مقدمة إلى

برمجة التطبيقات الشبكية باستخدام Socket API

م. محمد العليان

@mhdalyan

الترخيص

للتواصل مع الكاتب

LinkedIn

about.me

Twitter

مقدمة

ازدادت الحاجة إلى التطبيقات الشبكية مع تطور الشبكات الحاسوبية، ومما لا شك فيه أن هذه التطبيقات حظيت وما زالت تحظى بإهتمام الكثير من الشركات، سيما الكبيرة منها وحتى الشركات الصغيرة باتت تحتاج هذا النوع من التطبيقات.
سنوضح بشكل عميق من خلال هذا الكتيب البسيط مبادئ بناء التطبيقات الشبكية وكيفية تحقيقها باستعمال لغتي البرمجة C# و Java، وسنرى أن الفرق بينهما لا يتعدى اختلاف في أسماء الصفوف(classes)، ما يهمنا هو المبدأ وليس لغة البرمجة بحد ذاتها.
سأكون مسروراً حقاً بملاحظاتكم على هذا الكتيب، وأرجو ألا تبخلو بها.
آمل من الله تعالى أن يكون هذا الكُتيب مفيداً لكم وأن يقدم العون إلى كل من يريد أن يتعلم مبادئ برمجة التطبيقات الشبكية، وأرجو أن يكون عملي هذا في صحيفة أعمالي، والله من وراء القصد.

دمشق في 16-5-2014

محمد العليان

جدول المحتويات

تمهيد 1
أنواع التطبيقات 1
البرامج الوسيطة Middleware 1
ملاحظات متفرقة 1
Client/Server Model 2
طريقة التخاطب بين تطبيقين 2
Client/Server in JAVA 3
شرح لآلية عمل البرنامجين 5
ما هي أنواع التطبيقات التي يمكن إنشاؤها عن طريق Socket 5
حالة دراسية مخدم ملفات يستخدم حوض نياسب(Thread Pool) 7
خطوات إنشاء تطبيق مخدم في C# 9
خطوات إنشاء تطبيق Client 10
المتطلبات اللاوظيفية 16
تطبيق مخدم متعدد النياسب 16

تمهيد

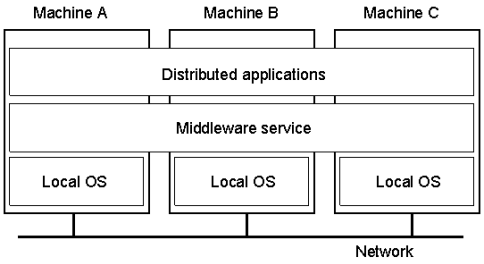
ظهرت الحاجة إلى التواصل بين المهام(processes) مع ظهور أنظمة التشغيل الحديثة، حيث ظهرمفهوم التواصل بين المهام (Inter-Process Communication) IPC. أحد وسائل التواصل بين المهام هو Socket API وهي التقنية التي سنتكلم عنها.

سنبدأ بشرح بعض المفاهيم الأساسية، ومن ثم سننتقل إلى طريقة تحقيقها لاحقاً باسخدام Java, #C.

أنواع التطبيقات

- تطبيقات شخصية (desktop)، و هي التطبيقات التي تعمل على نفس الحاسب.
- تطبيقات موزعة، و هي التطبيقات التي تعمل على الشبكة.

البرامج الوسيطة Middleware



هي مجموعة من البرمجيات الموزعة على كل آلة في النظام، و مهمتها تقديم واجهة للاتصال و التخاطب لأجزاء النظام التي تعمل على هذه الآلات.كما أنه يخفي تفاصيل الاتصال عبر الشبكة عن التطبيق الموزع.

ظهرت الحاجة للبرامج الوسيطة بشكل رئيسي في عام 1980 كحل لمشكلة ربط التطبيقات الحديثة مع التطبيقات القديمة(**systems legacy**).

بالإضافة إلى ذلك فإن البرنامج الوسيط هو معيار للتخاطب بين التطبيقات وله خصائص معينة، وهي في الحقيقة متطلبات غير وظيفية مثل الأمن ودعم الويب ودعم المناقلات والثبات وغيرها من الخصائص الضرورية في أي برنامج وسيط.

Socket API

كما ذكرنا سابقاً، ظهرت الحاجة للتواصل بين المهام مع ظهور أنظمة التشغيل ، في البداية كان التواصل بين المهام المحلية (على نفس الحاسب) ، وبعد ذلك تم تعميمه على المهام البعيدة والموجودة على حواسب مختلفة، وبذلك اصبح بالإمكان التواصل بين مهمتين تعملان على حاسبين موصولين عن طريق الشبكة، وهكذا كانت Socket API أول برنامج وسيط (middleware) يربط بين مهمتين سواء على نفس الحاسب أو على حاسبين مختلفين.

ملاحظات متفرقة

- يجب عدم الخلط بين التطبيقات التي تعمل على الشبكة والتطبيقات التي تعمل على الانترنت.
- عادةً يكون كل تطبيق يتألف من Process N، وكل Process تتألف من Thread N.
- المهمة (Process) :هو برنامج قيد التنفيذ.
- النيسب (Thread) : هو عبارة عن مسلك برمجي مستقل في المكدس، ويشترك في الـ Heap مع غيره من ال Threads.

Client/Server Model

مفهوم ال Client/Server هو مفهوم برمجي مستخدم في هندسة البرمجيات، أي أن Client و Server هما تطبيقان.
الذي يطلب هو الزبون (Client) والذي يتلقى الطلبات ويقدم الخدمة يسمى مُخدّم (Server).
لتحقيق الإتصال بين هذين البرنامجين (Client/Server) سوف نستخدم بنية برمجية معروفة ومتوفرة في معظم لغات البرمجة وهي المكتبة Socket.

إذاً ما هي ال Socket ؟

في الحقيقة يوجد عدة تعاريف لل Socket وسنذكر أهمها :

1- Socket : هي عبارة عن بنية برمجية مبنية فوق TCP/IP، وهي تعميم لمفهوم الـ Stream (يمكن إعتبارها Stream على الشبكة لتقريب المعنى) وتتعامل Socket بشكل أساسي مع بروتوكولات طبقة النقل (وتحديداً TCP , UDP) سنتكلم عن TCP فقط.بالإضافة إلى أنها موجودة بشكل معياري في جميع الأنظمة تقريباً.

2- تعريف آخر لـ Socket، هو عبارة عن :
Port: هو عنوان التطبيق المرسل أو المستقبل.
IP: هو عنوان الحاسب المنطقي وهو عنوان فريد unique.

طريقة التخاطب بين تطبيقين

- أولاً يتم تشغيل المخدم، المخدم ينتظر طلب إتصال من الزبون، وعند قيام الزبون بطلب الإتصال بالمخدم، يستقبل المخدم طلب الزبون حيث تتم الموافقة، ثم يتم أرسال وإستقبال المعطيات بين الطرفين وفق بروتوكول معين (يضعه المبرمج).
- الإتصال بين المخدم والزبون يتم عن طريق البروتوكول TCP,UDP كما يمكن استخدام بروتوكول الانترنت الشهير IP.

ملاحظات :

1- Socket تنقل معطيات فقط (مصفوفة بايتات)، ولا تحولها من شكل إلى أخر.
2- المكتبة Socket عابرة للمنصات(Cross Platform)، إي أنها تعمل على جميع نظم التشغيل و تستعمل في معظم لغات البرمجة، والتعامل معها يكون في مستوى منخفض قريب من نظام التشغيل.
3- TCP is Connection Oriented: أي يقيم رابطة ويتحقق من سلامة وصول الطرود، لذلك هو بطيء نوعاً ما ويستخدم في تطبيقات نقل الملفات FTP Server، وهو لايدعم الإرسال إلى عدة مستخدمين (Multicasting or Broadcasting) ، فقط unicast.
4- UDP is Connectionless : أي لا يقيم رابطة ولا يتحقق من سلامة وصول الطرود، لذلك هو أسرع من TCP، ويستخدم غالباً في نقل الفيديو الحي والمباشر (Video Conferencing) كما أنه غير موثوق.

Client/Server in JAVA

سنقوم في هذا المثال البسيط ببناء تطبيق مخدم/زبون بسيط يُسمى Echo Server، حيث يقوم باستلام رسالة الزبون ومن ثم عكس محارفها قبل إرسالها له. سنبدأ بتطبيق المخدم، ثم ننتقل للزبون.

```
package java_echo_server;
import java.net.*;
import java.io.*;

public class Java_echo_server {
    public static void runServer() {
        ServerSocket sock = null;
        try {
            //1: Server Creates Connection and Listening at A specific Prot=5000
            sock = new ServerSocket(5000);
            //2: this Socket is Reversed for The Client that  already Connect With //Server
            Socket clientSock = sock.accept();
            //3: Get input Stream
            InputStream input = clientSock.getInputStream();
            //4: Get outPut Stream
            OutputStream output = clientSock.getOutputStream();
            //To Read Primitive Data Type
            BufferedReader br = new BufferedReader(new InputStreamReader(input));
            //To Write Primitive Data Type
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(output));
            String s = "";
            try {
                s = br.readLine(); // Read the String form Client
                System.out.println(s + "From Client");//Reversing Word
                     String rs = "";
                StringBuffer sb = new StringBuffer(s);
                rs = sb.reverse().toString();
                // "\n" is important Because the Client uses Readline Method
                bw.write(rs + "  \n");
                //Wtithout  this Statment the bw Does not Send Data into Stream
                bw.flush();//Very important
            } catch (IOException e) {
                System.out.println(e.getMessage());
            } finally {
                sock.close();
                clientSock.close();
                bw.close();
                br.close();
                input.close();
                output.close();
            }
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
```

```
        }
    }

    public static void main(String[] args) {
        runServer();
    }
}
```

تطبيق الزبون في الكود التالي :

```
package java_echo_client;
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class Java_echo_client {
public static  void RunClient()
    {
        try
        {
            //1: Connect With Server at a Specific Prot=500
            Socket clientSocket = new Socket("localhost", 5000);
            //2: Get input Stream
            InputStream input =clientSocket.getInputStream();
            //3: Get output Stream
            OutputStream output=clientSocket.getOutputStream();
            //To Read Primitive Data Type
            BufferedReader br=new BufferedReader(new InputStreamReader(input));
            //To Write Primitive Data Type
            BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(output));
            //Reading String From Keyboard By Scanner Class
            Scanner scan=new Scanner(System.in);
            String Word=scan.next();
            // "\n" is important Because the Server uses Readline Method
           bw.write(Word+"  \n");
            //Wtithout  this Statment the bw Does not Send Data into Stream
           bw.flush();//Very important
           //Waiting for Reading The Reverse word from Server
           String s= br.readLine();
           System.out.println(s+"From Server");
           //Release The Resources
           clientSocket.close();
           br.close();
           bw.close();
           input.close();
           output.close();
        }
        catch (IOException ex)
        {
            System.out.println(ex.getMessage());
```

```
        }
    }
    public static void main(String[] args) {
      RunClient();
    }
}
```

شرح لآلية عمل البرنامجين

في البداية يقوم المخدّم بفتح إتصال على Port معين ثم ينتظر طلب الزبون وذلك عن طريق التعليمة:

```
Socket clientSock = sock.accept();
```

تُشبه هذه التعليمة تعليمة القراءة من لوحة المفاتيح، حيث أنها توقف العمل(blocking) حتى يتصل الزبون، عندها يتابع المخدم عمله، ثم يحصل على مجاري الدخل والخرج، ثم يقوم المخدم بإنتظار الزبون حتى يرسل له معطيات معينة عن طريق التابع ReadLine()، عند ذلك يقوم الزبون بإرسال سلسلة نصية معينة ثم يقوم المخدم بقراءتها وطباعتها على الشاشة، ومن ثم يقوم بعكسها وإرسالها إلى الزبون، (في هذه الحالة يكون الزبون في وضع إنتظار لرسالة المخدم والتي تحوي السلسلة المعكوسة، وذلك عن طريق التابع Redline() عند الزبون)، ثم يقوم الزبون بقراءتها وطباعتها على الشاشة، ثم يحرر كلا البرنامجين جميع الموارد (مثل المجاري المفتوحة Stream و Socket).

نلاحظ أن هذا التطبيق هوتطبيق بسيط يستقبل المخدم طلب واحد من زبون واحد ويخدمه فقط، ولكن ماذا إذا أراد المخدم أن يقدم خدماته لأكثر من زبون بنفس الوقت، ماذا نفعل؟
في الحقيقة إن برنامج الزبون في هذه الحالة لايتغير، ولكن برنامج المخدم هو الذي يتغير تغيراً جذرياً.

ما هي أنواع التطبيقات التي يمكن إنشاؤها عن طريق Socket

1- تطبيق Peer To Peer : وهو برنامج موجود على حاسبين مختلفين وكل تطبيق منهم هو بمثابة مخدم وزبون في آن واحد.

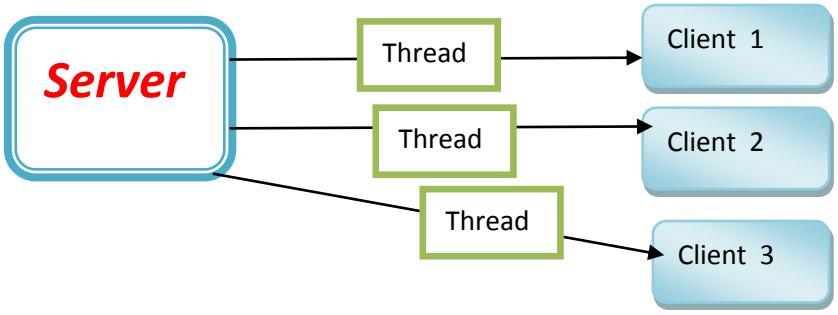
2- تطبيق Client /sever :

وهو مفهوم برمجي كما قلنا سابقاً، ولكن يمكن أن نضع إمكانيات أو قدرات أو مزايا للمخدم، منها :

- يمكن أن يرسل الزبون إلى المخدم ملفات أو رسائل نصية، ويستقبلها المخدم ثم يخزنها في الحاسب لديه، ثم يقوم الزيون بإغلاق الإتصال، في هذه الحالة إذا أراد الزبون إرسال رسالة مرة أخرى عليه الإتصال مرة أخرى مع المخدم(كما في مخدمات الويب)، لذلك في بعض التطبيقات التي تحتاج لإن يتصل الزبون مع المخدم لفترة معينة نقوم بإنشاء جلسة (Session) بين الزبون والمخدم تنتهي هذه الجلسة برغبة الزبون عند الضغط على زر Disconnect مثلاً.
- لكن إذا إتصل أكثر من زبون مع المخدم في نفس الوقت ماذا يفعل المخدم حينها؟
  - يوجد ثلاثة أوضاع (Mode) يمكن للمخدم أن يعمل بها لكي يتعامل مع عدد كبير من طلبات الزبائن في نفس الوقت الإستراتيجيات هي :

1- **Synchronization Mode :** وتسمى وضع التزامن أو تسمى Blocking Mode، ويعني أن المخدم يقوم بإنشاء نيسب (thread) خاص لكل زبون يتصل به، أي أن المخدم يستطيع أن يقدم خدماته لكل الزبائن المتصلين به في الزمن الحقيقي (Real Time).

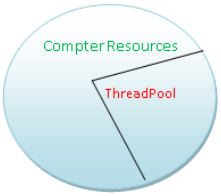
مثال : إذا إردت التحدث إلى 3 أشخاص بواسطة الهاتف فإنك تحتاج إلى 3 خطوط و 3 هواتف.



- **مزايا هذه الطريقة :**
  - سهلة نوعاً ما (مقارنة بوضع عدم التزامن).
  - جميع الطلبات يتم معالجتها بالزمن الحقيقي، لا يوجد تأخير.

- **مساوئ هذه الطريقة :**
  - مكلفة جداً من حيث إستهلاك موارد المخدم، حيث أن كل Thread يستهلك وقت وزمن من المعالج ويأخذ ذاكرة إيضاً وكما نعلم الموارد محدودة.
  - استهلاك عالي للموارد (Ram + CPU Time).
  - إقلاع النيسب يأخذ وقت طويل نسبياً.
  - يمكن لهذا المخدم أن يتعرض لهجمة، لذلك بعض المواقع تطلب إدخال حروف وأرقام من صورة، وذلك للتاكد أن من من يقوم بعملية التسجيل هو إنسان وليس برنامج يقوم بالهجوم على المخدم!

يمكن إستخدام هذه الطريقة في بعض الحالات التي نضمن فيها عدد محدد للزبائن المتصلين، كما يجب أن نضمن عدم تعرض المخدم لهجمة معينة.



2- **حوض النياسب Thread Pool :** نقوم بحجز مجموعة من النياسب التي تعمل بشكل دائم (لا تموت أبداً)، لتخديم عدد معين من الزبائن والبقية تنتظر لزمن معين (يُفترض أن يكون مقبول).
تخصص هذه الطريقة (Thread Pool) جزء من موارد الحاسب ليتم إعادة استخدامها.
- حجم هذا الحوض قد يكون ثابت أو متغير إلى حد معين.

- تقوم مخدمات الويب (Web Servers) ومخدمات قواعد المعطيات (Data Base Servers) باستخدام حوض من النياسب (Thread Pool) بدلاً من تخصيص نيسب خاص لكل زبون، وبذلك نستطيع التخلص من المشاكل الموجودة في وضعية التزامن.

3- **A Synchronization Mode :** وضع اللا تزامن ويسمى Unblocking Mode لتوضيح المعنى سنشرح المثال التالي :
لنفرض أنه لدينا تطبيق Server يتنصت على Port محددة ليستقبل المعطيات منها، ولنميز بين وضعي التزامن وعدم التزامن :

1- في وضع التزامن أو Blocking Mode : لا يستطيع المخدم أن يقوم بأي شيء حتى ينتهي من عملية إستقبال المعطيات، وعند إتصال زبون أخر لا يستطيع المخدم أن يقبل الإتصال منه لإنه محجوز من قبل أول زبون(Blocked) .
2- في وضع عدم التزامن None Blocking Mode: في هذه الحالة يستطيع المخدم عند إستقبال المعطيات أو التنصت على Port معينة أن يقوم بمعالجة طلبات أكثر من زبون في نفس الوقت. والخلاصة :

a. في وضعية عدم التزامن يستطيع المخدم معالجة عدة طلبات من عدة زبائن في نفس الوقت من دون إستخدام Thread بشكل صريح وذلك عن طريق توابع خاصة موجودة في شبكة Socket تبدأ ب Begin وتنتهي ب End.
مثال :

BeginAccept(); - EndAccept();

b. العملية تشبه تماماً عملية الإرسال على المسرى الغير المتزامن حيث نحتاج إلى إشارات Begin == Req() و Ack==End()
c. في الحقيقة إن هذه الطريقة من أفضل الطرق من حيث الإداء وتوفير الموارد، ولكنها الأصعب من حيث البرمجة حيث نستطيع تخديم أكثر من زبون على Thread واحدة بنفس الوقت.
d. في الحقيقة إن التوابع الخاصة بوضع اللاتزامن تستعمل Threading في مستوى نظام التشغيل.

أي الطرق نختار ؟

في الحقيقة إن الموضوع نسبي (حسب واقع العمل الذي يعمل به التطبيق).مثلاً : إذا قام كل زبون بإرسال رسالة أو ملف معين إلى المخدم، ثم قام بإغلاق الإتصال عندئذٍ تكون طريقة Thread Pool ممتازة جداً. ولكن إذا كان كل زبون يقيم جلسة مع المخدم ولا نعلم متى سينهي الإتصال، في هذه الحالة لا نعلم ما هو مقدار التأخير للزبائن في رتل الإنتظار لذلك تكون هذه الطريقة (Thread Pool) غير فعالة في هذه الحالة.

حالة دراسية مخدم ملفات يستخدم حوض نياسب(Thread Pool)

سنقوم ببناء نظام زبون/مخدم يسمح للزبون بطلب ملف من المخدم، وفي حال وجوده لديه يقوم بإرساله للزبون، وفي حال عدم وجوده يقوم بسؤال مخدم أخر عنه، في حال وجوده لديه يقوم بإرساله للمخدم الأول والمخدم الأول يرسله للزبون.
كلا المخدمين يستعملان حوض نياسب له حجم ثابت، وفي حال تم استخدام كامل النياسب في هذا الحوض يتمدد هذا الحوض بمقدار محدد وبشكل ديناميكي ليقوم بتخديم الزبائن.
الكود التالي يمثل مقاطع من تطبيق المخدم.

```
    //Number of Static Connection in The S_arr
    public static int StaticConnections=10; //Edited in The Configuration of The Server
    //Number of Dynamic  Connection in The S_arr
    public static  int dynConnections=5; //Edited in The Configuration of The Server
     // first (n=StaticConnections) Connections(object from HandleRequests)
    //is Static and Remains  is dynamic as nedded
    public static  HandleRequest [] S_arr=new HandleRequest[StaticConnections+dynConnections];
//S_arry is Static Array
```

```
//Return True if S_arr has unoccupied Threads
public static  boolean  isnotfull()
{
    for(int i=0; i<S_arr.length; i++)
    {
        if ((S_arr[i]!=null) &&(S_arr[i].ServerThreadState==HandleRequsetState.unoccupied))
                return  true;
          else if (S_arr[i]==null)
               return false;
    }
    return false;
}

public   static void main(String[] args)
{
    ServerSocket server=null;
    try
    {
        //Server Open Cnnection at Prot=5000
         server= new ServerSocket(5000);

         int co=Thread.activeCount();

         for(int i=0; i<StaticConnections; i++)
         {
             //All Thread is Shared With this Connections(Server)
             S_arr[i]=new HandleRequest(server);

             S_arr[i].start();
         }

        int co1=Thread.activeCount();

         boolean ok;
         //if S_arr has an unoccupied Threads
         while((ok=isnotfull()))
         {
             System.out.println("in While Number of Thread = "+Thread.activeCount());
         }

         for(int j=StaticConnections; j<S_arr.length;j++)
         {
             S_arr[j]=new HandleRequest(server);

             S_arr[j].start();

             // Also if S_arr has an unoccupied Threads
```

```
                while((isnotfull()))
                {
                   System.out.println("in for loop Number of Thread = "+Thread.activeCount());
                }
            }

        }//End Try
        catch (IOException ex)
        {
            Logger.getLogger(FTPServer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

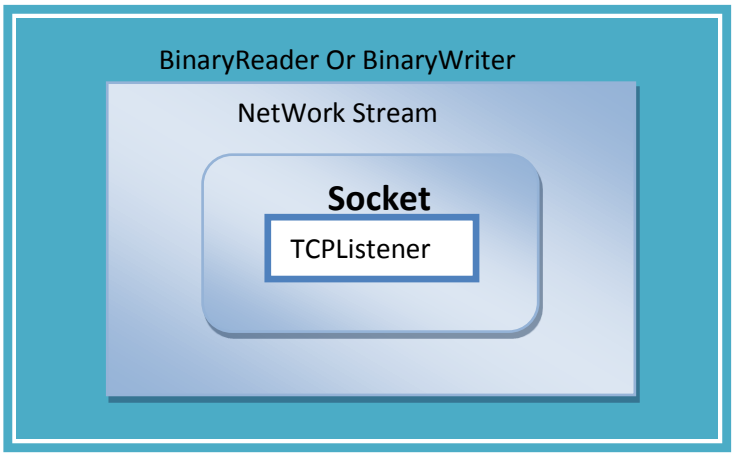
عندما يعمل تطبيق المخدم يقوم بإنشاء حوض نياسب بحجم مبدئي 10 نياسب وهي قيمة المتحول StaticConnections. في حال وصل للمخدم 11 طلب من الزبائن المتصلين في نفس الوقت فإن أول 10 زبائن يتم تخديمهم بواسطة النياسب العشرة. والزبون رقم 11 يتم إنشاء نيسب خاص له ليتم تخديمة وذلك من احتياطي عدد النياسب الإضافي والذي يمثله المتحول dynConnections .

يمكن الاطلاع على الكود البرمجي للتطبيق من خلال الرابط التالي :

https://github.com/MhdAlyan/JavaftpserverThreadPool

خطوات إنشاء تطبيق مخدم في C#

سنوضح كيفية تحقيق إتصال بين تطبيقين أحدهما مخدم والأخر زبون باستخدام بروتوكول TCP.



1- Tcplistener: وهو صف يستخدم فقط في تطبيق المخدم (يكافئ ServerSocket في جافا)، ويقوم بالتنصت على عنوان محدد لهذا التطبيق وهو عنوان فريد (على مستوى المهام الموجودة في نظام التشغيل)، يسمى هذا العنوان Port . يقوم بالتنصت على هذه الـ port ومن ثم يبدأ باستقبال الطلبات.

2- Socket: في الحقيقة نحن ننشئ Reference على Socket، والغرض من TcpListener يستخدم تابع يسمى AcceptSocket() يرد لنا Socket التي نعمل عليها حالياً.

3- NetworkStream : وهي stream لكن على الشكبة، نفس الفكرة هنا NetworkStream تغلف ال socket السابقة.

4- BinaryReader /BinaryWirter : هذان الصفان يغلفان أي stream وبالتالي فهما يغلفان NetworkStream، ثم يتم الكتابة أوالقراءة ولكن بمستوى أعلى من المستوى السابق، أي أننا نكتب أو نقرأ (Primitive data Type).

- كل ما يخص ال Socket موجود في :
  - فصاء الأسماء System .Net
  - فصاء الأسماء System.Net.Socket

طريقة التغليف

```
TcpListener listener =new TcpListener(5000);// 5000 is Port number
listener.Start(); // Start listening  for incomming connection Requests
Socket myscoket=listener.AcceptSocket();
// Encapsulate object from Socket Class(mysockets)
NetworkStream mynetworkStream=new NetworkStream(myscoket) ;
BinaryReader reader= new BinaryReader(mynetworkStream);
BinaryWriter writer=new BinaryWriter(mynetworkStream);
```

خطوات إنشاء تطبيق Client

نستخدم هنا صنف اسمه TcpClient وهو يمثل طلب زبون يقوم بالإتصال مع المخدم، ويحصل من هذا الإتصال على stream، ثم يقوم بتغليف هذا الـ stream بواسطة BinaryReader/BinaryWriter كما تكلمنا سابقاً. نقوم بعملية الإرسال والإستقبال ضمن بروتوكول محدد يتم الإتفاق عليه من قبل المرسل والمستقبل (نحن قمنا ببناء هذا البوتوكول).
سنقوم الآن بشرح مثال عن برنامج Chat بسيط يجسد مفهوم المخدم/الزبون. نبدأ بتطبيق المخدم.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Net.Sockets;
using System.Threading;

namespace socket
{
    /// <summary>
    ///  to build application As Server we have to :
    ///  1- object form Tcplistener(Class)
```

```
    ///  2- object from Socket(Class) that takes object from Tcplistener By
//AcceptSocket() method ;
    ///  3- object from Networkstream that tackes object from socket class
    ///  4- two  object from streamreader and streamwriter  Or  BinaryReader
   ///  and BinaryWritter ; each of this take object from newrorkstram(Class)
   /// </summary>
 public partial class Server : Form
{

       //Socket is Encapsulates object form Tcplistener (Ex: connection=new
       //TCplistener().Acceptsocket(); )
        Socket connection; //Socket is used to Link  Transport layer(TCPlistener OR
       TCPClient) With //Network layer(NetworkStream)

       hread read_thread; // to Run this Connection With anthor processes

       NetworkStream socket_stream; // Encapsulate object from Socket
       //Class(Connection)

       BinaryReader reader;//Encapsulate object from Networkstream(socket_stream)and Reading
       from   //Network

       BinaryWriter writer;//Encapsulate objectfrom Networkstream(socket_stream)and Writing
       on //Network


        // Thread reciev_thread;
         public Server()// Run Thread in Constructor of Server_Form
         {
             InitializeComponent();
             ThreadStart ts = new ThreadStart(Run_Server);

             read_thread = new Thread(ts);
             read_thread.Start();
         }

       //   إنهاء عمل كافة ال  threads عند إغلاق البرنامج
        private void Server_FormClosing(object sender, FormClosingEventArgs e)
         {
             //Terminates  All thread in the Application
            System.Environment.Exit(System.Environment.ExitCode);
         }

       // حدث إدخال نص حيث يتم إرسال هذا النص إلى الزبون
         private void input_text_KeyDown(object sender, KeyEventArgs e)
         {
             try
             {
                 if ((e.KeyCode == Keys.Enter) && (connection != null))
                 {
                     // sends the text to the client
                     writer.Write("Server>>" + input_text.Text);
                     output_text.Text += "\r\nSERVER>> " + input_text.Text;

                     // if the Client(Sender) wants to End the session
                     if (input_text.Text == "terminate")
                     {
                         connection.Close();
```

```
                    Application.Exit();
                }
                input_text.Clear();
            }
        }
        catch(SocketException se)
        {
            MessageBox.Show(se.Message);
        }
    }

    // allows a client to connect and displays the text it sends
    public void Run_Server()
    {
        TcpListener Tcp_listener;
        int counter = 1;

        // wait for a client connection and display the text
        // that the client sends
        try
        {
            // Step 1: create TcpListener
            Tcp_listener = new TcpListener(5000);

            // Step 2: TcpListener waits for connection request
            Tcp_listener.Start(); // Start listening  for incomming connection Requests

            // Step 3: establish connection upon client request
            while (true)
            {
                output_text.Text = "Waiting for connection\r\n";

                // accept an incoming connection ; Step 5
               //  Socket Class in Encapsulates TCPlistener Class
                connection = Tcp_listener.AcceptSocket();

                //  Step 6
                // create NetworkStream object associated with socket ;
                socket_stream = new NetworkStream(connection);
              // NetworkStream Class in Encapsulates Socket Class

        // create objects for transferring data across stream  Step 7 & 8
                writer = new BinaryWriter(socket_stream);

                reader = new BinaryReader(socket_stream);


          output_text.Text += "Connection " + counter + " received.\r\n";

// in form client that connection was successfull  ; Writin in the Stream
(Network)

                writer.Write("Server>> Connection Successful");

                string theReply = "";
```

```
                    // Step 9: read String data sent from client
                    do
                    {
                        try
                        {
                            // read the string sent to the server
                          theReply=reader.ReadString();//Reading from Network

                            // display the message
                            output_text.Text += "\r\n" + theReply;
                        }

                        // handle exception if error reading data
                        catch (Exception)
                        {
                            break;
                        }
                               // important Create Session
                    } while ((theReply != "Client>>terminate") && (connection.Connected));

                    output_text.Text += "\r\nUser terminated connection";

                    // Step 10: close connection

                    input_text.ReadOnly = false;

                    writer.Close();

                    reader.Close();

                    socket_stream.Close();

                    connection.Close();

                    ++counter;

                    Application.Exit();
                }
                } // end try

                catch ( Exception error )
                 {
                    MessageBox.Show( error.ToString() );
                 }

        }
    }// End Class Server
}
```

والآن ننتقل إلى الجزء الخاص بالزبون :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;

using System.IO;
using System.Net.Sockets;
using System.Threading;


namespace Socket_in_Client
{
    public partial class Client : Form
    {

        NetworkStream output_stream;
        BinaryReader reader;
        BinaryWriter writer;
        Thread readthread;
        string message = "";

        TcpClient client; // for Connect With Server

        // default constructor
        public Client()
        {
            InitializeComponent();

            ThreadStart ts = new ThreadStart(Run_client);

            readthread = new Thread(ts);

            readthread.Start();
        }

         //  إرسال الرسائل النصية إلى المخدم
        private void textBox1_KeyDown(object sender, KeyEventArgs e)
        {
            try
            {
                if (e.KeyCode == Keys.Enter)
                {
                   writer.Write("Client>>" + input_text.Text);//Send to Sever

                  output_text.Text += "\r\nClient Say:>> " + input_text.Text;

                   input_text.Clear();


                }
            }
             catch(SocketException se)
            {
                output_text.Text += "\n Error waiting Object";
            }
        }
```

```
        //  الزبون يستقبل الرسائل من المخدم
         public void Run_client()
         {
             try
             {
               output_text.Text += "Attempting to Connecting to the server\n";

                 // Step 1: Create TCPClient and connect to the Server
                 client = new TcpClient();

                client.Connect("localhost", 5000);
             // 5000 is the port number  that the Sever is listening on it

                 // Step 2: Get NetworkStream Associated With TcpClient
                 output_stream  = client.GetStream();

// Step 3: Create Object for Writing and Reading Across Stream( NetworkStream)
                 writer = new BinaryWriter(output_stream);
                 reader = new BinaryReader(output_stream);
                // output_text.Text += "\r\nGot IO Stream \r\n";
                 input_text.ReadOnly = false;

                 do
                 {
                     //Step 3:
                     try
                     {
                         //Reading the message form Server

                         message = reader.ReadString();

                         output_text.Text += "\r\n" + message;

                     }
                     catch (Exception e)
                     {
                         System.Environment.Exit(System.Environment.ExitCode);
                     }
                 } while (message!="Server>>terminate");


                 // Step 4: Closing Connection
                 writer.Close();
                 reader.Close();
                 output_stream.Close();
                 client.Close();

                 Application.Exit();
             }
             catch (SocketException se)
             {
                 MessageBox.Show(se.Message);
             }

         }
```

```
    }//End Client Class
}
```

كما يمكنكم مشاهدة و تحميل الشيفرة المصدرية كاملة من هنا

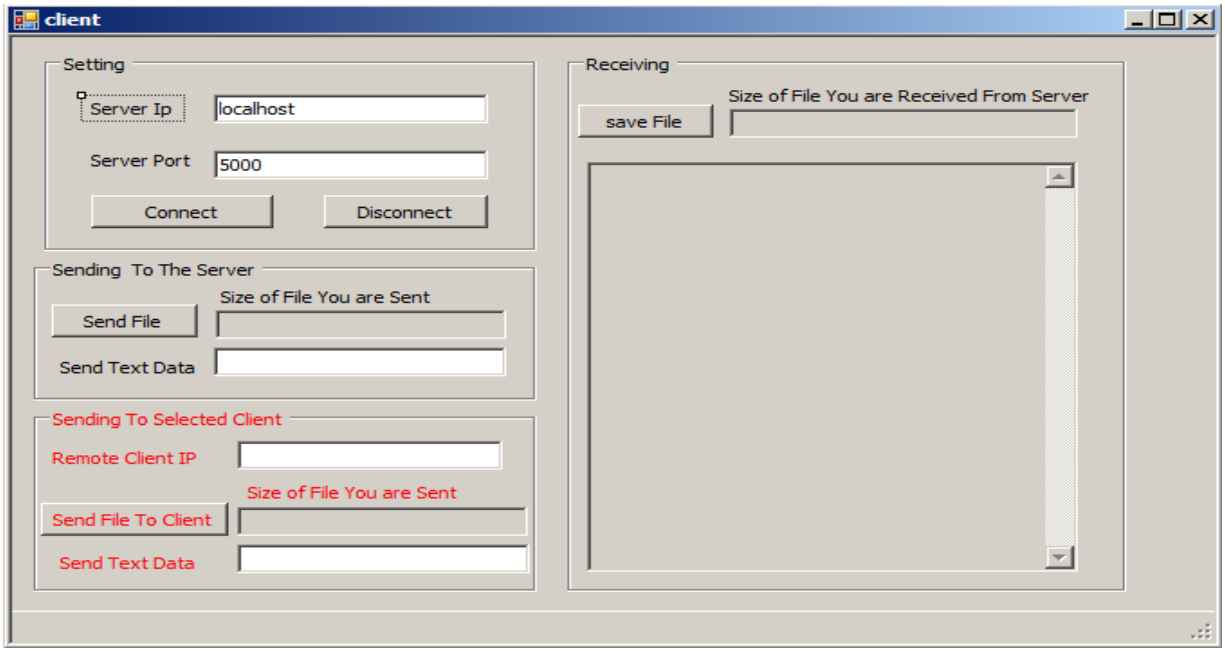
https://sourceforge.net/projects/basic-chat-program/

المتطلبات اللاوظيفية

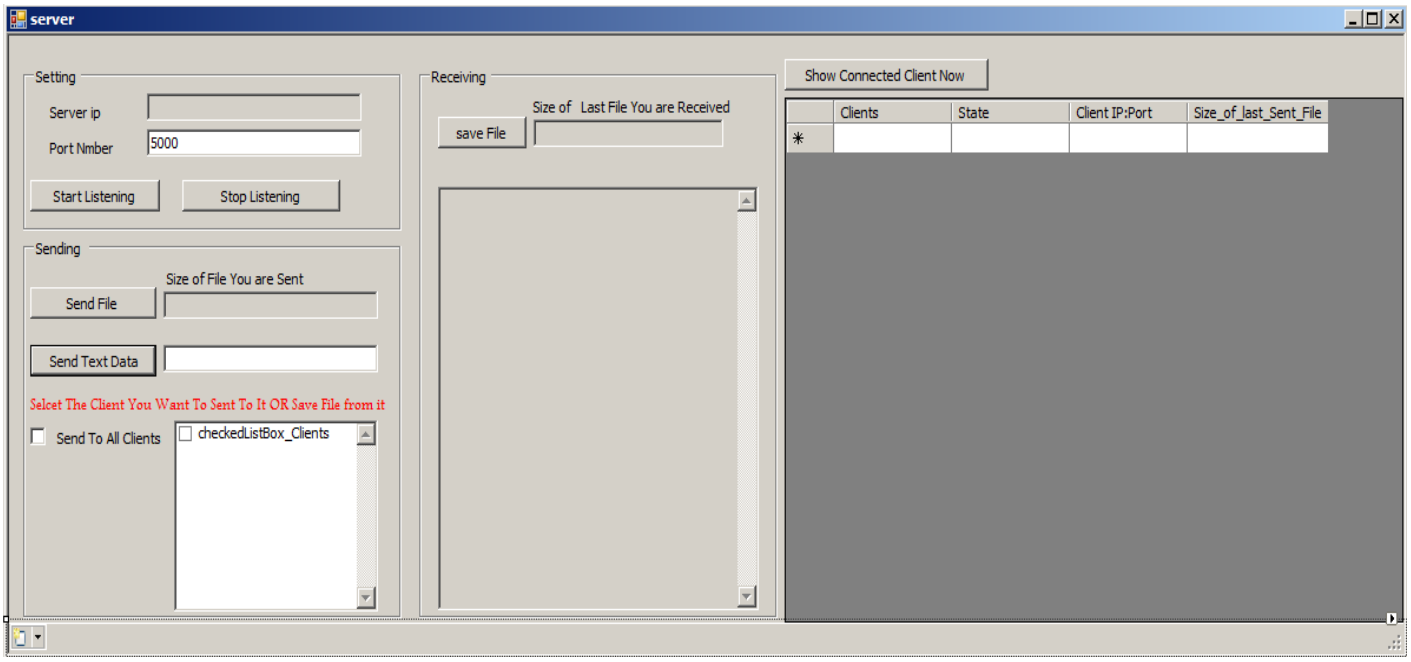
من المعروف أنّ لكل حاسب أداء، وأيضاً لكل تطبيق أداء والتطبيق الناجح هو التطبيق الذي يستغل موارد الحاسب الذي يعمل عليه بالشكل الأمثل، أداء التطبيق لا يتعلق بسرعة الحاسب وحجم الذاكرة المستخدمة فقط إنما يتعلق بالطريقة أو المنهجية التي يتبعها التطبيق في التعامل مع موارد الحاسب، ما يهمنا في الموضوع هو إمكانيات المخدم لأن معظم الحمل عليه لذلك يتوجب على الشركة التي سوف تشتري النظام أن تشتري مخدم قوي وهذا يتناسب طرداً مع السعر.

تطبيق مخدم متعدد النياسب

من خلال هذا المخدم يمكن للزبائن المتصلين أن يقوموا بالمحادثات النصية وتبادل الملفات مع بعضهم البعض ، كما يسمح للزبون أن يقوم بمحادثة مع المخدم وتبادل الملفات بينهما.
واجهة تطبيق الزبون في الصورة التالية.



واجهة تطبيق المخدم في الصورة التالية.



يمكن الاطلاع على الكود البرمجي للتطبيق من خلال الرابط التالي :

https://sourceforge.net/projects/csharp-multithreadedftpserver/