بسم الله الرحمن الرحیم

نام مقاله: سوکت نویسی در #C
نویسنده:علیرضا شیرازی
ویرایش و انتشار: PBN

در مدل کلاینت – سرور ، مبادلات زیر بین کلاینت و سرور رخ میدهد

۱ - سرور سوکتی را تعریف میکند

۲ - سرور سوکت را به یک IP که همان IP خودش است و یک پورت Bind میکند یا اختصاص میدهد

۳ - سرور به پورت گوش میدهد

۴ - کلاینت سوکتی را تعریف میکند و IP و پورت سرور را به آن اختصاص میدهد

۵ - کلاینت درخواست اتصال یا کانکت شدن به سرور را میدهد

۶ - سرور درخواست کلاینت را دریافت و آن را می پذیرد

۷ - کلاینت اطلاعاتی را ارسال می کند

۸ - سرور اطلاعات را می گیرد

۹ - سرور اطلاعات را ارسال میکند و کلاینت آن را میگیرد

۱۰ - سرور بسته میشود

۱۱ - کلانیت بسته میشود

ابتدا برنامه سمت سرور را مینویسم.

در این برنامه می بایست یک پورت را باز کرده و به آن گوش دهیم و دریافتی را نمایش دهیم.

ابتدا می بایست فضای نامهای زیر را با استفاده از using به کامپایلر سی شارپ معرفی کنیم:

کد:

```
System
System.Net
System.Net.Socket
System.Text
```

این کار را به این صورت انجام می دهیم:

کد:

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

اکنون متغیری به نام را به صورت سراسری و static تعریف میکنیم (از آنجایی که متد Main یک متد static است کلیه متغیرها و توابع مورد استفاده در آن نیز باید static باشند) .

متغیرها و متدهای Static را میتوان اینگونه تعریف کرد:

متغیرها و توابعی هستند که در یک کلاس به طور مشترک بین کلیه اشیاء گرفته شده از آن کلاس وجود دارند نه اینکه به ازای هر شیء یک نمونه از آن ایجاد شود. دسترسی به این متغیرها از طریق نام کلاس ممکن خواهد بود.

در مثال زیر کلاسی به نام Circle تعریف کرده ایم که دارای یک متغیر عادی به نام r است . این متغیر شعاع دایره را نگه میدارد.

متغیر PI به صورت static تعریف شده است چرا که عدد پی به ازای کلیه اشیاء این کلاس ثابت است.

در کلاس MainClass از کلاس Circle یک شیء ساخته ایم و از آن استفاده کرده ایم

کد:

```
class Circle
{
public static float PI =3.14;
public int r;

public static void PrintIt(double dbl)
{
Console.WriteLine(dbl);
}
}

class MainClass
{
static void Main(string[] args)
{
Circle cl = new Circle();
Console.WriteLine("Please Enter The Circle R");
cl.r = Convert.ToInt32( Console.ReadLine());

Circle.PrintIt(Circle.PI * cl.r * cl.r);
}
}
```

حال به مطلب اصلی خود بازگردیم:
یک شیء از کلاس سوکت به صورت سراسری و static ایجاد می کنیم:

کد:

```
static Socket sktListener;
```

در متد Main این سوکت را new مینکیم تا به آن حافظه اختصاص داده شود:

کد:

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

AddressFamily.InterNetworkبه معنای این است که از شبکه ای استفاده میکنیم که دارای IP نسخه ۴ است.
SocketType.Streamبرای سوکت هایی است که میخواهند به صورت Stream داده ها را تبادل کنند

ProtocolType.Tcpکه نوع پروتکل ما را مشخص میکند.

اکنون باید آدرس IP و یک Port به سوکت مان اختصاص دهیم:

کد:

```
IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any, 1800);
sktListener.Bind(ipLocal);
```

از آنجایی که این برنامه در سمت سرور اجرا میشود آدرس IP خاصی به آن نمیدهیم و پورت ۱۸۰۰ را باز میکنیم. کلاس IPEndPointبرای مشخص نمودن یک نود یا یک کامپیوتر در شبکه به کار میرود.

متد Bind نود مشخص شده را به سوکت اختصاص میدهد.

اکنون زمان گوش دادن به پورت است:

کد:

```
sktListener.Listen(100);
```

عدد ۱۰۰ نشانه آن است که حداکثر ۴ connection میتوانند در صف قرار گیرند.
اگر در این لحضه در command prompt دستور netstat –an را تایپ کنید میتوانید ببینید که پورت ۱۸۰۰ باز شده و در حال گوش دادن است.
حال میباید تقاضای کانکت شدن کلاینت را بپذیریم:

کد:

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

حال برای گرفتن داده ها ، می بایست یک بافرتعریف نماییم.
نکته : در سوکت پروگرمینگ ، داده ها به صورت آرایه ای از بایت ها منتقل می شوند. برای ارسال رشته های یونیکد و .... بایست آنها را کد گذاری کنیم . برای کد گذاری و کد گشایی از کلاس System.Text و متدهای آن استفاده کنیم .مثلا دستور زیر رشته salam را با فرمت Ascii به آرایه ای از بایت ها تبدیل میکند

کد:

```
byte[] byt = Encoding.ASCII.GetBytes("salam");
```

و متد زیر آن را رمزگشایی میکند:

کد:

```
string str = Encoding.ASCII.GetString(byt);
```

ما عمل رمزنگاری را موقع ارسال داده ها و عمل رمز گشایی را موقع دریافت آنها انجام میدهیم.
اکنون میخواهیم داده ها را دریافت کرده و رمز گشایی کنیم:

کد:

```
byte[] buffer = new byte[500];
sktListener.Receive(buffer);
string Data = Encoding.ASCII.GetString(buffer);
```

حال میتوانیم داده ها را پردازش کنیم:

کد:

```
Console.WriteLine(Data);
```

سوکت های آسنکرون

کدهایی که تا به اینجا دیدیم برای ایجاد سوکت های همگام یا سنکرون بوده است. این سوکت ها در برنامه های ویندوز و کلا سیستم های مالتی تسک کاربردی ندارند. چرا که بالفرض در زمانی از متد accept استفاده نموده ایم، در این حالت برنامه تا رسیدن یک سوکت به آن قفل شده و قادر به انجام کاری نیست.

در سوکت های آسنکرون از متدهای آسنکرون برای گوش دادن ، ارسال ، دریافت و ...استفاده میکنیم.در این مقاله ، یک برنامه سمنت سرور به صورت آسنکرون طراحی میکنیم که قادر به گوش دادن به یک کلاینت است.

نکته : قبل از ادامه ، آشنایی با delegate ها الزامی است. میتوانید delegateها را در مقاله مربوطه مشاهده فرمایید ولی اگر بخواهیم در یک جمله Delegateها را تعریف کنیم میتوانیم بگوییم:
delegateدر حقیقت چیزی نیست جز اشاره گر به تابع !

در سوکت های آسنکرون ، از delegate ای به نام AsyncCallback استفاده میکنیم. این Delegate به تابعی اشاره میکند که تنها یک آرگومان ورودی از نوع IAsyncResult دارد. متدهایی که به صورت آسنکرون کار میکنند ، اطلاعات مورد نظر خود را به صورت یک شی از این توع به تابع مورد نظر خود ارسال میکنند.

نکته : متدهای آسنکرون با پیشوندهای Beginو End شروع میگردند.

برای نوشتن برنامه ، ابتدا یک سوکت تعریف میکنیم که عمل گوش دادن را انجام دهد :

کد:

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

سپس عملیات معمول را بر روی سوکت انجام میدهیم:

کد:

```
IPEndPoint server = new IPEndPoint(IPAddress.Any, 1800);
Mainlistener.Bind(server);
```

همانگونه که میبینید ، در این برنامه سوکت مورد نظر ما به پورت ۱۸۰۰ گوش میدهد.

اکنون زمان آن است که یک delegate ایجاد کرده و آن را به تابع پردازشگر که در این مثال AcceptCallBack نام دارد ، منتسب کنیم .

کد:

```
AsyncCallback callBackMethod = new AsyncCallback(AcceptCallback);
```

اکنون باید سوکت تعریف شده به صورت غیر همگام)آسنکرون) شروع به گوش دادن به پورت کند:

کد:

```
Mainlistener.Listen(4);
Mainlistener.BeginAccept(AcceptCallback,Mainlistener);
```

در این مثال ، مشخص کرده ایم که سوکت شروع به عمل گوش دادن و انتظار کند و سپس به محض کانکت شدن یک کلاینت به کامپیوتر ما ، تابع AcceptCallBack اجرا گردد و به اموری که تعیین میکنیم رسیدگی کند.

نکته : پارامتر دوم تابع BeginAccept ، شیء ای است برای ارسال داده های وضعیت سوکت به تابعی که به سوکت رسیدگی میکند . در این جا این شیء خود سوکت است . اگر سوکت را به صورت سراسری تعریف میکردیم ، نیاز به ارسال این شیء نبود و به جای آن null قرار می دادیم. شیء مربوطه در قالب یک شیء از کلاس IAsyncResult ارسال خواهد شد.

تابع AcceptCallback بایستی اینگونه تعریف شود .

کد:

```
private void AcceptCallback(IAsyncResult ar)
{
...
{
```

تابع AcceptCallback بایستی اینگونه تعریف شود .

کد:

```
private void AcceptCallback(IAsyncResult ar)
{
...
}
```

در این تابع ، آرگومانی از نوع IAsyncResult وجود دارد. این آرگومان اطلاعات وضعیت فراخوان تابع آسنکرون که در اینجا یک سوکت است را نگهداری میکند. ابتدا این اطلاعات را استخراج میکنیم:

کد:

```
Socket temp = ((Socket)ar.AsyncState);
```

سپس به گوش دادن برای پذیرفتن کلاینت خاتمه میدهیم چرا که اکنون دیگر کلاینت مورد نظر به سرور کانکت شده و آماده برای ارسال اطلاعات است:

کد:

```
Socket worker = temp.EndAccept(ar);
```

نکته : دو دستور قبل را میتوانستیم در قالب یک دستور و به این شکل بنویسیم:

کد:

```
Socket temp = ((Socket)ar.AsyncState).EndAccept(ar);
```

بسیار خوب، اکنون که ارتباط کلاینت با برنامه ما برقرار گردیده است ، کافی است تا به صورت آسنکرون به دریافت اطلاعات مشغول شویم .باز هم مانند قسمت قبل، از متده‌ای آسنکرون استفاده کرده و تابعی تعیرف میکنیم که به محض دریافت اطلاعات فراخوانی گردیده و عملیات مورد نظر ما را انجام دهد .

نکته ای که بسیار حائز اهمیت است این که بایستی از یک بافر برای ذخیره اطلاعات دریافتی استافده کنیم. این بافر که در حقیقت آرایه ای از بایتها است به صورت یک آرایه سراسری تعریف میکنیم:

کد:

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

نکته : با توجه به این که در این برنامه صرفا یا یک کلاینت کار میکنیم ، سراسری بودن بافر مشکلی ایجاد نمیکند، اما چنانچه قصد داشتیم با چند کلاینت کار کنیم برای هریک میبایست بافر مخصوص به خودش را تعریف میکردیم که اصولا پیاده سازی آن برنامه به گونه ای دیگر خواهد بود که در مقالات آتی مورد بررسی قرار خواهد گرفت.

برای دریافت اطلاعات به صورت آسنکرون ، از متد BeginReceive استفاده میکنیم. البته بایستی بافر، اندیس اولیه ای که میخواهیم بافر از آنجا پر شود و همچنین اندیس حد نهایی بافر را مشخص کنیم.

کد:

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

پس تا اینجا ، متد AcceptCallback به این صورت است:

کد:

```
private void AcceptCallback(IAsyncResult ar)
{
 Socket temp = ((Socket)ar.AsyncState);
 Socket worker = temp.EndAccept(ar);
 AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
 worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,new
ReceiveMethod, worker);
}
```

اکنون متد **ReceiveCallBack** را تعریف میکنیم:

کد:
```
private void ReceiveCallBack(IAsyncResult ar)
{
...
{
```

درون این متد ، ابتدا اطلاعات وصعیت را بدست می آوریم:

کد:
```
Socket worker = ((Socket)ar.AsyncState);
```

سپس به گوش دادن به صورت موقت خاتمه میدهیم تا بتوانیم داده های فعلی را پردازش کنیم. این کار را با متد EndReceive انجام میدهیم . مقدار بازگشتی این متد تعداد بایتهای دریافت شده میباشد:

کد:
```
int bytesReceived = worker.EndReceive(ar);
```

حال میبایست اطلاعات دریافت شده که به صورت آرایه ای بایتها درون بافر هستند را پردازش کرده و جهت نمایش به رشته (string)تبدیل کنیم:

کد:
```
string str = System.Text.UTF8Encoding.UTF8.GetString(buffer);
```

حال میتوانیم اطلاعات را نمایش دهیم . نکته مهمی که اینجا وجود دارد این است که در برنامه نویسی به شیوه آسنکرون دیگر نمیتوانیم به صورت عادی این رشته را به یک TextBox و یا کلا یک کنترل بفرستیم . دلیل این امر این است که تنها آن threadای که کنترل مورد نظر را در ابتدای برنامه ایجاد کرده است میتواند با کنترل کار کرده و اطلاعات لازم را به آن بفرستد . پس کاری که انجام میدهیم استفاده از یک متد کمکی است که ابتدا threadایجاد کننده کنترل را پیدا کرده و سپس کار مورد نظر را با استفاده از آن انجام میدهد . در این مثال نام این تابع را ShowInfo قرار داده ایم و تنها یک پارامتر از نوع رشته ای به آن ارسال میکنیم . البته فعلا دانستن جزئیات این متد چندان لازم نیست:

کد:
```
delegate void ShowInfoCallback(string text);
private void ShowInfo(string text)
{
 if (this.txtMain.InvokeRequired)
  {
   ShowInfoCallback d = new ShowInfoCallback(ShowInfo);
    this.Invoke(d, new object[] { text });
   {
  else
  {
    txtMain.Text+=text+"\n";

  }

 }
```

نکته : در برنامه اصلی فرض کرده ایم ، کنترلی که قرار است اطلاعات را نمایش دهد یک RichTextBox به نام txtMain میباشد.

بسیار خب ، تاکنون توانستیم اطلاعات را دریافت کرده و به صورت یک رشته در آوریم. سپس آن را پردازش کنیم . اگر به یاد داشته باشد برای پردازش کردن اطلاعات ، عمل دریافت اطلاعات را متوقف کرده ایم. اکنون این عمل را برای دریافت ادامه اطلاعات از سر میگیریم:

کد:

```
AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);

worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
ReceiveMethod, worker);
```

پس متد ReceiveCallBack به این صورت است:

کد:

```
private void ReceiveCallBack(IAsyncResult ar)
{
  Socket worker = ((Socket)ar.AsyncState);
  int bytesReceived = worker.EndReceive(ar);
  string str = System.Text.UTF8Encoding.UTF8.GetString(buffer);
  ShowInfo(str);
  AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
  worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
 ReceiveMethod, worker);
 }
```

متن کامل برنامه:

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

public Form1()
{
            InitializeComponent();

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

 IPEndPoint server = new IPEndPoint(IPAddress.Any, 1800);
 Mainlistener.Bind(server);

 AsyncCallback callBackMethod = new AsyncCallback(AcceptCallback);

 Mainlistener.Listen(4);
 Mainlistener.BeginAccept(AcceptCallback,Mainlistener);
}

private void AcceptCallback(IAsyncResult ar)
{
            Socket temp = ((Socket)ar.AsyncState);
            Socket worker = temp.EndAccept(ar);
            AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
            worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
ReceiveMethod, worker);
}

private void ReceiveCallBack(IAsyncResult ar)
{
            Socket worker = ((Socket)ar.AsyncState);
            int bytesReceived = worker.EndReceive(ar);

            string str = System.Text.UTF8Encoding.UTF8.GetString(buffer);
            ShowInfo(str);
            AsyncCallback ReceiveMethod = new AsyncCallback(ReceiveCallBack);
            worker.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
ReceiveMethod, worker);
}

delegate void ShowInfoCallback(string text);
private void ShowInfo(string text)
{
            if (this.txtMain.InvokeRequired)
            {
                ShowInfoCallback d = new ShowInfoCallback(ShowInfo);
                this.Invoke(d, new object[] { text });
            }
            else
            {
                txtMain.Text+=text+"\n";

            }

}
```

برای تست کردن برنامه میتوانیم از Telnet استفاده کنیم .یکی از امکانات ویندوز است که به کمک ان میتوان به یک آدرس IP و یک شماره پورت خاص وصل شده و به ارسال اطلاعات مشغول شویم. کافی است ایتدا برنامه را اجرا کرده و سپس در منوی RUN تایپ کنید:
کد:

```
telnet 127.0.0.1 1800
```

سپس در پنجره مشکی رنگ ظاهر شده ، شروع به تاپیک کردن کنید . آنچه شما تایپ میکنید، برای برنامه ارسال میگردد و میتوانید آن را مشاهد فرمایید .

نکته مهم این است که چنانچه برنامه TELNET را پیش از بستن برنامه سرور که نوشته ایم ببندید ، برنامه ما در یک loop می افتد و آخرین اطلاعات ارسال شده را به صورت مداوم تکرار میکند !!! البته این مشکل از برنامه ما نیست و به دلیل عملکرد برنامه Telnetاین اتفاق می افتد. در مقاله بعدی یاد خواهیم گرفت که برنامه سمت کلاینت را نیز به صورت آسنکرون بنویسیم و بنابراین نیازی به Telnet نخواهیم داشت .