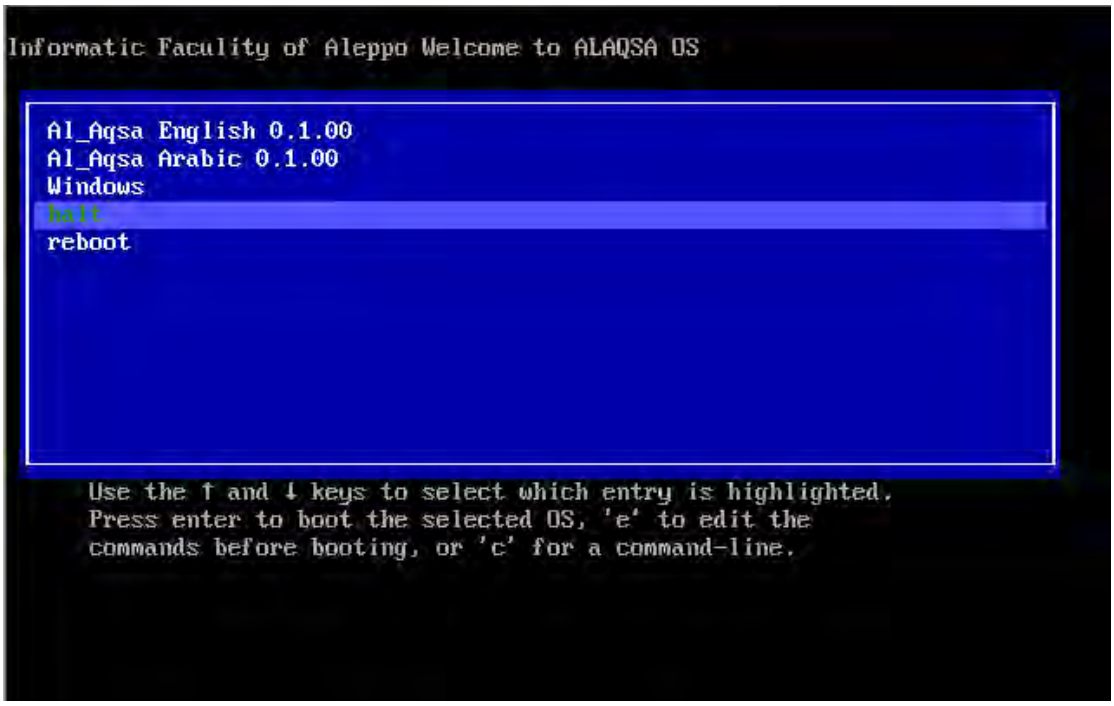
```
        sec = (elf32_shdr_t *)((uint32_t)f->e_shoff + (uint32_t)f);
        for (i=0; i<sec_num; i++)
        {
                copy_section( f, sec );
                (uint32_t)sec += sec_entsize;
        }
        return( f->e_entry );
```

// Copy the sections in memory

الباب □. تنفيذ واختبار المشروع

17 ـ تنفيذ واختبار المشروع

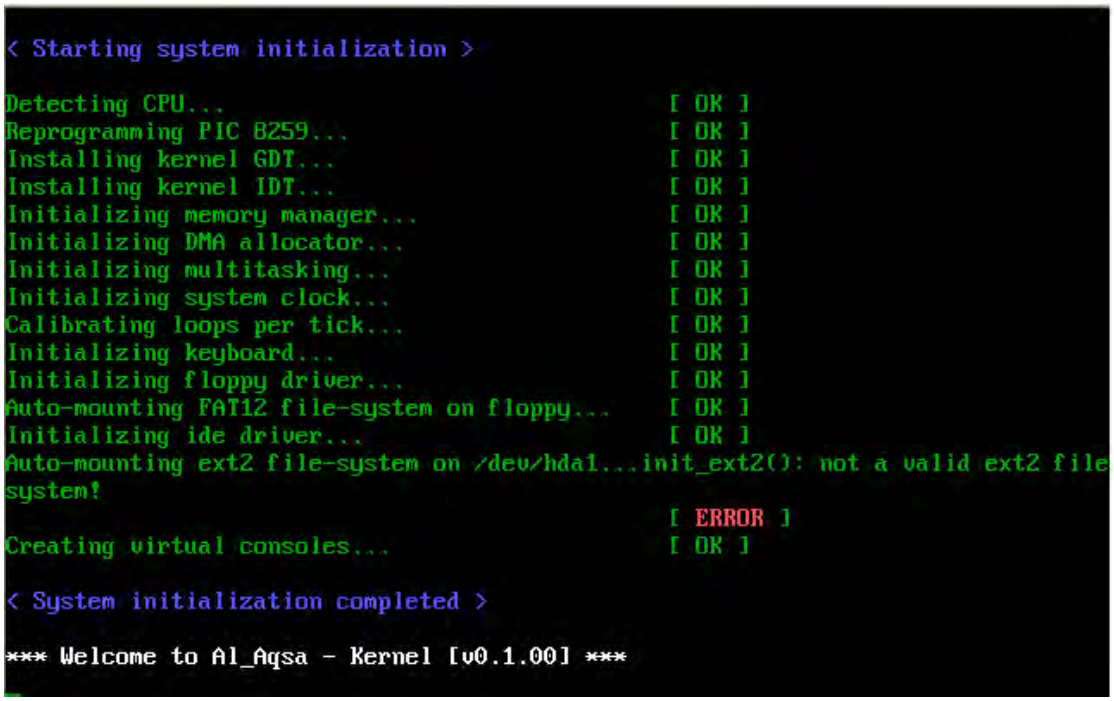
بالإمكان الإقلاع من نظام التشغيل إما عن طريق القرص المضغوط الإقلاعي أو عن طريق القرص المرن، و في كلتا الحالتين سيظهر لديك الشكل التالي:



الشكل172

و الذي يطلب من المستخدم اختيار نسخة نظام التشغيل التي يريد أن يعمل عليها / باللغة العربية أوالانكليزية/. كما تيتح له إيقاف تشغيل الجهاز مباشرة، أو إعادة تشغيل الجهاز، أو الدخول إلى نظام Windows إذا كان مثبتاً على الجهاز.

عند اختيار المستخدم لبدء نظام التشغيل باللغة الانكليزية يبدأ النظام بعملية التهيئة لبدء العمل وتظهر للمستخدم رسائل ليتعرف على مراحل التهيئة التي تجري



الشكل 173

وكذلك بالنسبة للنسخة العربية تظهر له الرسائل المقابلة باللغة العربية:

الشكل 174

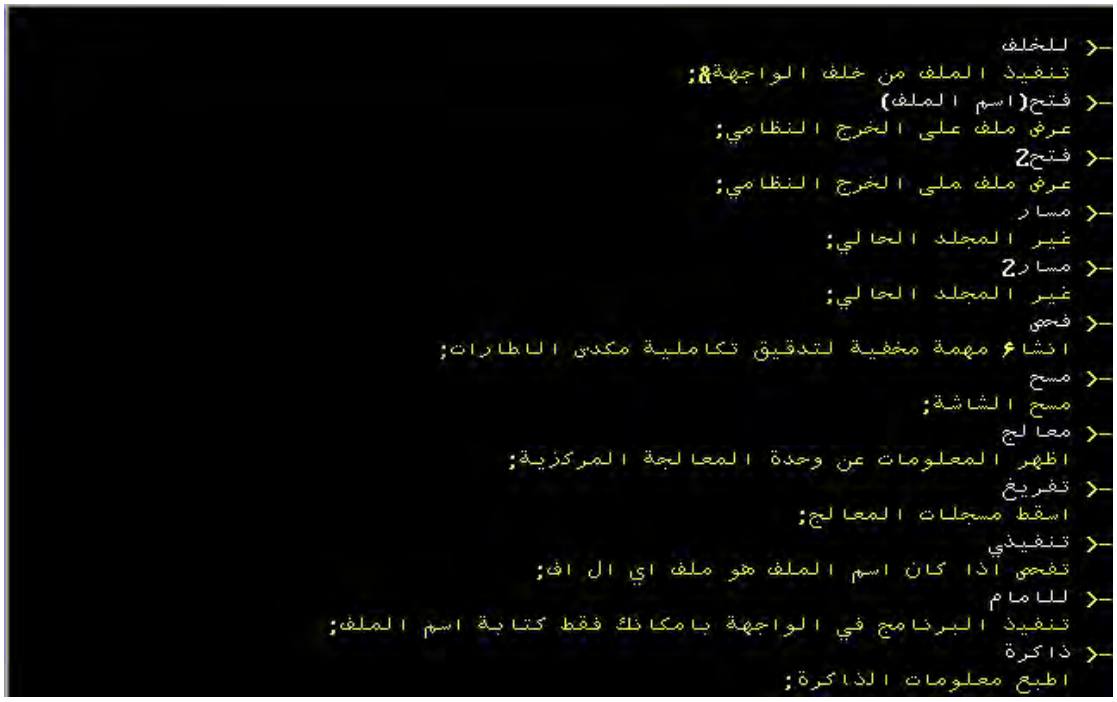
و بعد ذلك يطلب نظام التشغيل من المستخدم إدخال اسم المستخدم وكلمة المرور, وذلك من أجل كل console ويتيح نظام التشغيل لعشر مستخدمين العمل في نفس الوقت على نفس الجهاز.

وبذلك يتمكن المستخدم من الدخول إلى النظام والعمل عليه.

وكمثال على إحدى الأوامر التي يستطيع المستخدم تنفيذها الأمر *عرض* والذي يقابل الأمر ls في النسخة الانكليزية والذي يقوم بعرض محتويات فهرس ما كما في الشكل:

الشكل175

و بإمكان المستخدم عرض كامل الأوامر المتاحة في النظام عن طريق تعليمة *مساعدة* والتي تعطي للمستخدم شرحاً مبسطاً عن مهمة كل أمر من الأوامر كما في الشكل:



الشكل176

والتعليمة المقابلة بالنسخة الانكليزية هي help وتعطي الأوامر وشرحاً مبسطاً بللغة الانكليزية:

```
   Execute a program in foreground (you can simply type <filename> without fg);
-> free
   Print memory informations;
-> halt
   Power down the system (if possible, otherwise halt);
-> help [cmd]
   Show the help of a command;
-> idedeviceinfo
   Display complete info of a specified device;
-> ideread
   Read a block from a device on a specified ide channel;
-> idewrite
   Write a block to a device on specified ide channel;
-> ifconfig <up|down> [promisc]
   Initialize RTL8139 ethernet card;
-> kill <pid>
   Kill the user process with the ID <pid>;
-> kmem
   Dump kernel memory map;
-> ls
   List informations about FILEs;
-> ls2
   List information about ext2 FILEs;
-> lsframe
   Dump free frames;
```

الشكل 177

كما أنه يوجد ضمن نظام التشغيل عملية تعمل كمخدم web. وهي تعمل في الخلف، وتقوم بإعادة صفحة web بسيطة عندما يقوم أي زبون موصول على الشبكة بطلب عنوان IP الخاص بالجهاز الذي يعمل بنظام التشغيل AQSA ضمن مستعرض الانترنت.

والصفحة البسيطة تظهر في الشكل التالي:

الشكل 178

جدول أوامر النظام

31 جدول

شرح التعليمة	التعليمة في النسخة العربية	التعليمة في النسخة الانكليزية
جعل العملية تعمل في الخلفية	للخلف	bg
عرض محتويات ملف (ضمن نظام Fat12)	فتح	cat
عرض محتويات ملف (ضمن نظام Ext2)	فتح2	cat2
تغيير المسار الحالي (ضمن نظام Fat12)	مسار	cd
تغيير المسار الحالي (ضمن نظام Ext2)	مسار2	cd2
تقوم بخلق مهمة في الخلفية تفحص تكاملية إطار المكدس	فحص	checkmem
مسح محتويات الشاشة	مسح	clear
عرض معلومات مختلفة عن معالج الجهاز	معالج	cupid
تفريغ مسجلات المعالج	تفريغ	dump
تتأكد إذا ما كان الملف قابل للتنفيذ بصيغة elf	تنفيذي	elf
تقوم بتنفيذ العملية أو البرنامج في المقدمة	للامام	fg
طباعة العناوين الذاكرية الفارغة	ذاكرة	free
إيقاف تشغيل الجهاز	ايقاف	halt
عرض مساعدة وشرح عن تعليمة ما	مساعدة	help
عرض معلومات كاملة عن القرص الصلب	قرص	idedeviceinfo
قراءة كتلة ما من قناة ide محددة	قراءةق	ideread
كتابة كتلة ما من قناة ide محددة	كتابةق	idewrite
تهيئة بطاقة الشبكة	تشبيك	ifconfig
قتل عملية من خلال الرقم التعريفي الخاص بها	قتل	kill
طباعة معلومات عن الكتل المحجوزة في الذاكرة الافتراضية	ذاكرةا	kmem
عرض محتويات مجلد على شكل قائمة (ضمن نظام Fat12)	عرض	ls
عرض محتويات مجلد على شكل قائمة (ضمن نظام Ext2)	عرض2	ls2
عرض عناوين الذاكرة الفارغة	ذاكرةف	lsframe
عرض عناوين الذاكرة التي تم تغييرها	ذاكرةغ	lspage
عرض البطاقات الموصولة على pci	بطاقات	lspci
تحميل نظام الملفات FAT12 من القرص المرن	تحميل	mount
إلغاء تحميل نظام الملفات FAT12 إلى القرص المرن	رفع	umount
عرض حالة جميع العمليات	عمليات	ps
عرض مسار الفهرس الحالي	هنا	pwd

read	قراءة	قراءة كتلة من القرص المرن
write	كتابة	كنابة كتلة إلى القرص المرن
reboot	اقلاع	إعادة إقلاع الجهاز
rm	حذف	حذف ملف من القرص المرن
sync	مزامنة	تفريغ ذاكرة القرص إلى القرص المرن
uname	نواة	طباعة معلومات نواة النظام
date	توقيت	طباعة التاريخ والوقت الحالي
editor	محرر	الدخول إلى محرر الملفات
color	لون	تحديد لون الخط
create	جديدف	إنشاء فهرس جديد على القرص المرن
add_user	جديدم	إنشاء حساب مستخدم جديد
copy	نسخ	نسخ محتويات ملف مصدر إلى ملف جديد
cut	قص	قص ملف مصدر من موقعه إلى ملف جديد
chat	دردشة	إنشاء مخدم للدردشة مع برنامج على نظام windows

18 -إدارة تصميم نظام تشغيل

مُقَدِّمَة

غطينا في الفصول السابقة العديد من الأساسيات، وألقينا النظر على بعض المفاهيم والأمثلة المتعلقة بنظم التشغيل.و قمنا بإسقاط هذه المفاهيم ضمن مشروع متكامل عملي ولكن دراسة نظام تشغيل موجود تختلف عن تصميم نظام جديد. في هذا الفصل، سنلقي نظرة سريعة على المسائل والعقبات التي يجب أن يضعها المصممون في حسابهم عند تصميم وتحقيق نظام تشغل جديد. يوجد بعض التقليد في تحديد ما هو الجيد وما هو السيء عند جمهور أنظمة التشغيل، ولكن المدهش أن القليل منها تمت كتابته. لعل كتاب Fred Brooks (إصدار 1975) والمدعو شهر الإنسان الأسطوري هو أهم كتاب في هذا المجال. والذي يتعلق بخبرته في تصميم وتطبيق نظام التشغيل .عدلت بعض المواد في طبعة الذكرى السنوية العشرين وأضيفت فصول جديدة (1995)، وقد يكون الكتاب الوحيد الذي يتعامل مع التصميم بشكل جدي هو كتاب Operating System A design approach (1997) و كما يوجد العديد من الأبحاث واستطاعت هذه الأبحاث و الكتب البقاء حتى الآن لأن معظم أفكارها ما زالت صالحة كما كانت عند نشرها لأول مرة.

يعتمد هذا الفصل على هذه المصادر بالإضافة إلى الخبرة الشخصية، بما أنه لا يوجد إجماع بين مصممي نظم التشغيل على أفضل طريقة لتصميم نظام التشغيل، لذا سيكون هذا الفصل شخصياً وتأملياً بشكل أكبر، ومن دون شك أكثر إثارة للجدل من الفصول السابقة.

18 .1 -طبيعة مشكلة التصميم

إن تصميم نظام تشغيل هو أكثر من مشروع هندسي في نفس العلم. من الصعوبة بمكان وضع أهداف واضحة وتحقيقها، لنلق نظرة في البداية على النقاط التالية.

18 .1 .1 -الأهداف

يجب أن تكون لدى المصممين فكرة واضحة عما يريدون من أجل تصميم نظام تشغيل ناجح، يؤدي النقص في الأهداف إلى صعوبة في القرارات المتلاحقة، ولإيضاح هذه النقطة سنلقي نظرة على لغتي برمجة وهما PL/I و C.صممتPL /I من قبل IBM في الستينات لأنه من الصعب دعم كل من Fortran وCobol ومن المربك الخوض في ثرثرة أكاديمية عن أن لغة أفضل من كليهما.

لذا أنشئت لجنة لإنتاج لغة تكون كل شيء للناس وهيPL/I . تحوي هذه اللغة بعضاً من Fortran وبعضاً من Cobol وبعضاً من Algol . فشلت هذه اللغة لأنها افتقرت إلى

رؤية موحدة. ببساطة إنها مجموعة من المزايا في الحرب ضد الآخر، ولكنها مرهقة جداً لترجمتها بفعالية للإقلاع.

لنتحدث الآن عن C، صممت هذه اللغة بواسطة شخص واحد دينس ريتشي لهدف واحد هو برمجة النظام. كانت نجاحاً عظيماً لسبب بسيط جداً، وهو أنه عرف ما يريد وما لا يريد، وبالنتيجة ما زالت منتشرة بشكل واسع بعد عقود من ظهورها.إن وجود رؤية واضحة لمنا نريد هو أمر حاسم. ماذا يريد مصممو نظم التشغيل؟ يتنوع هذا الأمر من نظام إلى آخر. ويختلف في النظم المضمنة عنه في نظم الملقمات.

على أي حال، بالنسبة لنظم التشغيل عامة الأغراض، تخطر في البال أربعة أشياء:

1- تعريف الأفكار التجريدية.

2- تقديم العمليات الأولية

3- ضمان العزل

4- إدارة العتاد الصلب

كل من هذه العناصر سنناقشه أدناه

إن أهم مهمة وأكثرها صعوبة في نظام التشغيل هي تعريف الأفكار المجردة الصحيحة. بعض هذه الأفكار مثل الملفات والعمليات، يستغرق وقتاً طويلاً للوصول إلى فكرة واضحة. وبعضها الآخر مثل الخيوط، أكثر حداثة وأقل نضجاً. مثلاً، إذا تم تفريع عملية متعددة الخيوط، أحد خيوطها متوقف ينتظر إدخال من لوحة المفاتيح، هل سيوجد خيط في العملية الجديدة ينتظر إدخالاً من لوحة المفاتيح أيضاً ؟.

الأفكار المجردة الأخرى تتعلق بالتزامن والإشارات ونموذج الذاكرة ونمذجة الدخل والخرج والعديد من الأمور الأخرى، يمكن تمثيل كل فكرة مجردة على شكل بنى بيانات صلبة. يمكن للمستخدمين إنشاء العمليات والملفات و..الخ. تعالج العمليات الأولية بنى المعطيات هذه. مثلاً يمكن للمستخدمين قراءة وكتابة الملفات. تطبق العمليات الأولية على شكل استدعاءات نظام. من وجهة نظر المستخدم يتشكل قلب نظام التشغيل من الأفكار المجردة والعمليات المتاحة عليها عبر استدعاءات النظام. بما أنه يمكن لعدة مستخدمين الولوج إلى الحاسب في نفس الوقت، يحتاج نظام التشغيل لتقديم آلية لإبقائهم منفصلين. ويجب أن لا تتعارض موارد مستخدم مع موارد مستخدم آخر. يستخدم مفهوم العملية لتجميع الموارد مع بعضها لأغراض الحماية. تحمى الملفات وبنى المعطيات الأخرى بنفس الأسلوب.

يعتبر الهدف المفتاحي لتصميم النظام هو جعل كل مستخدم ينجز عمليات معتمدة على بيانات مرخصة. قد يحتاج المستخدمون أو يودون مشاركة البيانات والموارد الأخرى، وبالتالي يجب أن يكون العزل اختيارياً وتحت سيطرة المستخدم، وهو ما جعل الأمر أكثر صعوبة.إحدى المسائل المتعلقة بهذا الأمر هو عزل الأخطاء. فإذا فشل أحد أجزاء النظام، مثل عملية مستخدم ما فيجب أن لا تؤدي إلى فشل النظام بأكمله. يجب أن يضمن تصميم النظام عزل أجزاء النظام عن بعضها البعض جيداً. وبشكل مثالي يجب عزل أجزاء النظام عن بعضها للسماح بأخطاء مستقلة.

أخيراً، يجب على نظام التشغيل إدارة العتاد الصلب، وبشكل أكثر تحديداً يجب أن يهتم برقائق المستوى المنخفض مثل متحكمات المقاطعات والممرات، ويجب أن يقدم إطار عمل للسماح لبرامج قيادة الأجهزة بإدارة أكبر عدد ممكن من أجهزة الدخل / الخرج مثل الأقراص والطابعات والشاشات.

18 2 ـ لماذا من الصعب تصميم نظام تشغيل ؟

يقول قانون مور Moore الشهير أن العتاد الصلب للحاسب يتحسن 100 مرة كل عشر سنوات ولم يصدر أحد قانوناً يقول أن نظم التشغيل تتحسن 100 مرة كل عشر سنوات. أو حتى أنها تحرز أي تحسن. في الواقع قد نجد بعض أنظمة التشغيل أسوأ من سابقاتها في بعض النواحي الرئيسية كأن تكون أسوأ من النظام UNIX الذي ظهر في السبعينات.

لماذا ؟ غالباً ما يتحمل القصور الذاتي والرغبة في التوافقية مع ما قبله معظم المسؤولية، والفشل بالالتزام بمبادئ التصميم الجيد هو أحد الأسباب. ولكن يوجد المزيد من الأسباب. تختلف نظم التشغيل بشكل أساسي من عدة نواح عن برامج التطبيقات الصغيرة التي تباع في مخازن البيع بسعر 50 دولاراً.

لننظر إلى ثماني مسائل تجعل تصميم نظام التشغيل أصعب من تصميم برنامج تطبيق ما.

أولاً ـ أصبحت نظم التشغيل برامج ضخمة جداً. لا يستطيع أي شخص أن يجلس أمام الحاسب وينتج نظام تشغيل في عدة أشهر. جميع الإصدارات الحالية من تتجاوز المليون سطر من الشفرة يحوي Windows 2000 على 29 مليون سطراً. لا يستطيع أي شخص أن يفهم مليون سطر من الشفرة فما بالك إذا كان 29 مليون سطر. إذا كان لديك منتج لا يستطيع أي من مصمميه أن يفهمه بشكل كمل، فمن الطبيعي أن تكون النتائج بعيدة عن المثالية.لا تعتبر نظم التشغيل هي النظم الأعقد بل يوجد ما هو أعقد مثل حاملة الطائرات. ولكن يمكن تجزئتها إلى نظم فرعية معزولة، فمثلاً، لن يهتم مصممو المغاسل بوحدة الرادار. لأن لكل جزء خاصيته ولا يوجد احتكاك كبير بينها. في حين أنه في نظم التشغيل يحتك نظام الملفات كثيراً بنظام الذاكرة بأشكال غير متوقعة أو محسوبة.

ثانياً ـ يجب على نظم التشغيل التعامل مع مسألة التوازي. يوجد عدة مستخدمين وعدة أجهزة دخل / خرج وكلها فعالة في نفس الوقت. إدارة العمليات المتوازية أصعب بكثير من إدارة فعالية تسلسلية واحدة. حالات السباق والاستعصاءات هما مشكلتان من المشاكل الكثيرة المرافقة للمعالجة المتوازية.

ثالثاً ـ يجب على نظم التشغيل التعامل مع مستخدمين عدائيين بشكل نسبي، وهم مستخدمون يريدون التصادم مع نظام التشغيل أو عمل شيء ممنوع. مثل سرقة ملفات شخص آخر. يجب أن يأخذ نظام التشغيل احتياطاته لمنع هؤلاء من التصرف بشكل غير لائق. لا تملك برامج معالجة النصوص أو تحرير الصور مثل هذه المشاكل.

رابعاً ـ التقليل من شأن الحقيقة القائلة أنه لا يثق جميع المستخدمين ببعضهم. فقد لا يود البعض منهم التشارك ببعض معلوماتهم ومواردهم مع مستخدمين آخرين محددين. يجب أن

يجعل نظام التشغيل هذا الأمر ممكناً. ولكن بطريقة لا تؤدي إلى التصادم مع المستخدمين المسيئين، مرة أخرى لا تواجه برامج التطبيقات مثل هذه التحديات.

خامساً- يعيش نظام التشغيل لفترة طويلة، فقد أصبح عمر UNIX حوالي ثلث قرن، وأصبح عمر وندوز يزيد عن عقد كامل ولم يبدأ في الانقراض. وبالنتيجة على المصممين أن يفكروا كيف سيتغير العتاد الصلب والتطبيقات في المستقبل البعيد وكيف يجب الإعداد لها. النظم التي تتقيد بنظرة قريبة جداً إلى العالم عادة ما تموت.

سادساً- لا يعرف مصممو نظم التشغيل كيف ستستخدم نظمهم بشكل حقيقي. لذا يتوجب عليهم تقديم المزيد من العمومية. لم يصمم أي من أو وفي البال فكرة عن البريد الإلكتروني أو مستعرضات الإنترنت. بل توجد العديد من الحواسب التي تعمل أقل من ذلك.لا أحد يطلب من مصمم سفينة أن يصمم سفينة بدون أن يحدد هل هي سفينة صيد أم سفينة نقل أم سفينة حربية، أو حتى يغير رأيه بعد انتهاء التصميم.

سابعاً- تصمم عادة نظم التشغيل الحديثة لتكون قابلة للحمل، أي أن تكون قابلة للتشغيل على منصات عتاد صلب مختلفة. ويجب عليها أن تدعم المئات وحتى الآلاف من أجهزة الدخل / الخرج وكلها مصمم بشكل مستقل بغض النظر عن الآخر.

وكمثال على أن هذا الاختلاف قد يسبب مشاكل هو الحاجة إلى نظام تشغيل يعمل على حاسب صغير وحاسب آخر كبير بنفس الوقت. المثال الثاني يمكن رؤيته في نظام التشغيل دوس ، فعندما يحاول مستخدم ما، تركيب بطاقة صوت مثلاً، وبطاقة مودم يستخدمان نفس منافذ الدخل / الخرج أو نفس خطوط طلب المقاطعة، يجب على العديد من البرامج بالإضافة إلى نظام التشغيل التعامل مع تحديد المشاكل الناتجة عن التضارب في العتاد الصلب.

ثامناً وأخيراً- الحاجة الملحة لتوافقية النظام مع بعض الأنظمة التي سبقته، قد يكون للنظام محدوديات في طول الكلمات أو أسماء الملفات أو أي اعتبارات ينظر إليها المصممون الآن على أنها مهملة ولكنهم ينشغلون بها وينهمكون في حلها فيما بعد. وهو أمر مشابه لتحويل معمل إلى إنتاج سيارات العام القادم عوضاً عن سيارات العام ا لحالي مع الاستمرار بإنتاج سيارات هذا العام وبطاقة إنتاجية كاملة.

18 3 -تصميم الواجهة

يجب أن يكون واضحاً من البداية أن كتابة نظام تشغيل حديث ليست أمراً سهلاً. ولكن عند البدء بواحد فمن المحتمل أن أفضل مكان للبدء هو التفكير بالواجهة التي يقدمها. يقدم نظام التشغيل مجموعة من الخدمات، وأنواع المعطيات (كالملفات)، وعمليات عليها (كالقراءة)، تشكل هذه الأمور معاً واجهة النظام للمستخدم. لاحظ في هذا السياق أن مستخدمي نظام التشغيل هم مبرمجون يكتبون شفرة تستخدم استدعاءات النظام وليسوا أشخاصاً يشغلون برامج تطبيقات. لنظام التشغيل واجهات أخرى بالإضافة إلى واجهة النظام الرئيسية. مثلاً قد يحتاج بعض المبرمجين لكتابة برامج تشغيل الأجهزة لإدارتها في نظام التشغيل. ترى هذه البرامج مزايا محددة وتجري استدعاءات إجراءات محددة. وتعرف هذه المزايا والاستدعاءات واجهة مختلفة جداً عن تلك التي يراها مبرمجو برامج التطبيقات.يجب تصميم جميع هذه الواجهات بحذر إذا كان النظام سينجح.

18 4 -المبادئ الإرشادية

هل توجد مبادئ يمكن أن ترشد في تصميم الواجهة ؟ نحن نعتقد بأنه توجد بعض المبادئ وباختصار شديد هي البساطة والكمال وإمكانية تطبيقها بفعالية.

18 4 1 -المبدأ 1 البساطة

من السهل فهم وتطبيق واجهة بسيطة بطريقة خالية من الأخطاء. يجب على جميع المصممين أن يتذكروا هذه المقولة للكاتب الفرنسي Antoine de st Exupery.لا يكون الوصول للكمال عندما لا يوجد شيء يمكن إضافته وإنما عندما لا يوجد شيء يمكن حذفه.يقول هذا المبدأ أن القليل الكافي خير من الكثير الزائد، على الأقل في نظم التشغيل.

18 4 2 -المبدأ 2 الكمال

يجب أن نجعل الواجهة تسمح للمستخدم بفعل أي شيء يريده. وهذا يعني أنها يجب أن تكون كاملة. وهذا يقودنا إلى مقولة لألبرت اينشتاين وهي يجب أن يكون كل شيء بسيطاً قدر الإمكان ولكن ليس أبسط من ذلك.وبمعنى أخر، يجب أن يقدم نظام التشغيل المطلوب منه ليس أكثر. إذا احتاج المستخدمون لتخزين البيانات، فعليه أن يقدم آلية لتخزين البيانات. إذا احتاج المستخدمين للاتصال ببعضهم، يجب أن يقدم النظام آلية للاتصال، وهكذا.يجب أن يقوم كل تابع أو ميزة أو استدعاء للنظام بالعمل المطلوب منه. أي يجب أن يفعل شيئاً واحداً ولكن بشكل جيد. فإذا اقترح أحد أعضاء فريق التصميم توسيع استدعاء للنظام أو إضافة ميزة جديدة، فعلى الباقين أن يتساءلوا، هل سيحدث شيء شنيع إذا لم نفعل، فإذا كان الجواب لا ولكن ربما يجد شخص ما هذا الأمر مفيداً يوماً ما، ضعه في مكتبة ضمن مستوى المستخدم لا في نظام التشغيل حتى ولو كان أبطأ بهذه الطريقة. فليس من المطلوب من كل ميزة أن تكون أسرع من الرصاصة. فالهدف هو تقديم أقل ما يمكن من الآليات.

لنقدم الآن مثالين من خبرتنا الشخصية وهما MINIX و Amoeba يملك MINIX ثلاثة استدعاءات لجميع الأهداف والنوايا وهي Send وReceive وSendrec. بني النظام كمجموعة من العمليات مع إدارة ذاكرة ونظام ملفات. وكل برنامج تشغيل جهاز هو عملية قابلة للجدولة بشكل منفصل.المقاربة الأولى هي أن كل ما تفعله النواة هي جدولة العمليات والتعامل مع تمرير الوسائل بينها. وبالنتيجة، سيحتاج إلى استدعائي نظام فقط وهما Send لإرسال رسالة و receive لاستقبال رسالة أخرى.

الاستدعاء الثالث وهوsendrec هو ببساطة عملية أمثلة من أجل أسباب تتعلق بالكفاءة للسماح لإرسال رسالة وطلب الرد بخطوة واحدة من النواة. وكل شيء آخر ينجز بطلب عمليات أخرى (مثل عملية نظام الملفات أو برنامج تشغيل محرك الأقراص)، لإنجاز العمل.

إن Amoeba أبسط من ذلك، فلديه استدعاء نظام واحد وهو إنجاز استدعاء إجراء بعيد. يرسل هذا الاستدعاء رسالة وينتظر الرد. وهو مشابه بشكل أساسي للاستدعاء sendrec الموجود في MINIX وكل شيء آخر بني على هذا الاستدعاء.

18 4 3 -المبدأ 3 الكفاءة

المبدأ الثالث هو كفاءة التحقيق. فإذا لم يكن من الممكن تحقيق ميزة أو استدعاء بشكل فعال، فقد لا تستحق أن توجد. ويجب أن يكون واضحاً وبشكل يكون بديهياً للمبرمج كم سيكلف كل استدعاء نظام مثلاً، يتوقع مبرمجو UNIX أن يكون الاستدعاء Iseek أرخص من الاستدعاء read لأن الأول يغير مؤشراً في الذاكرة فقط، في حين أن الآخر يتطلب عمليات دخل / خرج للقرص. فإذا كانت التكاليف المحسوبة خاطئة، لن يتمكن المبرمجون من كتابة برامج فعالة.

18 5 -إدارة المشروع

إن المبرمجين دائمو التفاؤل و يعتقد معظمهم أن طريقة كتابة البرامج هي المضي إلى لوحة المفاتيح و البدء في الكتابة و بعد ذلك بقليل نحصل على برنامج منقح و مكتمل. بالنسبة للبرامج الكبيرة فالأمر ليس كذلك أبدا و سنتحدث في الفقرات التالية عن إدارة مشاريع البرامج الكبيرة و خصوصا مشاريع نظم التشغيل الضخمة.

18 5 1 -أسطورة رجل / شهر

لا تستغرب فالعنوان قادم من كتاب Fred Brooks أحد مصممي النظام OS/360، أثناء إجابته على السؤال لماذا من الصعب جداً بناء نظام تشغيل كبير ؟ عندما يسمع معظم المبرمجين مقولته التي تنص على أنه يستطيع المبرمج أن ينتج 1000 سطر فقط من الشفرة المنقحة في السنة في المشاريع الضخمة، سيتساءلون فيما إذا كان يعيش في الفضاء الخارجي، وسيتذكر معظمهم أنهم قد تمكنوا ذات ليلة من كتابة 1000 سطر وسيتساءلون كيف يمكن أن يكون هذا الرقم هو الناتج السنوي لأي شخص ذي معدل ذكاء أكبر من 50 ؟.

ما أشار إليه Brooks هو أن المشاريع الضخمة التي تحوي المئات من المبرمجين، مختلفة تماماً عن المشاريع الصغيرة، وأن النتائج التي يمكن الحصول عليها من المشاريع الصغيرة لا يمكن مقارنتها بالمشاريع الضخمة. في المشاريع الضخمة يستهلك مقدار هائل من الوقت في التخطيط لكيفية تقسيم العمل إلى وحدات. وتحديد الوحدات النمطية وواجهاتها بحذر، ومحاولة تخيل كيف ستتفاعل الوحدات النمطية، حتى قبل أن تبدأ البرمجة. ثم يجب برمجة الوحدات النمطية وتنميتها بشكل منعزل. أخيراً، يجب تجميع الوحدات النمطية والنظام بأكمله وفحصه. الحالة العادية هي أن تعمل كل وحدة نمطية بشكل جيد عند اختبارها لوحدها. ولكن النظام ينهار فوراً عندما تجمع كل الأجزاء مع بعضها. قدر Brooks الزمن كما يلي:

- 1/3 للتخطيط
- 1/6 للبرمجة
- 1/4 اختبار الوحدات النمطية
- 1/4 اختبار النظام

وبعبارة أخرى فإن البرمجة هي الجزء الأسهل، الجزء الأصعب هو اكتشاف ماذا ستكون الوحدات النمطية وجعل الوحدة النمطية A تتفاعل مع الوحدة النمطية B بشكل صحيح في البرامج الصغيرة التي تكتب بواسطة مبرمج واحد، فإن كل ما تحدثنا عنه أمر سهل.

جاء عنوان كتاب Brooks من إصراره على أنه لا يمكن للناس والوقت أن يحل أحدهما محل الآخر، إذ لا توجد وحدة قياس تسمى رجل / شهر. إذا كان مشروع ما يتطلب 15

شخصاً وسنتين لبنائه، فإنه من غير الصحيح القول أنه يمكن أن يقوم 360 شخصاً بإنجازه في شهر وربما لن يكون من الممكن إنجازه بواسطة 60 شخصاً في 6 أشهر.

توجد ثلاثة أسباب لهذا التأثير. أولاً، لا يمكن موازاة العمل بشكل كامل. لا يمكن البدء بأي برمجة حتى يتم التخطيط وتحديد الوحدات النمطية المطلوبة وكيف ستكون واجهاتها. ففي مشروع يستغرق سنتين، قد يستغرق التخطيط لوحده ثمانية أشهر.ثانياً، للانتفاع الكامل من عدد كبير من المبرمجين، يجب تقسيم العمل إلى عدد كبير من الوحدات النمطية بحيث يوجد لكل شخص عمل ينجزه. طالما أنه يمكن لكل وحدة نمطية أن تتفاعل مع كل الوحدات النمطية الأخرى، فإن عدد التفاعلات بين وحدة نمطية ووحدة نمطية أخرى التي تجب مراعاتها يكبر ليصبح مربع عدد الوحدات النمطية، أي يحتاج إلى مربع عدد المبرمجين.يخرج هذا التعقيد بسرعة عن السيطرة. أكدت القياسات الدقيقة حول 63 برنامجاً أن العلاقة بين الأشخاص والأشهر لهذه للبرامج بعيدة عن الخطية في المشاريع الضخمة.ثالثاً، التنقيح هو أمر تسلسلي إلى حد كبير، فعند تعيين عشرة منقحين لحل مشكلة لا يعني أنهم سيجدونها بسرعة أكبر بعشر مرات من سرعة منقح واحد، وقد يكونوا أبطأ، لأنهم سيهدرون وقتاً كثيراً في التحدث إلى بعضهم البعض.

جمع Brooks خبرته في مشكلة الزمن والأشخاص بقانون Brooks الذي يقول: إن إضافة قوى بشرية إلى مشروع برمجي متأخر تجعله يتأخر أكثر. المشكلة في إضافة الأشخاص، أنه يجب تدريبهم على المشروع، ويجب تقسيم الوحدات النمطية من جديد لتطابق العدد الكبير من المبرمجين المتوفرين حالياً. سيتطلب الأمر العديد من الاجتماعات لتنسيق جميع الجهود وما شابه.

أثبت Madnick هذا القانون بالتجريب، ولكنهم صاغوه بشكل يختلف قليلاً عن قانون بالمقولة التالية: تستغرق أي امرأة تسعة أشهر لحمل طفل واحد بغض النظر عن عدد النساء المكلفين بالعمل.

18 5 2 -هيكلية الفريق

نظم التشغيل التجارية هي مشاريع برمجية ضخمة وتتطلب فرقاً ضخمة من الأشخاص. وتعتبر نوعية الأشخاص مهمة بشكل هائل. لقد عرف لعقود أن لمبرمجي القمة إنتاجية تساوي عشرة أضعاف إنتاجية المبرمجين السيئين. المشكلة هي أنه عندما نحتاج إلى 200 مبرمج، فإنه من الصعب إيجاد 200 مبرمج ممتاز، بالتالي يجب عليك التنظيم مع طيف واسع من النوعيات. الأمر المهم الآخر في أي مشروع تصميم كبير سواء كان برمجياً أو غير ذلك، هي الحاجة إلى التماسك المعماري، يجب أن يوجد عقل واحد يتحكم بالتصميم. أخذ Brooks كاتدرائية Reims كمثال على المشاريع الضخمة التي تستغرق عقوداً للبناء. حيث تعاقب على المشروع معماريون متعددون وكل منهم حاول إضفاء طابعه الشخصي على المشروع، النتيجة هي تماسك معماري غير مطابق لأي كاتدرائية أوروبية.

في عام 1970، دمج Harlan Mills الملاحظة التي تقول أن بعض المبرمجين أفضل من غيرهم مع الحاجة للتماسك المعماري ضمن اقتراح سماه نموذج طاقم المبرمجين الرؤساء. تقوم فكرته على تنظيم فريق البرمجة مثل فريق الجراحة عوضاً عن كونه مشابهاً لفريق الجزارين. فعوضاً عن أن يقوم كل فرد بالتصرف بمفرده كمجنون، يجب أن يوجد شخص واحد يحمل المشرط. وعلى الباقين أن يقدموا الدعم. فمثلاً اقترح هيكلية الفريق المبينة في الشكل

لفريق مكون من عشرة أشخاص.مضت ثلاثة عقود منذ ظهور هذه الفكرة وتطبيقها. تغيرت بعض الأمور ولكن مازالت الحاجة إلى وجود عقل واحد يتحكم بالتصميم صحيحة.

يجب أن يكون هذا العقل قادراً على العمل 100 % في التصميم والبرمجة، مع وجود الحاجة إلى طاقم الدعم، ومع وجود مساعدة الحاسب فقد يكون طاقم أصغر كافياً، ولكن الفكرة في جوهرها مازالت صالحة. يحتاج أي مشروع ضخم إلى تنظيمه بشكل هرمي، في المستوى السفلي توجد عدة فرق صغيرة، وكل منها يرأسه مبرمج رئيس، يوجد في المستوى التالي مجموعات من الفرق التي يجب إدارتها بواسطة مدير، تظهر الميزة أن إدارة كل شخص تستهلك 10 % من وقتك، لذا فإن كل مجموعة مكونة من 10 أشخاص تحتاج إلى مدير يعمل بدوام كامل. يجب أيضاً إدارة هؤلاء المديرين، وهكذا صعوداً في الشجرة.لاحظ Brooks أن الأخبار السيئة لا تصعد في الشجرة بشكل جيد. دعى Jerry Saltzer هذه الظاهرة بمقوم الأخبار السيئة.

لا يريد أي مبرمج رئيس أو مدير أن يقول لرئيسه أن المشروع تأخر أربعة أشهر ولا توجد لديه فرصة لإنجاز المشروع قبل موعده النهائي لأنه يوجد تقليد عمره 1000 سنة ينص على قتل حاملي الأخبار السيئة.وكنتيجة فإن الإدارة العليا تكون بمعزل عن حالة المشروع. عندما يصبح واضحاً أنه لا يمكن إنجاز العمل عند الموعد النهائي، تستجيب الإدارة العليا بإضافة المزيد من الأشخاص وهو ما يرفضه قانون Brooks.

في الواقع فإن الشركات الكبيرة والتي تملك خبرة طويلة في إنتاج البرامج وتعرف ماذا سيحصل إذا أنتجت بشكل عشوائي، فإنها تميل إلى العمل بشكل صحيح وعلى العكس فإن الشركات الأحدث والأصغر والتي تكون مندفعة للدخول إلى السوق، لا تهتم دائماً بإنتاج برامجها بعناية. يؤدي هذا التعجل إلى الابتعاد عن النتائج الأمثلية.

لم يشهد أي من Brooks أوMill حركة المصادر المفتوحة، وبالرغم من النجاحات التي حققتها، فإنها مازالت تسعى إلى إيجاد وسيلة لإنتاج كميات كبيرة من البرمجيات الجيدة بالأسلوب القديم.الأمر الملحوظ هو أن المشاريع البرمجية ذات المصادر المفتوحة التي تحقق أكبر النجاحات، استخدمت بوضوح نموذج المبرمج الرئيسي أي وجود عقل واحد يتحكم بالتصميم المعماري (مثلاً Linus Torvalds في حالة نظام Linux).

18 5 3 -دور الخبرة

وجود مبرمجين خبراء هو أمر حساس لأي مشروع نظام تشغيل. وأشار Brooks إلى أن معظم الأخطاء لا تكون في الشفرة وإنما تكون في التصميم. يفعل المبرمجون ما يطلب منهم بشكل صحيح. وقد يكون الخطأ في الأمر المطلوب فعله. لا يوجد أي نوع من الاختيارات التي تكتشف المتطلبات الخاطئة. الحل الذي اقترحه Brooks يقوم على التخلي عن نموذج التطوير التقليدي المبين في الشكل a واستخدام النموذج المبين في الشكل b . الفكرة هي كتابة البرنامج الرئيسي أولاً والذي يستدعي إجراءات المستوى الأعلى.

في اليوم الأول للمشروع يمكن ترجمة النظام وتشغيله حتى لو لم يقم بأي شيء. مع مرور الزمن تدرج الوحدات النمطية في النظام الكامل.نتيجة هذه الطريقة أنه ينجز باستمرار اختبار تكامل النظام، وبالتالي تظهر الأخطاء في التصميم بشكل أبكر بكثير. وبالنتيجة فإن عملية التعليم التي يسببها التصميم السيء ستبدأ بشكل أبكر في الحلقة.القليل من المعرفة أمر خطير.

لاحظ Brooks ما دعاه باسم تأثير النظام الثاني غالباً ما يكون المنتج الأول من قبل فريق التصميم أصفريا بسبب خوف المصممين من ألا يعمل النظام أبداً. وبالتالي سيكونون حذرين جداً في وضع مزايا عديدة فيه. إذا نجح المشروع، فإنهم سيبنون النظام التالي وبتأثير من نجاحهم سيضعون في المرة الثانية كل المحسنات التي تركوها في المرة الأولى.وكنتيجة فإن النظام الثاني سيكون منتفخاً ويعمل بشكل ضعيف، في المرة الثالثة سيصممون بتأثير من فشل النظام الثاني وسيصبحون حذرين مرة أخرى.

يعتبر الزوج CTSS-MULTICS مثالاً على الحالة السابقة، كان CTSS أول نظام عام للتشارك الزمني وكان نجاحاً هائلاً بالرغم من محدودية وظائفه. كان النظام التالي طموحاً جداً وعانى كثيراً. كانت الأفكار جيدة ولكن كانت توجد أشياء عديدة جديدة وبالتالي عمل النظام بضعف لسنوات ولم يكن أبداً نجاحاً تجارياً كبيراً.

النظام الثالث في خط التطوير هذا هو UNIX كان أكثر حذراً وأكثر نجاحاً.

18 6 -اتجاهات تصميم نظم التشغيل

التنبؤ أمر صعب جداً خصوصاً فيما يتعلق بالمستقبل، فمثلاً ادعى Charles H Duell عام 1899، أن كل ما يمكن اختراعه قد تم اختراعه، ولكن مع ذلك ظهرت منذ ذلك الحين الكثير من الاختراعات بدءاً من مصباح أديسون إلى مكوك الفضاء. لذا ندعوك إلى شحن خيالك والمضي معنا في التكهن بالاتجاهات المستقبلية لنظم التشغيل.

18 6 1- نظم التشغيل ذات مساحات العنونة الضخمة

مع انتقال الحواسب من حيز العنوان ذي 32 بت إلى حيز العنوان 64 بت، أصبحت النقلات الرئيسية في تصميم نظم التشغيل ممكنة. حيز العنوان ذو 32 بت ليس كبيراً بالقدر المطلوب.فإذا أردت تقسيم 2 بايت على سكان الأرض بحيث تعطي كل شخص بايتاً واحداً، فلن يكفيك هذا الحيز. وعلى العكس فإن يساوي تقريباً وبالتالي ستصبح حصة كل شخص 3 جيغا بايت.يمكن إنشاء الكائنات بمعدل 100 ميغا بايت / ثا لمدة 5000 سنة، قبل أن نستهلك حيز العناوين بشكل كامل، سنحتاج إلى قرص تخزين كبير جداً لتخزين هذا القدر من البيانات، ولكن لأول مرة في تاريخ صناعة الحواسب، سيصبح الحد هو مساحة القرص وليس حيز العناوين.

مع وجود عدد كبير من الكائنات في حيز العناوين. سيصبح من المثير السماح لهذه العمليات بالعمل في حيز العناوين نفسه وبنفس الوقت لمشاركة البيانات بالطريقة العامة. سيقود مثل هذا التصميم إلى نظام تشغيل مختلف عن الذي نملكه حالياً.

إحدى مسائل نظام التشغيل التي يجب إعادة التفكير بها مع عناوين بطول 64 بت هي الذاكرة الظاهرية. مع حيز عناوين ظاهري قدره بايت وحجم صفحة مساو 8 كيلو بايت سيصبح لدينا صفحة. لا تعتبر جداول الصفحات التقليدية مناسبة لهذا الحجم، وبالتالي سنحتاج إلى شيء آخر، إحدى الإمكانيات هي جداول الصفحات المقلوبة ولكن توجد العديد من الأفكار الأخرى المقترحة.

18 6 2- التشبيك

صممت نظم التشغيل الحالة للحواسب المستقلة. كان التشبيك في الدرجة الثانية وكان الولوج إلى حواسب الشبكة يتم بواسطة برامج خاصة، مثل مستعرضات الويب أوFTP أو Telnet. في المستقبل قد يكون التشبيك أساس جميع نظم التشغيل. سيصبح الحاسب المستقل الذي لا يحوي اتصال شبكة أمراً نادراً مثل الهاتف غير المتصل بالشبكة، وستصبح سرعة الاتصال بعدة جيغا بت / ثا أمراً عادياً.

يجب أن تتغير نظم التشغيل لكي تلاءم نموذج الانتقال هذا. قد يكون الفارق بين البيانات المحلية والبيانات البعيدة ضبابياً لدرجة أنه لن يعرف أو يهتم أي شخص بمكان تخزين البيانات. يمكن للحواسب في أي مكان أن تعالج البيانات الموجودة في أي مكان كبيانات محلية. يعتبر هذا الأمر صحيحاً بشكل محدود في حالة نظام الملفات ولكنه سيصبح أكثر تطوراً وأكثر تكاملاً.

18 6 3 -النظم التفرعية والموزعة

أحد المجالات المتصاعدة هو النظم التفرعية والموزعة. نظم التشغيل الحالية للمعالجات المتعددة والحواسب المتعددة هي مجرد نظم تشغيل الحواسب وحيدة المعالج مع بعض التعديلات على المجدول لمعالجة التوازي بشكل أفضل. قد نرى في المستقبل نظم تشغيل يكون التوازي فيها أكثر مركزية من الوضع الحالي.

سيثار هذا التأثير بشكل كبير إذا كان للحواسب الحالية معالجان أو أربعة أو أكثر مع إعدادات متعددة المعالجات. قد يقودنا هذا إلى العديد من البرامج المصممة للمعالجات المتعددة مع الطلب الحالي لدعم أفضل من نظام التشغيل.تكاد النظم المتعددة المعالجات تهيمن على الحواسب العملاقة العلمية والهندسية في السنوات القادمة. ولكن نظام التشغيل الخاص بها ما زال بدائياً، يحتاج توازن الحمل وتوضع العمليات والاتصالات إلى الكثير من العمل.

بنيت معظم النظم الحالية كعتاد وسيط لأن نظم التشغيل الموجودة لا تقدم التسهيلات المناسبة للتطبيقات الموزعة. قد تصمم نظم التشغيل المستقبلية مع أخذ النظم الموزعة بعين الاعتبار، وبالتالي ستصبح جميع المزايا المطلوبة موجودة في نظام التشغيل منذ البداية.

18 6 4 -الوسائط المتعددة

نظم الوسائط المتعددة هي النجم الصاعد بوضوح في عالم الحواسب. لن يفاجأ أحد إذا دمجت الحواسب والهواتف وأجهزة التلفاز وأجهزة تشغيل الأشرطة الصوتية في جهاز واحد قادر على دمج الصور الثابتة العالية الدقة والصوت والفيديو، وموصول إلى شبكة عالية السرعة بحيث يمكن تحميل ملفات الوسائط المتعددة بسهولة، وتبادلها والوصول إليها عن بعد. سيكون نظام تشغيل هذه الأجهزة أو حتى أجهزة الصوت والفيديو مختلف بشكل جوهري عن ذلك الحالي.

سنحتاج بالتحديد إلى ضمانات الزمن الحقيقي وهذه ستقود تصميم النظام. سيصبح المستهلكون أكثر تذمراً من الإخفاقات المتكررة لأجهزة التلفاز الرقمية، وهكذا ستظهر الحاجة لبرامج أفضل وأقل خطأ. كذلك تميل ملفات الوسائط المتعددة لتصبح أطول وبالتالي يجب تعديل نظم الملفات لتصبح قادرة على معالجتها بكفاءة.

18 6 5 -الحواسب المغذاة بواسطة المدخرة

ستصبح الحواسب القوية ذات حيز العناوين بعرض 64 بت وتشبيك عريض الحزمة ومعالجات متعددة وصوت وفيديو عالي النوعية بلا شك شيئاً مألوفاً. ويجب أن تكون نظم تشغيلها مختلفة إلى حد كبير عن تلك الحالية لتعالج هذه المتطلبات. على أي حال يوجد جزء أكبر من السوق المتنامية وهو الحواسب المغذاة بواسطة المدخرات، وتتضمن الحواسب المحمولة والكفية وألواح الويب وبعض الهواتف الهجينة، سيكون لبعض منها وصلات لا سلكية بالعالم الخارجي وسيعمل بعضها بدون اتصال عندما نكون خارج المنزل. سنحتاج إلى نظم تشغيل مختلفة أصغر وأسرع وأكثر مرونة وأكثر وثوقية من تلك الحالية. قد تتشكل هذه النظم من أنواع مختلفة من الأنوية الصغيرة والنظم القابلة للتوسع.

يجب أن تتعامل نظم التشغيل هذه مع حواسب موصولة بشكل كامل (مشبكة سلكياً مثلاً) أو موصولة بشكل ضعيف (الوصل اللاسلكي) وعمليات بدون اتصال، تتضمن تخزين البيانات قبل قطع الاتصال، وتواصل دقيق عند إعادة الاتصال بشكل أفضل من النظم الحالية. يجب عليها أن تعالج مشاكل التنقل أكثر من النظم الحالية (مثلاً، إيجاد طابعة ليزرية والولوج إليها وإرسال ملف إليها بواسطة الأمواج الراديوية). تتضمن إدارة الطاقة حواراً واسعاً بين نظام التشغيل والتطبيقات عن المقدار المتبقي من البطارية وكيفية استخدامها بشكل أفضل ويجب أن يكون هذا الأمر أساسياً. أخيراً يجب وجود نظم دخل وخرج جديدة تتضمن المصافحة والمحادثة والتي قد تتطلب تقنيات في نظام التشغيل لتحسين النوعية.

18 6 6 -النظم المضمنة

أحد الاتجاهات التي ستتوسع باتجاهها نظم التشغيل هو النظم المضمنة. ستختلف نظم تشغيل الغسالات الآلية والأفران الميكروية وأجهزة المذياع ومسجلات الفيديو والمصاعد عن كل ما سبق وعن بعضها البعض. وسيعد كل واحد منها بحذر للتطبيق المحدد من أجله. فمن غير المعقول أن نقحم بطاقة PCI في جهاز قياس السرعة لتشغيله في ضابط المصعد. طالما أن النظم المضمنة تشغل عدداً محدوداً من البرامج المعروفة عند التصميم، فإنه من الممكن إجراء أمثلة ليست متوفرة في النظم العامة الأغراض.

الفكرة الواعدة بشأن النظم المضمنة هي نظم التشغيل القابلة للتوسع، يمكن أن تكون هذه النظم خفيفة أو ثقيلة حسب متطلبات التطبيق، ولكن بطريقة متناسقة لكل التطبيقات. وبما أن النظم المضمنة ستنتج بمئات الملايين، فإنها ستكون السوق الرئيسية لنظم التشغيل الجديدة.

18 7 -الخلاصة

يبدأ تصميم نظام التشغيل بتحديد ما يجب أن يفعله. يجب أن تكون الواجهة بسيطة وكاملة وفعالة. يجب أن يكون له نموذج واجهة مستخدم واضح ونموذج تنفيذ ونموذج بيانات.يجب أن يكون النظام مهيكلاً بشكل جيد، باستخدام إحدى التقنيات المعروفة مثل الطبقات المتعددة أو ملقم ـ عميل. يجب أن تكون المكونات الداخلية مستقلة إحصائياً عن بعضها البعض ويجب الفصل بين النهج والآلية بوضوح. يجب الاهتمام ببعض المسائل مثل بنى المعطيات الساكنة والحيوية والتسمية وزمن الربط وترتيب تحقيق الوحدات النمطية.

الأداء مهم ولكن يجب اختيار أسلوب الأمثلة بحذر لكي لا يؤدي إلى تدمير بنية النظام. تستحق مسائل التوازن بين الزمن والمساحة والتخزين المخبئ والتلميحات واستثمار المحلية وأمثلة الحالة العامة أن يبذل جهد ما لأجلها.

تختلف كتابة نظام بعدد قليل من الأشخاص عن إنتاج نظام كبير بواسطة 300 شخص. في تلك الحالة تلعب هيكلية الفريق وإدارة المشروع دوراً حساساً في نجاح أو فشل المشروع.

أخيراً، يجب أن تتغير نظم التشغيل في السنوات القادمة لتلاحق الاتجاهات الجديدة وتقابل التحديات الجديدة. قد تتضمن هذه الاتجاهات فضاءات عناوين ذات 64 بت والاتصالات على نطاق واسع والحواسب المكتبية المتعددة المعالجات والوسائط المتعددة والحواسب الكفية اللاسلكية بالإضافة إلى النظم المضمنة. ستكون السنوات القادمة أوقاتاً مثيرة لمصممي نظم التشغيل.

19 - استنتاجات ومقترحات حول المشروع.

الآفاق المستقبلية والتطوير:

كل شيء من صنع البشر يفتقر إلى الكمال الذي يريده هذا الكائن، وبما أن مشروعنا المتواضع هو من صنع البشر أيضا، فإننا أردنا أن ننوه هنا عن الأمور التي من الممكن السير فيها من قبل الذين يريدون التطوير المنتج الذي وصلنا إليه حتى الآن:

- تحقيق مفهوم الـ **Swapping** لتعامل بين الذاكرة والأقراص الصلبة، وذلك حتى يتم التعامل مع البرامج الضخمة التي لا تتسع الذاكرة الفيزيائية لها.
- إضافة خلق المجلد والملف إلى مكتبة الـ **EXT2**.
- إضافة مكتبات التعامل مع أنظمة الملفات **FAT32** والـ **VFS** لتسهيل تعامل المستخدم مع أنظمة الملفات المختلفة.
- إضافة إمكانية العمل في نمط الـ **Graphic Mode** وتعريبه.
- إدخال إمكانية تعامل المستخدم مع الفأرة، وتخديم أحداثها.
- بناء واجهة **Socket** لتسهيل تعامل المبرمج مع الشبكة.
- كتابة سواقة (**Driver**) للمودم ، وتضمين بروتوكول **PPP** لجعل النظام قادرا على العمل مع الانترنت.
- جعل النظام قادرا على العمل على معالجات أخرى غير معالج الـ **Intel**، بحيث يحقق نوعا من محمولية أكبر على مختلف أنواع المعالجات.

كلمة شكر

الآن و بعد خمس سنوات مفعمة غنية بالعلم ذاخرة بالمعرفة لا يسعنا إلا أن نتوجه بالشكر إلى تلك الشموع المضيئة التي احترقت لإبقاء شعلة العلم متقدة، والتي كانت ولا تزال تمهد لنا الطريق لنصل إلى مرحلة التحم فيها الطموح مع الواقع.

ونتوجه بشكر خاص إلى الدكتور عامر بوشي الذي تفضل مشكورا بالإشراف على هذا المشروع.

الباب . الملاحق

ملحق أ -متطلبات التطوير

إن عملية تطوير نظام التشغيل وبناءه هي مثلها مثل عملية تطوير أي برمجية أخرى تحتاج إلى بيئة ولغة برمجة مجموعة من أدوات الفحص و الترجمة والتنقيح, وكما الحال في البرمجيات فإننا غير محصورين بنوع معين من الأدوات وهنالك مجموعة من الخيارات لدينا للقيام بعملية تطوير نظام التشغيل وسنلقي في هذا الفصل نظرة سريعة على الأدوات التي اعتمدنا عليها في تطوير نظام الأقصى.

كما أسلفنا سابقا نحتاج لتطوير نظام التشغيل إلى:

- بيئة أو منصة عمل **Platform**.
- لغة برمجة **Programming Language**.
- مترجم **Compiler**.
- الرابط **Linker**
- أدوات مساعدة في تطوير النظام.
- منقح أو محاكي للتجربة و الاختبار **Emulator**.

سنأتي على كل عنصر من هذه العناصر لكي نوضح الحاجة والوظيفة إليها.

أ 1 -منصة العمل:

إن القيام بعملية بحث سريعة على الانترنت و في المراجع العملية التي تساعدك على بناء نظام تشغيل فإن أول نصيحة تقدم لك لتطوير نظام التشغيل هي استخدام نظام لينكس. فنظام التشغيل لينكس تم من خلالها تطوير أكبر عدد من نظم التشغيل. و مع أنه لا يوجد ارتباط بين تطوير نظام التشغيل ومنصة العمل إلا أن نظام لينكس فيه من الأدوات و التسهيلات التي تساعدك بشكل كبير على تطوير نظامك من خلال لينكس.

من المميزات التي ترجح كفة لنكس على غيره من أنظمة التشغيل هي انه في الدرجة الأولى نظام مفتوح المصدر ومجاني تحت رخصة GNU, و بالإضافة لذلك فإنه مضمن به ومتوافق تمام مع أقوى المترجمات للغة الـ C و الـ C++ و التي سنأتي عليها لاحقا. عدا عن ذلك كثرة اعتماد المراجع العلمية حول بناء أنظمة التشغيل من خلال نظام لينكس.

ونسخة نظام الليـنكس التي اعتمدناها في تطوير نظام تشغيلنا هي نسخة لينكس Red Hat 9.

أ 2 -لغة البرمجة:

أن معظم أنظمة التشغيل الحالية سواء منها نظم التشغيل الأكاديمية التي صممت بغرض الدراسة العلمية أو نظم التشغيل التجارية مكتوبة بلغة C بالإضافة إلى أن بعض أجزاء نظام التشغيل لا يمكن كتابتها إلى بلغة التجميع Assembly.

قديما في الأجيال الأولى لنظم التشغيل كانت اللغة المستخدم لبناء نظام التشغيل هي لغة التجميع وكان أول نسخة من نظام Unix قد تم كتابته كاملا بلغة التجميع ولكن بعض ظهور

لغات البرمجة عالية المستوى ولكن ارتفاع مستوى اللغة عن العتاد الصلب للحاسب ليس مناسبا في الكثير من الأحيان ولكن لغة الـ C التي تم ابتكارها في 1970 من تومبسون وريتشي كانت تمثل اللغة الهجينة أو الوسيطية بين لغة التجميع ولغة العالية المستوى. كانت تقدم تسهيلات كبيرة للتعامل السريع مع العتاد الصلب للحاسب من جهة ومن جهة أخرى كانت سهلة القراءة و التتبع من قبل المبرمجين, لذا تم الاعتماد عليها بشكل كبير لتطوير نظام التشغيل التي أحوج ما تحتاج إليه هي السرعة في التعامل مع العتاد.

و هنالك العديد من إصدارات لغة الـ C و التي يمكن استخدامها في تطوير, و لكن اللغة التي اعتمدنا لتطوير نظام تشغيل الأقصى هي اللغة المعيارية Ansi C standard. وهي من أقوى و أفضل إصدارات هذه اللغة وذلك لأنها لغة مجانية و مفتوحة المصدر و مرخصة تحت رخصة GNU. وبالإضافة لذلك فهي تتميز بالمحمولية ولنقل إنها ليست ميزة مقتصر عليها و لكن يمكن إضافة هذه الميزة إلى ميزات استخدام لغات البرمجة العالية المستوى في برمجة أنظمة التشغيل. المحمولية Portability تعني قدرة نقل برنامج المكتوب بهذه اللغة من حاسب لآخر بدون أي تعديلات أو إعادة لتوليد الشفرة. وهذا ما يحتاجه نظام التشغيل بشكل بديهي.

و كما ذكرنا في البداية أن هنالك بعض من أجزاء نظام التشغيل و التي تحتاج إلى السرعة في التعامل مع العتاد و التعامل مع مكدسات المعالج من مثل إدارة برامج تخديم المقاطعات وبرامج إدارة استدعاءات النظام وقسم الإقلاع فإننا نحتاج لكتابتها بلغة التجميع Assembly. وأيضا هنالك العديد من الإصدارات للغة التجميع و الإصدار المستخدم في نظامنا هي AT&T Assembly Standard وهي لغة معيارية أيضا متوافقة مع معالجات Intel 80x.

أ 3 ـ المترجم

مما سبق وجدنا أننا نحتاج للغة الـ C ولغة التجميع Assembly لتطوير و بناء النظام. و كما نعلم أن لغة البرمجة تحتاج إلى المترجم لتحويل النص البر مجي إلى ترميز لغة الآلة. ونحن نحتاج لمترجم للغة C و مترجم لغة التجميع.

مترجم لغة C :

كما سبق وذكرنا أن نظام تشغيل لينكس مضمن مع مترجم للغة C وهو من أقوى مترجمات هذه اللغة على الإطلاق و هو مترجم GCC, ويمكننا أن نقوم بتنصيب هذا المترجم على منصات عمل أخرى مثل وندوز باستخدام Cross Compiling إن كنت مصرا عزيزي القارئ على البقاء عند Windows فهذا متاح. ومترجم GCC هذا يتمتع بأنه أيضا مجاني و مفتوح المصدر ومرخص تحت رخصة GNU للمصادر المفتوحة.

إن الإصدار GCC3.2.2 هو الإصدار المستخدم في تطوير نظام الأقصى وهذا الإصدار مضمن مع نظام لينكس Redhat9 بشكل تلقائي و يمكنك تطوير هذا الإصدار من المترجم.

```
:\>gcc      flags  -o     output.o     input.s
```

مترجم لغة التجميع :

كما ذكر سابقا يوجد اجزاء من النظام لا يمكن كتابتها إلا بلغة التجميع, وبالتالي نحن بحاجة إلى مترجم لهذه اللغة إلى لغة الآلة. هنالك العديد من المترجمات المستخدمة في ترجمة هذه اللغة من مثل GNU as الذي هو من أحد اصدارات منظمة الـ GNU في مجموعة أدوات التطوير المضمنة مع نظام لينكس Binutils Packages. حيث يستطيع ترجمة شفرات لغة التجميع المكتوبة وفق لنحوية AT&T و Intel Syntax (يتم تحديد المعيار المعتمد من خلال ضبط أحد الخيارات في المترجم as). و في الحقيقة أن مترجم AS ليس مترجم أو مجمع للغة واحدة وإنما يستطع ترجمة مجموعة من اللغات وهذه اللغات هي اللغات المعتمدة من المعالجات المدعومة من قبل GCC. كما يقدم الـ AS تكاملية عالية مع الأدوات الأخرى المتاحة من قبل أدوات الـ GNU Tools من مثل Make و GCC Compiler.

```
:\>as flags  -o     output.o     input.s
```

أ 4 ـ أدوات الربط Linkers :

من الأدوات المطلوبة لإكمال دورة التطوير لنظام التشغيل هي برامج الربط, فنظام التشغيل ليس ملفا واحدا و إنما هو عبارة عن مجموعة من الملفات التي يتم ترجمتها في البداية من خلال المترجمات سابقة الذكر ومن ثم يتم تجميع الملفات الناتجة عن الترجمة في ملف واحد من خلال الرابط, Linker. و بالتالي مهمة الرابط الـ Linker هي:

تحديد صيغة الملف النهائي ومكان توضع وتحميل الشفرة والمعطيات و المكدس في الذاكرة
تحديد نقطة بداية الشفرة في الذاكرة Entry Point
الربط بين التوابع واستدعاءاتها الموجودة في ملفات مختلفة ونفس الأمر بالنسبة للمتحولات العامة.
كما يقوم بتجميع الملفات الناتجة عن الترجمة من أكثر من لغة.

يقدم لنا GNU مجمعين أحدهما :

LD : و هو أقوى المجمعات المتوفرة[2]. حيث يستطيع أخذ التعليمات من ملف دفعي Script File وباستخدام هذه الميزة يمكننا أن نقوم بتعريف متحولات يتم الوصول إليها و تعديلها من قبل البرنامج المراد ربطه.

AR : مجمع بسيط يقوم بتجميع الملفات الناتجة عن الترجمة obj files إلى ملف تنفيذي نهائي.

لتنفيذ برنامج الرابط LD يمكنك ذلك من خلال كتابة الأمر التالي بالشكل التالي من خلال محرر الأوامر في اللينكس Shell.

```
\:>ld  –Tscriptfile
            -o     output
            objfile1.o     objefile2.o……       objfileN.o
```

output : تمثل الملف الذي يتم تجميع الملفات فيه أي خرج LD

[2] موقع www.mega-tokyou.com/osfaq2.

objfilesN تمثل الملفات الناتجة عن عملية الترجمة و التي نريد تجميعها في ملف واحد

و بالنسبة للرابط AR :

```
\:>ar -rcs archive.a objfile1.o objefile2.o..... objfileN.o
```

output : تمثل الملف الذي يتم تجميع الملفات فيه أي خرج LD
objfilesN تمثل الملفات الناتجة عن عملية الترجمة و التي نريد تجميعها في ملف واحد

أ 5 -أدوات مساعدة للتطوير

لإكمال دورة تطوير النظام فهنالك مجموعة من الأدوات المساعدة وذلك لتساعدنا في عملية تنفيذ واختبار نظام التشغيل. هذه الأدوات هي :

أ 5 1- برنامج إنشاء صورة عن القرص الصلب Image Files

إن عمليات الاختبار لنظام التشغيل تتطلب منا ان يكون لدينا قطاع للإقلاع وجود قطاع للإقلاع يعني اننا بحاجة لأن يكون لدينا إما قرص فيزيائي أو قرص مرن للكتابة عليه ويفضل وجود القرص المرن وذلك لإن سعره أخفض وأقل تكلفة من القرص الصلب, ولكن أيضا عملية الكتابة على القرص المرن ومن ثم الإقلاع من خلاله تستغرق وقتا هذا بالإضافة إلى إمكانية حدوث تلف لكثرة الكتابة المباشرة على القرص, وعلى هذا الأساس فهنالك برامج تسمح لي بالتعامل مع محاكيات للقرص المرن. حيث فكرة هذه البرامج هي إنشاء ملف وإعطائه تنسيق القرص المرن و الكتابة على هذا الملف تماما كما لو أنه قرص مرن. ويتم استخدام بعض الأدوات لإنشاء هذه الصور مثل winimage في نظام ويندوز أو من خلال برنامج dd ضمن مجموعة mtools.

أ 5 2- برنامج الكتابة المباشرة على الأقراص أو الصور

و يستخدم لنسخ الملف و المجلدت التي نريد وضعها في صورة القرص المرن. فنحن اعتبرنا الصورة هي القرص المرن وسنتعامل معها كأنه قرص مرن,.و هنالك مجموعة من الأدوات التي تمكننا من ذلك مثل تعليمة mcopy و mformat في mtools.

أ 5 3- برنامج استعراض محتويات الأقراص التخزينية

و هي برامج تستخدم لقراءة المحتويات على الأقراص الفيزيائية وقراءة محتوى القطاعات و المسارات أي المحتويات الثنائية. مثل برنامج winhex في ويندوز. ويستخدم هذا البرنامج في عمليات فحص محتويات الأقراص لدى اختبار عمليات التعامل مع انظمة الملفات.

أ 6 -المحاكي

كما في كل البرمجيات فإن عملية تطوير البرمجية تحتاج لمرحلة الاختبار و التجربة و عملية الاختبار والتجربة بالنسبة لنظام التشغيل فإن عملية التجربة تحتاج إلى أدوات خاصة. فنظام التشغيل ليس برنامج تنفيذي يتم تشغيلها مباشرة ضمن نظام التشغيل الآخر و بالتالي نحن نحتاج إلى حاسب في كل مرة نحتاج فيها إلى تجربة النظام. و من الواضح أن عملية التجريب

بهذه الطريقة صعبة جدا ومضيعة للوقت. فتخيل معي أنك في كل مرة تريد اختبار نظامك بعد عملية ترجمة للنظام فإنك سوف تقوم بإطفاء الحاسب و تحميل نسخة عن النظام إلى القرص المرن والإقلاع عن النظام.

إن هذه العملية طويلة و مملة, و لذا هنالك برامج تحاكي عمل الحاسب تدعى بالمحاكيات وهنالك عدة برامج و أنواع من هذه المحاكيات مثل :

- **Virtual Machine**
- **VMWare**
- **Bochs**

و قد اعتمدنا في نظام تشغيلنا المحاكي Bochs. وذلك لأنه نظام مجاني ومفتوح المصدر وقد اعتمدنا الإصدار Boch 2.1.1. وكما أنه يمكن للمحاكي Bochs أن يعمل على كل من نظامي التشغيل لينكس و ويندوز.

للقيام بتنصيب المحاكي Bochs على نظام ويندوز فإنه فقط يتطلب ذلك أن تحصل على نسخة عن المحاكي[3], وتنفيذ نسخة install وكذلك بالنسبة لنظام لينكس.

أ 7 -سناريو للعمل على الأدوات :

بعد أن قمنا باستعراض الأدوات السابقة بشكل نظري سوف نقوم بتقديم مثال بسيط ونقوم من خلاله باستخدام هذه الادوات و ذلك فقط لتوضيح مدى أهمية هذه الأدوات ولتوضيح كيفية التكامل بين هذه الأدوات.

سنقوم في مثالنا هذا بكتابة برنامج بسيط يقوم بإقلاع الحاسب و كتابة جملة بسيطة على الشاشة. وذلك بالإستفادة مما سبق ذكره في فصل إقلاع.

الخطوة الأولى هي عملية كتابة النص البرمجي وسنقتصر هنا فقط على لغة الأسمبلي و بعد أن قمنا بكتابة البرنامج من ثم نقوم بترجمة الملف وكتابته على القرص المرن وتحويله لقرص إقلاع.

الآن نقوم بربط قرص الإقلاع بنظام الـ Bochs ويتم ذلك من خلال ملف config يحتوي على الخصائص التي يجب على المحاكي أن يأخذها. وهذه الخصائص هي على الشكل التالي :

[3] "http://bochs.sourceforge.net".

```
megs: 32
Romimage : file=/user/local/share/bios-bochs-latest, address=0xf0000
Vgaromimage : /user/vgabios-elpin-2.40
Floppya:1_44=/dev/fd0
Log:Bochs.txt
Clock: sync=realtime
```

السطر الأول : يمثل حجم الذاكرة التي سنحملها للمحاكي وهي هنا 32 ميغا
السطر الثاني : تمثل ملف صورة نظام الإدخال والإخراج وهو يكون ملف مرفق مع المحاكي
السطر الثالث : ايضا ملف يمثل ذكرة الخاصة ببطاقة الشاشة.
السطر الرابع : للربط المحاكي مع القرص المرن.
السطر الرابع و الخامس يمثل ملف الأخطاء التي تحصل خلال عمل المحاكي وتردد ساعة نظام المحاكي

نحفظ ملف config السابق تحت اسم معين ويفضل أن يكون bochsrc.txt الآن نقوم باستدعاء المحاكي من خلال shell بكتابة الأمر التالي :

```
:/>Bochs      bochsrc.tct
```

أ 8 ـ مصادر علمية مفيدة

إن عملية القيام بتطوير نظام التشغيل تتطلب منا الاطلاع على العديد من الكتب و المراجع و مواقع الانترنت التي تقدم لنا معلومات تفصيلية عن كيفية تصميم و بناء نظام التشغيل ومن هذه الكتب مثلا

اسم المرجع	المؤلف	الإصدار	السنة
Understanding the Linux Kernel	Daniel P. Bovet Marco Cesati	First Edition	October 2000 ISBN: 0-596-00002-2,
Modern Operating Systems (with pdf index).pdf	**Andrew Tanenbaum**	**second Edition**	1992
Operating Systems Design and Implementation.	**Andrew Tanenbaum**	**second Edition**	1992
مفاهيم نظم التشغيل	**الجمعة العلمية السورية للمعلوماتية**	**الإصدار الأول.**	2005
تصمتم وتنفيذ نظم التشغيل الحديثة	**يمان اللبني & أسامة العبدالله**	**الإصدار الأول**	2005 دار شعاع للنشر
The IA-32 Intel Architecture Software	**Intel**	Order Number 253665	**June 2005**

			Developer's Manual, Volume 1: Basic Architecture
June 2005	Order Numbers 253666 and 253667	**Intel**	The IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference.
June 2005	**Order Number: 253668-016**	**Intel**	IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide.
1996 دار شعاع للنشر.	**الطبعة الأولى**	**حيان السيد**	برمجة الحواسيب الشخصة بلغة التجميع
1997 جامعة حلب	**الطبعة الأولى**	**يحيى النجار**	كتاب المعالج المصغر
1995 ISBN 1-5575-304-1		**Micle Ticher & prun o jennrich**	PC Intern
			C with Assembly language

http://www.osdever.net
http://www.gnu.org
http://www.linuxchix.org/content/courses/kernel_hacking
http://www.mega-tokyo.com/osfaq2
http://members.lycos.co.uk/

ملحق ب - ملحق جداول الأسكي

ب .1 -الأسكي العربية

الجدول التالي يحتوي على شفرة الأسكي العربية التي قمنا بتحميلها, كل حرف يمكنك استخلاص شفرة الأسكي التابعة له من خلال تركيب العمود مع السطر الواقع فيه شكل الحرف مثلا الحرف "ب" شفرة الأسكي الخاصة به هي : 1E.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	-	-	-	-	-	-	-	ـا	ظ	ا	ظـ	ا	ظ	ا	ظ
1	-	-	-	-	-	-	-	-	ـب	ع	بـ	عـ	بـ	عـ	ب	ع
2	-	-	-	-	-	-	-	-	ـت	غ	تـ	غـ	تـ	غـ	ت	غ
3	-	-	-	-	-	-	-	-	ـث	ف	ثـ	فـ	ثـ	فـ	ث	ف
4	-	-	-	-	-	-	-	-	ـج	ق	جـ	قـ	جـ	قـ	ج	ق
5	-	-	-	-	-	-	-	-	ـح	ك	حـ	كـ	حـ	كـ	ح	ك
6	-	-	-	-	-	-	-	-	ـخ	ل	خـ	لـ	خـ	لـ	خ	ل
7	-	-	-	-	-	-	-	-	ـد	م	دـ	مـ	دـ	مـ	د	م
8	-	-	-	-	-	-	-	-	ـذ	ن	ذـ	نـ	ذـ	نـ	ذ	ن
9	-	-	-	-	-	-	-	-	ـر	ه	رـ	هـ	رـ	هـ	ر	ه
A	-	-	-	-	-	-	-	-	ـز	و	زـ	وـ	زـ	وـ	ز	و
B	-	-	-	-	-	-	-	-	ـس	ي	سـ	يـ	سـ	يـ	س	ي
C	-	-	-	-	-	-	-	-	ـش	ئ	شـ	ئـ	شـ	ئـ	ش	ئ
D	-	-	-	-	-	-	-	-	ـص	ة	صـ		صـ	ء	ص	ة
E	-	-	-	-	-	-	-	-	ـض	ى	ضـ		ضـ		ض	ى
F	-	-	-	-	-	-	-	-	ـط	ؤ	طـ		طـ		ط	ؤ

ب 2 ـ خريطة لوحة المفاتيح الانكليزية

```
word regular [128] = {
0x0000,0x011B,0x0231,0x0332,0x0433,0x0534,0x0635,0x0736,0x0837,0x0
938,0x0A39,0x0B30,0x0C2D,0x0D3D,0x0E08,0x0F09,

0x1071,0x1177,0x1265,0x1372,0x1474,0x1579,0x1675,0x1769,0x186F,0x1
970,0x1A5B,0x1B5D,0x000D,0x1D00,0x1E61,0x1F73,

0x2064,0x2166,0x2267,0x2368,0x246A,0x256B,0x266C,0x273B,0x2827,0x2
960,0x2A00,0x2B5C,0x2C7A,0x2D78,0x2E63,0x2F76,

0x3062,0x316E,0x326D,0x332C,0x342E,0x352F,0x3600,0x372A,0x3800,0x3
920,0x3A00,0x3B00,0x3C00,0x3D00,0x3E00,0x3F00,

0x4000,0x4100,0x4200,0x4300,0x4400,0x4500,0x4600,0x4700,0x48FE,0x4
900,0x4A2D,0x4BFD,0x4C00,0x4DFC,0x4E2B,0x4F00,

0x50FF,0x5100,0x5200,0x5300,0x5400,0x5500,0x5600,0x8500,0x8600,0x0
000,0x0000,0x5B00,0x5C00,0x5D00
;{

!// !//The keypad map.
static
byte keypad_char[] = {'7','8','9','-','4','5','6','+','1','2','3','0;{'.'

!//US keyboard keymap :: "with CTRL" keys.
static
word with_control[128} = [


0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0
000,0x0000,0x0000,0x0000,0x0000,0x0000,0x9400,

0x1011,0x1117,0x1205,0x1312,0x1414,0x1519,0x1615,0x1709,0x180F,0x1
910,0x0000,0x0000,0x1C0A,0x1D00,0x1E01,0x1F13,

0x2004,0x2106,0x2207,0x2308,0x240A,0x250B,0x260C,0x0000,0x0000,0x0
000,0x2A00,0x0000,0x2C1A,0x2D18,0x2E03,0x2F16,

0x3002,0x310E,0x320D,0x0000,0x0000,0x9500,0x3600,0x9600,0x3800,0x3
920,0x3A00,0x5E00,0x5F00,0x6000,0x6100,0x6200,

0x6300,0x6400,0x6500,0x6600,0x6700,0x4500,0x4600,0x7700,0x8D00,0x8
400,0x8E00,0x7300,0x8F00,0x7400,0x9000,0x7500,
0 x9100,0x7600,0x9200,0x9300,0x5400,0x5500,0x5600,0x8900,0x8A00
;{
```

```
US keyboard keymap :: "with SHIFT" keys.
static
word with_shift[128} = [

0x0000,0x011B,0x0221,0x0340,0x0423,0x0524,0x0625,0x075E,0x0826,0x0
92A,0x0A28,0x0B29,0x0C5F,0x0D2B,0x0E08,0x0F00,

0x1051,0x1157,0x1245,0x1352,0x1454,0x1559,0x1655,0x1749,0x184F,0x1
950,0x1A7B,0x1B7D,0x000D,0x1D00,0x1E41,0x1F53,

0x2044,0x2146,0x2247,0x2348,0x244A,0x254B,0x264C,0x273A,0x2822,0x2
97E,0x2A00,0x2B7C,0x2C5A,0x2D58,0x2E43,0x2F56,

0x3042,0x314E,0x324D,0x333C,0x343E,0x353F,0x3600,0x372A,0x3800,0x3
920,0x3A00,0x5400,0x5500,0x5600,0x5700,0x5800,

0x5900,0x5A00,0x5B00,0x5C00,0x5D00,0x4500,0x4600,0x4700,0x4800,0x4
900,0x4A2D,0x4B00,0x4C00,0x4D00,0x4E2B,0x4F00,

0x5000,0x5100,0x5200,0x5300,0x5400,0x5500,0x5600,0x8700,0x8800,0x0
000,0x0000,0x5B00,0x5C00,0x5D00
;{

!//US keyboard keymap :: "with ALT" keys.
static
word with_alt[128} = [

0x0000,0x0100,0x7800,0x7900,0x7A00,0x7B00,0x7C00,0x7D00,0x7E00,0x7
F00,0x8000,0x8100,0x8200,0x8300,0x0E00,0xA500,

0x1000,0x1100,0x1200,0x1300,0x1400,0x1500,0x1600,0x1700,0x1800,0x1
900,0x1A00,0x1B00,0x1C00,0x1D00,0x1E00,0x1F00,

0x2000,0x2100,0x2200,0x2300,0x2400,0x2500,0x2600,0x2700,0x2800,0x2
900,0x2A00,0x2B00,0x2C00,0x2D00,0x2E00,0x2F00,

0x3000,0x3100,0x3200,0x3300,0x3400,0x3500,0x3600,0x3700,0x3800,0x3
900,0x3A00,0x6800,0x6900,0x6A00,0x6B00,0x6C00,

0x6D00,0x6E00,0x6F00,0x7000,0x7100,0x4500,0x4600,0x9700,0x9800,0x9
900,0x4A00,0x9B00,0x9C00,0x9D00,0x4E00,0x9F00,
0 xA000,0xA100,0xA200,0xA300,0x5400,0x5500,0x5600,0x8B00,0x8C00
;{
```

ب 3 ـخريطة لوحة المفاتيح العربية

```
static
word ar_regular_cutlast[128} = [

0x0000,0x011B,0x0231,0x0332,0x0433,0x0534,0x0635,0x0736,0x0837,0x0
938,0x0A39,0x0B30,0x0C2D,0x0D3D,0x0E08,0x0F09,

0x10EE,0x11ED,0x12E3,0x13F4,0x14F3,0x15F2,0x16F1,0x17F9,0x18E6,0x19
E5,0x1AE4,0x1BE7,0x000D,0x1D00,0x1EEC,0x1FEB,

0x20FB,0x21E1,0x22F6,0x23E0,0x24E2,0x25F8,0x26F7,0x27F5,0x28EF,0x29
E8,0x2A00,0x2B5C,0x2CFC,0x2D78,0x2EFF,0x2FE9,

0x3062,0x31FE,0x32FD,0x33FA,0x34EA,0x35F0,0x3600,0x372A,0x3800,0x3
920,0x3A00,0x3B00,0x3C00,0x3D00,0x3E00,0x3F00,

0x4000,0x4100,0x4200,0x4300,0x4400,0x4500,0x4600,0x4700,0x4800,0x4
900,0x4A2D,0x4B00,0x4C00,0x4D00,0x4E2B,0x4F00,

0x5000,0x5100,0x5200,0x5300,0x5400,0x5500,0x5600,0x8500,0x8600,0x0
000,0x0000,0x5B00,0x5C00,0x5D00
;{
```

ملحق ج -الأشكال والجداول

ج 1- الأشكال

ج 2 -الجداول

ملحق د -المصطلحات العلمية :

A	
Abort	إخفاق
Access	وصول
Access	ولوج
Access time	زمن الوصول
Acknowledge	إشعار
Adapter	موائم
Address Space	فضاء ذاكري
Addressing	عنونة
Allocate	حجز
Application layer	طبقة التطبيقات
ASCII	أسكي
Assembly	أسمبلي
Attribute	سمة
B	
Batch	دفعية
BIOS	بيوس
Block	كتلة
Blocking	تجميع
Booting	إقلاع
Buffer	بفر
Bus	ممر
Busy Wait	انتظار مشغول
Byte	بايت
C	
Cache	ذاكرة مخبئية
Cache	كاش
Call	استدعاء
Char	محرف
Clock	ساعة
Code	شفرة
Command	أمر
Compile	ترجمة
Computer	حاسب
Console	كونسول
Contiguous Allocation	تخصيص متجاور
Controller	متحكم
Copy On Write	نسخ عند الكتابة
Counter	عداد
D	
Data	معطيات - بيانات

Data Link layer	طبقة ربط البيانات
Datagram	حزمة البيانات
Deadlock	حالة ميتة
Debug	تنقيح
Debug	منقح
Descriptor	واصف
Destination	مصدر
DOS	دوس
Drive	سواقة
Driver	مشغل

E

Embedded	مضمنة
Encapsulate	تغليف
Exception	استثناء
Executable	تنفيذية
Executable and linking format	صيغة الربط و التنفيذ

F

Fault	خطأ
File System	نظام ملفات
Flag	علم
Floppy Disk	قرص مرن
Flow	تدفق
Folder	فهرس
Format	تهيئة
Format	صيغة
Frame	إطار
Free	تحرير
Free	هدم

G

Gate	بوابة
Graphical Mode	نمط صوري
Graphical User Interface	واجهات المستخدم الرسومية

H

Handle	مقبض
Handler	برنامج خدمة
Hard Disk	قرص صلب
Hardware	عتاديات
Header	ترويسة
Heap	كومة

I

Implement	تنفيذ
Inode	عقدة
Input	دخل

Install	تنزيل
Install	تنصيب
Interface	واجهة
Interrupt	مقاطعة
K	
Kernel	نواة
Keyboard	لوحة المفاتيح
L	
Layered	طبقية
Level	مستوى
Liner	عنوان خطي
Link List Allocation	تخصيص باللائحة المترابطة
Linux	لينكس
Load	تحميل
Loader	محمل
Local	محلي
M	
Macro	ماكرو
Map	خريطة
MISCELLANEOUS REGISTER	المسجل المتنوع
Mode	نمط
Mount	ربط
Multiboot	تعدد الإقلاع
Multimedia	وسائط متعددة
Multiplexing	توزيع
Multitasking	تعددالمهام
N	
Netmask	قناع الشبكة
O	
Output	خرج
P	
Packet	رزمة
Paging	تصفيح
Parallel	تفرعية
Parameter	بارامتر
Partition	تجزئة
Path	مسار
Pixel	عنصورة
Pooling	استجواب
Port	منفذ
Privilege	سماحية
Procedure	إجرائية

Processes	عمليات
Prompt	محث
Protected	محمي
Protection	حماية
Protocol	بروتوكول
Pseudo Parallelism	توازي ظاهري

Q

Queue	رتل

R

Ram	رام
Real	حقيقي
Real Time	زمن حقيقي
Register	مسجل
Relocatable	التنفيذية
Remote	بعيد
Resetting	إعادة تهيئة
Resource	مورد
root	جذر
Router	موجه

S

Scan	مسح
Scheduler	مجدول
Section	مقطع
Sector	قطاع
Segment	قسم
Segment	مقطع
Segmentation	تقطيع
Segmenting	تقسيم
Selector	ناخب
Semaphore	قفل
Sequence	سلسلة
Servers	مخدمات
Session layer	طبقة الجلسة
Shell	محرر أوامر
Source	وجهة
Space	فضاء
Stack	تكديس
Static	ستاتيكية
Status	حالة
Super block	كتلة كبرى
Swap	تبديل
Synchronize	متزامنة

T

Terminal	طرفية
Text Mode	نمط نصي
The Second Extended File System	نظام الملفات الممتد الثاني
Threads	خيوط التنفيذ
Timer	المؤقت
Token	توكن
Transport layer	طبقة النقل
Transport layer	طيقة النقل
Trap	فخ
Trigger	القادح
Typematic	سرعة الاستجابة

U

Unicode	يونكود

V

VGA	طبقة العرض
Video RAM	ذاكرة فيديوية
Virtual	ظاهرية
Virtual Memory	ذاكرة افتراضية

Designing and Implementing an Operating System

Written by:

Ahmed maatoki – Rami Kasab- Fadi Amroush- Mustafa Kanawati – M.Sammer Srouji – Mohammed Nashed – Marwan Alrihawi.

Supervised by PHD. Ammer Boushi

June/2005

Submitted in partial fulfillment of the requirement of degree of Bachelor in the Department of

Network Department

Faculit of Informatics Engieering

Aleppo University

Aleppo – Syria