برنامه‌نویسی سوکت TCP با فرم‌های ویندوز در C#

برای برقراری یک ارتباط TCP، به یک برنامه سرویس‌دهنده (Server) و یک برنامه مشتری (Client) نیاز داریم:

- در برنامه سرویس‌دهنده، یک سوکت تعریف می‌شود که باید به درخواست‌های احتمالی روی یک پورت مشخص «گوش کند» و پس از دریافت درخواست آن را بپذیرد.
- در برنامه مشتری باید درخواست اتصال به سوکت مزبور (IP:Port Number) ارسال شود و پس از پذیرش توسط سرویس‌دهنده، تبادل اطلاعات آغاز گردد.

دو نوع برنامه سرویس‌دهنده وجود دارد:

۱) برنامه سرویس‌دهنده سنکرون

این برنامه در حال انتظار برای دریافت درخواست اتصال، قفل (Block) می‌شود و توانایی انجام هیچ کاری را ندارد؛ اما نوشتن و درک آن ساده است.

۲) برنامه سرویس‌دهنده آسنکرون

این برنامه به کمک threadها، وظیفه پذیرش درخواست‌های اتصال و دریافت داده‌ها را بدون قفل شدن انجام می‌دهد. این برنامه نسبت به نسخه سنکرون پیشرفته‌تر و در عین حال پیچیده‌تر است.

نوشتن برنامه‌های سوکت در محیط کنسول .NET چندان مشکل نیست. لذا در این نوشتار، به تشریح برنامه‌های سرویس‌دهنده و مشتری که به کمک فرم‌های ویندوز در C# طراحی می‌شوند، می‌پردازیم.

برنامه سرویس‌دهنده سنکرون در C#

برای نوشتن این برنامه، یک سوکت به نام sktListener به صورت عمومی به شکل زیر تعریف می‌شود:

```
Socket sktListener;
```

توجه کنید برای استفاده از کلاس Socket باید فضاهای نام جدیدی را به برنامه‌تان اضافه کنید:

```
using System.Net;
using System.Net.Sockets;
```

به کمک دستور زیر، مشخصات سوکت را تعریف می‌کنیم:

```
sktListener = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
```

به کمک دستور زیر، یک سوکت روی پورت ۱۸۰۰ کامپیوتر سرویس‌دهنده تعریف می‌شود:

```
IPEndPoint ipLocal = new IPEndPoint(IPAddress.Parse("127.0.0.1"),1800);
```

توجه کنید که از آدرس Loop back به جای آدرس IP کامپیوتر سرویس‌دهنده استفاده کرده‌ایم. طبیعی است که می‌توانید از آدرس IP کامپیوتر سرویس‌دهنده استفاده کنید.

اکنون باید آدرس سوکت تعریف شده (ipLocal) را به سوکت sktListener پیوند بزنیم:

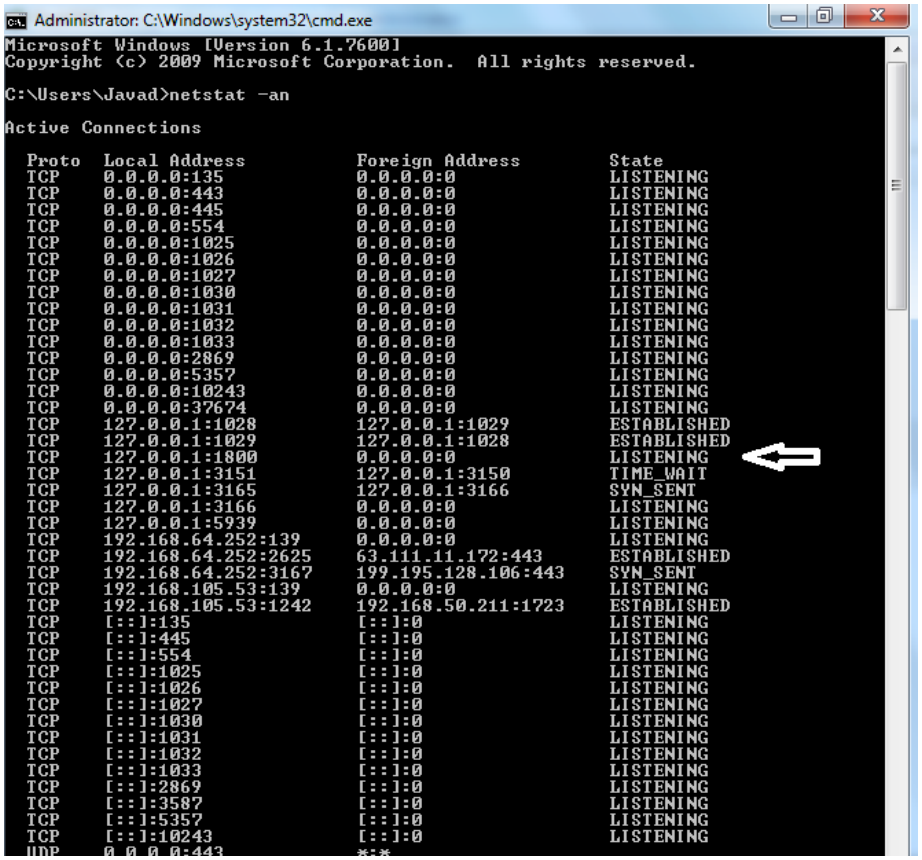
```
sktListener.Bind(ipLocal);
```

سوکت مزبور باید به درخواست‌های احتمالی اتصال گوش فرا دهد:

```
sktListener.Listen(10);
```

این دستور به سیستم‌عامل اعلام می‌کند که برنامه‌ی سرویس‌دهنده، می‌تواند تا سقف ۱۰ مشتری که در یک لحظه درخواست اتصال به برنامه‌ی سرویس‌دهنده بدهند را سرویس بدهد؛ در این حالت یکی از درخواست‌ها بلافاصله و بقیه‌ی درخواست‌ها نیز در ادامه پذیرفته می‌شوند.

با اجرای دستور netstat -an در محیط command prompt ویندوز، می‌توانید مشاهده کنید که برنامه مزبور روی پورت ۱۸۰۰ مشغول گوش کردن است:

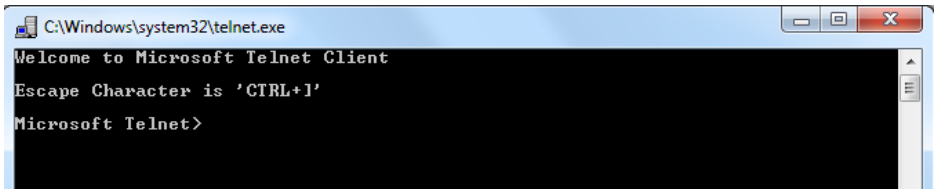


اکنون زمان پذیرش درخواست‌های اتصال است:

```
sktListener = sktListener.Accept();
```

با اجرای این دستور، برنامه سرویس‌دهنده قفل می‌کند و در انتظار ارائه درخواست برنامه‌های مشتری می‌ماند. در این حالت برنامه سرویس‌دهنده قادر به انجام هیچ کاری نیست.

به کمک برنامه Telnet می‌توانید به برنامه سرویس‌دهنده مزبور متصل شوید. برای این کار، در کادر Run از منوی Start ویندوز کلمه Telnet را تایپ و کلید Enter را فشار دهید. محیط Telnet به صورت زیر نمایش داده خواهد شد:



در صورتی که از ویندوز پیغام خطا دریافت کردید، باید از مسیر زیر قابلیت Telnet سیستم خود را از بخش Windows Features فعال کنید:

```
Control Panel\Programs and Features
```

اکنون دستور زیر را در محیط Telnet تایپ کنید:

```
open 127.0.0.1 1800
```

با این کار، اتصالی بین برنامه مشتری (Telnet) و برنامه سرویس‌دهنده برقرار شده و برنامه سرویس‌دهنده از حالت قفل‌شدگی خارج می‌شود. به کمک دستور زیر در برنامه سرویس‌دهنده می‌توانید مشخصات مشتری متصل شده به برنامه را ببینید:

```
MessageBox.Show(sktListener.RemoteEndPoint.ToString());
```

اکنون اتصال بین برنامه‌های مشتری و سرویس‌دهنده برقرار شده و می‌توانند با هم تبادل اطلاعات کنند.

توجه کنید که چون برنامه Telnet را روی همان کامپیوتری اجرا کرده‌ایم که برنامه سرویس‌دهنده روی آن اجرا می‌شود، باز هم از آدرس Loop Back استفاده کرده‌ایم. بدیهی است اگر مشتری از روی کامپیوتر دیگری اجرا شود، باید آدرس IP کامپیوتر سرویس‌دهنده به جای آدرس Loop Back مورد استفاده قرار گیرد.

با اجرای دستور netstat –an در محیط command prompt ویندوز، می‌توانید مشاهده کنید که برنامه مزبور روی پورت ۱۸۰۰ مشغول گوش کردن است و در عین حال، یک اتصال نیز برقرار (Established) شده است.

اکنون می‌توانید برنامه سرویس‌دهنده را تکمیل کنید که داده‌های ارسال شده توسط مشتری را بپذیرد:

```
try
{
    byte[] buffer = new byte[500];
    sktListener.Receive(buffer);

    string str = System.Text.UTF8Encoding.UTF8.GetString(buffer);
    MessageBox.Show(str,"Received Message");
}
catch (SocketException err)
{
    MessageBox.Show(err.Message,"Server Error");
}
```

با این کار، برنامه سرویس‌دهنده داده‌هایی را که از طرف برنامه مشتری دریافت می‌کند، نمایش می‌دهد. مثلاً بعد از برقراری اتصال در برنامه Telnet، هر آنچه تایپ کنید پس از اجرای دستورات بالا در برنامه سرویس‌دهنده نمایش داده خواهند شد.

یک نمونه برنامه سرویس‌دهنده سنکرون را می‌توانید از اینجا دریافت کنید. با اجرای برنامه مزبور روی کامپیوتر سرویس‌دهنده، کادری به شکل مقابل ظاهر خواهد شد. آدرس IP کامپیوتر سرویس‌دهنده و آدرس پورت مد نظرتان را در جعبه‌های متنی وارد کنید و دکمه «صدور مجوز ارتباط» را فشار دهید. خواهید دید که برنامه به حالت بلوک‌شده خواهد رفت. سپس به کمک برنامه Telnet به این برنامه سرویس‌دهنده متصل شوید. با این کار، دکمه «صدور مجوز ارتباط» غیرفعال شده و دکمه «به‌روزرسانی» فعال می‌شود. متنی را در Telnet تایپ کنید و دکمه «به‌روزرسانی» را در برنامه فوق فشار دهید تا متن ارسال شده را ببینید.

پرسش) برنامه‌ی دانلود شده را طوری تغییر دهید که در آنِ واحد بتواند به حداکثر ۱۰ مشتری سرویس دهد؛ برای این کار، یک آرایه‌ی ۱۰ عنصری از جنس Socket تعریف کنید و برنامه را طوری تغییر دهید که قابلیت اتصال و دریافت/ارسال اطلاعات با هریک از سوکت‌ها را داشته باشد. یک نمونه برنامه را می‌توانید از اینجا دریافت کنید.

برنامه مشتری در C#

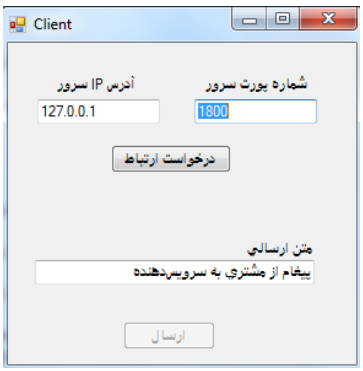
در این برنامه، یک سوکت مانند آنچه در برنامه سرویس‌دهنده گفته شد به صورت عمومی تعریف می‌شود که نام آن را client می‌گذاریم. دستورات زیر، سوکت مزبور را به سوکت تعریف شده در برنامه سرویس‌دهنده متصل می‌کند:

```
client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

try
{
   client.Connect(IPAddress.Parse("127.0.0.1"), 1800);
}
catch (SocketException err)
{
   MessageBox.Show(err.Message,"Client Error");
}
```

این اتصال دقیقاً مانند اتصال توسط برنامه Telnet عمل می‌کند. در صورتی که قبل از اجرای دستورات مزبور، سوکت برنامه سرویس‌دهنده مشغول گوش کردن باشد، اتصال بین سوکت‌های مشتری و سرویس‌دهنده برقرار می‌شود. اکنون می‌توانید داده‌های ماشین مشتری را برای ماشین سرویس‌دهنده ارسال کنید:

```
try
{
  byte[] msg = Encoding.UTF8.GetBytes("Hello");
  client.Send(msg);
}
catch (SocketException err)
{
  MessageBox.Show("Error:" + err.Message);
}
```



یک نمونه برنامه مشتری را می‌توانید از اینجا دریافت کنید. پس از وارد کردن آدرس IP کامپیوتر سرویس‌دهنده و پورت مورد نظر، دکمه درخواست ارتباط را فشار دهید. در صورتی که برنامه سرویس‌دهنده مشغول گوش کردن باشد، ارتباط برقرار شده و دکمه «ارسال» فعال می‌شود. با فشار این دکمه، متنی که در کادر زیر آن تایپ شده است برای برنامه سرویس‌گیرنده ارسال خواهد شد.

توجه کنید که از متد Send برای ارسال داده‌ها در برنامه سرویس‌دهنده نیز می‌توانید استفاده کنید؛ مثلاً اگر با Telnet به برنامه سرویس‌دهنده متصل شده باشد، اجرای دستورات زیر در برنامه سرویس‌دهنده پس از برقراری ارتباط، پیغام Hello را در محیط Telnet نمایش می‌دهد:

```
byte[] ss = Encoding.UTF8.GetBytes("Hello");
sktListener.Send(ss);
```

به این طریق می‌توانید یک ارتباط دوسویه بین کامپیوتر سرویس‌دهنده و محیط Telnet برقرار کنید.

تمرین) یک برنامه سرویس‌دهنده بنویسید که روی پورت ۲۲۲۲ یک اتصال از Telnet دریافت کند. در محیط Telnet سه دستور به شرح زیر تعریف می‌کنیم:

```
USER xxxx
PASS xxxx
```

SHUTDOWN

برنامه سرویس‌دهنده پس از دریافت اتصال باید پیغام Welcome! را برای Telnet ارسال کند. پس از دریافت دستور USER xxxx برنامه سرویس‌دهنده باید نام کاربری وارد شده را کنترل کند و در صورتی که صحیح بود پیغام OK و در غیر این صورت پیغام Not Accepted را برای Telnet ارسال کند. به همین نحو باید صحت رمز عبور نیز کنترل شود. در صورتی که نام کاربری و کلمه عبور صحیح بود، با ارسال فرمان SHUTDOWN از محیط Telnet، باید کامپیوتر سرویس‌گیرنده خاموش شود. پیش از خاموش شدن، برنامه سرویس‌دهنده باید از طریق Telnet از کاربر سؤال کند که آیا مطمئن است می‌خواهد سیستم را خاموش کند و در صورت دریافت پاسخ مثبت، درخواست وی را اجرا کند. از قطعه کد زیر برای خاموش کردن کامپیوتر استفاده کنید:

```
using System.Diagnostics;

    void Shutdown()
    {
        Process.Start("shutdown.exe", "-s -t 00");
    }
```

توجه کنید که اگر ابتدا به کمک Telnet و سپس به کمک برنامه مشتری به برنامه سرویس‌دهنده متصل شوید، هر دو اتصال پذیرفته خواهند شد.

برنامه سرویس‌دهنده آسنکرون

این برنامه‌ها بر اساس مفهوم Thread نوشته می‌شوند. ریسه یا Thread کوچک‌ترین رشته از دستورالعمل‌های یک نرم‌افزار برنامه‌ریزی شده است که به صورت مستقل می‌تواند توسط زمان‌بندی سیستم‌عامل مدیریت گردد.

برنامه‌های سرویس‌دهنده آسنکرون پس از آغاز پذیرش اتصال، به حالت بلوک‌شده نمی‌روند؛ بلکه یک Thread در این برنامه‌ها تعریف می‌شود که مسئول دریافت اتصال‌های جدید است. به همین لحاظ برنامه اصلی که در Thread دیگری اجرا می‌شود، قفل نخواهد شد. پس از دریافت اتصال، برای دریافت داده‌ها نیز از Thread استفاده می‌کنیم.

برای نوشتن برنامه سرویس‌دهنده آسنکرون، ابتدا تعاریف زیر را به صورت عمومی انجام می‌دهیم:

```
Socket MainListener;
byte[] buffer = new byte[1024];
```

مانند قبل، مشخصات سوکت و اتصال آن را تعریف و ارتباط لازم را برقرار می‌کنیم:

```
MainListener = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint server = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 1800);
MainListener.Bind(server);
```

اکنون دستوراتی می‌نویسیم که یک Thread برای پذیرش درخواست‌های اتصال ایجاد کنند:

```
AsyncCallback AcceptMethod = new AsyncCallback(AcceptCallBack);
MainListener.Listen(10);
MainListener.BeginAccept(AcceptMethod, MainListener);
```

این دستورات به سیستم‌عامل اعلام می‌کنند که در صورتی که روی سوکت MainListener درخواستی دریافت شد، تابع AcceptCallBack را اجرا کند. این کار به صورت آسنکرون انجام می‌شود و به همین دلیل، برنامه اصلی را از کار نمی‌اندازد. تابع AcceptCallBack به صورت زیر نوشته می‌شود:

```
private void AcceptCallBack(IAsyncResult ar)
{
   Socket temp = ((Socket)ar.AsyncState);
   Socket worker = temp.EndAccept(ar);


   string str = "Connected Client Socket is " +
worker.RemoteEndPoint.ToString();

   ShowClientInfo(str);

   temp.BeginAccept(AcceptCallBack, MainListener);

   AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
   worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
ReceiveMethod, worker);
}
```

در ابتدای این تابع، به صورت موقت عملیات پذیرش را متوقف می‌کنیم تا به اتصال دریافت شده رسیدگی کنیم و سپس مجدداً آن را در انتهای تابع راه‌اندازی می‌کنیم. با این کار، درخواست اتصال بعدی می‌تواند مجدداً Thread فوق را فعال کند.

برای دریافت داده‌ها از اتصال برقرار شده، یک Thread دیگر به نام ReceiveMethod تعریف می‌کنیم که پس از دریافت داده‌ها فعال می‌شود. قبل از توضیح تابع مربوط به این Thread، باید نکته‌ای را متذکر شویم. چون Thread برنامه اصلی و Thread پذیرش اتصال با هم متفاوت هستند، در برنامه اصلی نمی‌توان به داده‌های داخل Thread پذیرش اتصال دسترسی داشت. بنابراین، باید از یک تابع واسط بین این دو Thread استفاده کنیم و داده مورد نظر را به صورت پارامتر به آن منتقل کنیم. در کد بالا، از تابع ShowClientInfo برای این منظور استفاده شده است که رشته حاوی اطلاعات مشتری متصل شده به برنامه سرویس‌دهنده را به عنوان پارامتر می‌پذیرد و آن را نمایش می‌دهد. طریقه نوشتن این تابع به صورت زیر است:

```
delegate void ShowClientInfoCallBack(string msg);
private void ShowClientInfo(string msg)
{
   if (this.InvokeRequired)
   {
     ShowClientInfoCallBack d = new ShowClientInfoCallBack(ShowClientInfo);
     this.Invoke(d, new object[] { msg });
   }
   else
```

```
    MessageBox.Show(msg);
}
```

اکنون به Thread دریافت داده‌ها می‌پردازیم که در Thread پذیرش داده‌ها تعریف شد:

```
private void ReceiveCallBack(IAsyncResult ar)
{
  try
  {
    Socket worker = ((Socket)ar.AsyncState);
    int bytesReceived = worker.EndReceive(ar);
    string str = System.Text.UTF8Encoding.UTF8.GetString(buffer);

    ShowReceivedMsg(str);

    Array.Clear(buffer, 0, buffer.Length);
    AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
    worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
ReceiveMethod, worker);
  }
  catch (SocketException err)
  {
     MessageBox.Show(err.Message,"Server Error");
     Application.Exit();
  }
}
```

مانند Thread قبلی، در ابتدای تابع دریافت داده‌ها را متوقف کرده و پس از رسیدگی به داده‌های دریافت شده، در انتهای تابع مجدداً دریافت داده‌ها را فعال می‌کنیم.

تابع ShowReceivedMsg برای نمایش داده‌های دریافت شده به صورت زیر تعریف می‌شود:

```
delegate void ShowReceivedMsgCallBack(string msg);
private void ShowReceivedMsg(string msg)
{
  if (this.InvokeRequired)
  {
     ShowReceivedMsgCallBack d = new ShowReceivedMsgCallBack(ShowReceivedMsg);
     this.Invoke(d, new object[] { msg });
  }
  else
     MessageBox.Show(msg);
}
```

یک نمونه برنامه سرویس‌دهنده آسنکرون را می‌توانید از اینجا دریافت کنید. پس از انجام تنظیمات و فشار دادن دکمه «صدور مجوز ارتباط» برنامه مزبور برای پذیرش درخواست‌های اتصال آماده می‌شود؛ اما دیگر برنامه قفل نمی‌شود. اکنون هر متنی که از طرف یک برنامه مشتری برای این برنامه ارسال شود، در کادر پایین نمایش داده می‌شود. توجه کنید چون دریافت داده‌ها به کمک Threadها انجام شده است، دیگر نیازی به دکمه «به‌روزرسانی» که در برنامه سرویس‌دهنده سنکرون وجود داشت، نیست.

پرسش) اگر با Telnet به برنامه مزبور متصل شوید، هر کاراکتری که در آن محیط تایپ می‌کنید در کادر برنامه سرویس‌دهنده نمایش داده می‌شود. کد برنامه سرویس‌دهنده را طوری تغییر دهید که کل رشته دریافتی در کادر مزبور نمایش داده شود.

پرسش) نشان دهید چند برنامه مشتری می‌توانند به صورت همزمان به این برنامه سرویس‌دهنده متصل شوند و رشته‌های ارسالی هرکدام در برنامه سرویس‌دهنده نمایش داده خواهد شد.

پرسش) برنامه‌های سرویس‌دهنده و مشتری برای انتقال یک فایل تصویری را بنویسید. برای این کار در برنامه مشتری از متد SendFile استفاده کنید و در برنامه سرویس‌دهنده، آن‌قدر عملیات Recieve را تکرار کنید تا کل فایل دریافت شود (چون ممکن است برنامه مشتری فایل را قطعه‌قطعه کرده و در بسته‌های مجزا ارسال کند). برای اینکه از دریافت کامل فایل مطمئن شوید، می‌توانید طول دقیق (تعداد بایت‌های فایل) را قبل از ارسال فایل برای برنامه سرویس‌دهنده ارسال کنید تا برنامه مزبور بداند باید منتظر دریافت چند بایت باشد.