بسم الله الرحمن الرحیم

Socket Programming

به کمک C#

گردآورنده: بابک لونی

دانشکده ریاضی و علوم کامپیوتر دانشگاه صنعتی امیرکبیر

دی ماه ۱۳۸۶

چکیده

در این مستند به شرح چگونگی طراحی و پیاده سازی یک نرم افزار ساده به کمک تکنیک های برنامه نویسی سوکت در زبان برنامه نویسی C# پرداخته ایم. مهم ترین مفاهیمی که از آنها برای پیاده سازی پروژه مورد استفاده قرار گرفته است پروتکل TCP/IP و مفاهیم سوکت بوده که در این مستند ابتداعاً مروری بر این مفاهیم داشته ایم سپس از این مفاهیم برای پیاده سازی یک کد ساده بهره جسته ایم.

در ادامه مقدمه ای بر برنامه نویسی شبکه به خصوص در C# و تکنولوژی .NET. داشته ایم سپس قسمت های اصلی پیاده سازی را به همراه کد به کار رفته در آنها شرح داده ایم.

فهرست

مفاهیم اولیه پروتکلTCP/IP

TCP/IP ، یکی از مهمترین پروتکل های استفاده شده در شبکه های کامپیوتری است . اینترنت بعنوان بزرگترین شبکه موجود ، از پروتکل فوق بمنظور ارتباط دستگاه های متفاوت استفاده می نماید. پروتکل ، مجموعه قوانین لازم بمنظور قانونمند نمودن نحوه ارتباطات در شبکه های کامپیوتری است. در این بخش مواردی همچون : فرآیند انتقال اطلاعات ، معرفی و تشریح لایه های پروتکل TCP/IP و نحوه استفاده از سوکت برای ایجاد تمایز در ارتباطات ، تشریح می گردد.

امروزه اکثر شبکه های کامپیوتری بزرگ و اغلب سیستم های عامل موجود از پروتکل TCP/IP ، استفاده و حمایت می نمایند. TCP/IP ، امکانات لازم بمنظور ارتباط سیستم های غیرمشابه را فراهم می آورد. از ویژگی های مهم پروتکل فوق ، می توان به مواردی همچون : قابلیت اجراء بر روی محیط های متفاوت ، ضریب اطمینان بالا، قابلیت گسترش و توسعه آن ، اشاره کرد . از پروتکل فوق، بمنظور دستیابی به اینترنت و استفاده از سرویس های متنوع آن نظیر وب و یا پست الکترونیکی استفاده می گردد. تنوع پروتکل های موجود در پشته TCP/IP و ارتباط منطقی و سیستماتیک آنها با یکدیگر، امکان تحقق ارتباط در شبکه های کامپیوتری را با اهداف متفاوت ، فراهم می نماید. فرآیند برقراری یک ارتباط ، شامل فعالیت های متعددی نظیر : تبدیل نام کامپیوتر به آدرس IP معادل ، مشخص نمودن موقعیت کامپیوتر مقصد ، بسته بندی اطلاعات ، آدرس دهی و روتینگ داده ها بمنظور ارسال موفقیت آمیز به مقصد مورد نظر ، بوده که توسط مجموعه پروتکل های موجود در پشته TCP/IP انجام می گیرد.

معرفی پروتکل TCP/IP

TCP/IP ، پروتکلی استاندارد برای ارتباط کامپیوترهای موجود در یک شبکه مبتنی بر ویندوز ۲۰۰۰ است. از پروتکل فوق، بمنظور ارتباط در شبکه های بزرگ استفاده می گردد. برقراری ارتباط از طریق پروتکل های متعددی که در چهارلایه مجزا سازماندهی شده اند ، میسر می گردد. هر یک از پروتکل های موجود در پشته TCP/IP ، دارای وظیفه ای خاص در این زمینه (برقراری ارتباط) می باشند . در زمان ایجاد یک ارتباط ، ممکن است در یک لحظه تعداد زیادی از برنامه ها ، با یکدیگر ارتباط برقرار نمایند. TCP/IP ، دارای قابلیت تفکیک و تمایز یک برنامه موجود بر روی یک کامپیوتر با سایر برنامه ها بوده و پس از دریافت

داده ها از یک برنامه ، آنها را برای برنامه متناظر موجود بر روی کامپیوتر دیگر ارسال می نماید. نحوه ارسال داده توسط پروتکل TCP/IP از محلی به محل دیگر ، با فرآیند ارسال یک نامه از شهری به شهر، قابل مقایسه است .

برقراری ارتباط مبتنی بر TCP/IP ، با فعال شدن یک برنامه بر روی کامپیوتر مبدا آغاز می گردد . برنامه فوق ،داده های مورد نظر جهت ارسال را بگونه ای آماده و فرمت می نماید که برای کامپیوتر مقصد قابل خواندن و استفاده باشند. (مشابه نوشتن نامه با زبانی که دریافت کننده ، قادر به مطالعه آن باشد) . در ادامه آدرس کامپیوتر مقصد ، به داده های مربوطه اضافه می گردد (مشابه آدرس گیرنده که بر روی یک نامه مشخص می گردد) . پس از انجام عملیات فوق ، داده به همراه اطلاعات اضافی (درخواستی برای تائید دریافت در مقصد) ، در طول شبکه به حرکت درآمده تا به مقصد مورد نظر برسد. عملیات فوق ، ارتباطی به محیط انتقال شبکه به منظور انتقال اطلاعات نداشته ، و تحقق عملیات فوق با رویکردی مستقل نسبت به محیط انتقال ، انجام خواهد شد .

لایه های پروتکل TCP/IP

TCP/IP ، فرآیندهای لازم بمنظور برقراری ارتباط را سازماندهی و در این راستا از پروتکل های متعددی در پشته TCP/IP استفاده می گردد. بمنظور افزایش کارآئی در تحقق فرآیند های مورد نظر، پروتکل ها در لایه های متفاوتی، سازماندهی شده اند . اطلاعات مربوط به آدرس دهی در انتها قرار گرفته و بدین ترتیب کامپیوترهای موجود در شبکه قادر به بررسی آن با سرعت مطلوب خواهند بود. در این راستا، صرفا کامپیوتری که بعنوان کامپیوتر مقصد معرفی شده است ، امکان باز نمودن بسته اطلاعاتی و انجام پردازش های لازم بر روی آن را دارا خواهد بود. TCP/IP ، از یک مدل ارتباطی چهار لایه بمنظور ارسال اطلاعات از محلی به محل دیگر استفاده می نماید: Internet, Transport, Application و Network Interface ، لایه های موجود در پروتکل TCP/IP می باشند. هر یک از پروتکل های وابسته به پشته TCP/IP ، با توجه به رسالت خود ، در یکی از لایه های فوق، قرار می گیرند که توضیح کارکرد این لایه ها در این مقال نمی گنجد.

مشخص نمودن برنامه ها

در شبکه های کامپیوتری ، برنامه ها ی متعددی در یک زمان با یکدیگر مرتبط می گردند. زمانیکه چندین برنامه بر روی یک کامپیوتر فعال می گردند ، TCP/IP ، می بایست از روشی به منظور تمایز یک برنامه از

برنامه دیگر، استفاده نماید. بدین منظور ، از یک سوکت (Socket) بمنظور مشخص نمودن یک برنامه خاص ، استفاده می گردد.

آدرس IP

برقراری ارتباط در یک شبکه ، مستلزم مشخص شدن آدرس کامپیوترهای مبداء و مقصد است (شرط اولیه به منظور برقراری ارتباط بین دو نقطه ، مشخص بودن آدرس نقاط درگیر در ارتباط است) . آدرس هر یک از دستگاه های درگیر در فرآیند ارتباط ، توسط یک عدد منحصربفرد که IP نامیده می شود ، مشخص می گردند. آدرس فوق به هریک از کامپیوترهای موجود در شبکه نسبت داده می شود . IP : ۱۰. ۱۰.۱.۱ ، نمونه ای در این زمینه است .

پورت TCP/UDP

پورت مشخصه ای برای یک برنامه و در یک کامپیوتر خاص است. پورت با یکی از پروتکل های لایه "حمل" (TCP و یا UDP) مرتبط و پورت TCP و یا پورت UDP ، نامیده می شود. پورت می تواند عددی بین صفر تا ۶۵۵۳۵ را شامل شود. پورت ها برای برنامه های TCP/IP سمت سرویس دهنده ، بعنوان پورت های "شناخته شده " نامیده شده و به اعداد کمتر از ۱۰۲۴ ختم و رزرو می شوند تا هیچگونه تعارض و برخوردی با سایر برنامه ها به وجود نیاید. مثلا برنامه سرویس دهنده FTP از پورت TCP بیست و یا بیست ویک استفاده می نماید.

سوکت (Socket)

سوکت ، ترکیبی از یک آدرس IP و پورت TCP ویا پورت UDP است . یک برنامه ، سوکتی را با مشخص نمودن آدرس IP مربوط به کامپیوتر و نوع سرویس (TCP برای تضمین توزیع اطلاعات و یا UDP) و پورتی که نشاندهنده برنامه است، مشخص می نماید. آدرس IP موجود در سوکت ، امکان آدرس دهی کامپیوتر مقصد را فراهم و پورت مربوطه ، برنامه ای را که داده ها برای آن ارسال می گردد را مشخص می نماید.

برنامه نویسی شبکه

شبکه ها (و برنامه نویسی شبکه) از راه درازی متجاوز از ۲۰ سال پیش می آیند. در اوایل محاسبات تحت شبکه (در دهه ۸۰)، برنامه نویسی شبکه به برنامه نویسان پیشرفته که اساسا کاربردها را با استفاده از برنامه نویسی به زبان C در محیط Unix اجرا می کردند، واگذار شده بود. امروزه شبکه ها همه جا هستند، از شرکتهای کوچک گرفته تا کاربران کوچک خانگی. به وسیله چندین کامپیوتر متصل به یکدیگر از طریق شبکه ها، کاربردهای اطلاعی شبکه الزمات پذیرفته شده هستند. کاربردهای موجود بایستی ویژگیهای شبکه را برای رقیب باقی ماندن در بازار ترکیب کنند و اضافه کردن ارتباط شبکه با کاربردها ضروری است. برنامه های شبکه برای هر کاری از بازیهای کودکان گرفته تا سیستمهای پایگاه داده شرکتهای پیشرفته استفاده می شوند .

برنامه نویسی شبکه همواره یک ویژگی کلیدی سیستم عاملهای Windows مایکروسافت بوده است. متأسفانه شما مجبورید مفاهیم پیشرفته برنامه نویسی C یا C++ را برای بکار بردن ویژگیهای برنامه نویسی شبکه در برنامه های Windows بدانید. حال با این وجود زبانهای NET Framework. کار اضافه کردن ویژگیهای شبکه به کاربردهای شما را آسانتر می کنند کتابخانه های NET. کلاسهای شبکه فراوانی که می توانند برنامه نویسی شبکه را جمع کنند، فراهم می کند .

مدیریت شبکه و نیازهای امنیتی امروزی ، شبکه را ذاتا ملزم به ارتباط با وسایل شبکه و پیگیری ایستگاه های کاری در شبکه کرده اند. تلاش برای نگارش سریع کد شبکه بی نقص، وقتی که شما با ساختار API های سوکتی C کار می کنید، می تواند سخت باشد و اجرای کاربردهای Java اغلب به علت کند کردن سرعت پردازش و ناچیز بودن پشتیبانیWindows ، تجربه دردناکی می باشند .

زبان C# بسیاری از مشکلات برنامه نویسی شبکه را با اجازه تهیه سریع یک مدل اولیه و گسترش دادن کاربردهای شبکه با استفاده از کلاسهای C# حل کرده است. الحاق کتابخانه فرمهای C# برای نوشتن کدهای گرافیکی با کتابخانه سوکت C# برای نوشتن کد شبکه سازی کاربردهای ایجاد شبکه حرفه ای را ساده می کند. بوسیله کلاسهای شبکه C# ، نوشتن کدهایی که اغلب نوشتنشان یک روز طول می کشد، کمتر از یک ساعت نوشته می شوند.

مراحل پیاده سازی برنامه نویسی سوکت در C#

طریقه اتصال

به طور کلی برای برنامه نویسی سوکت در مدل کلاینت – سرور ، مبادلات زیر بین کلاینت و سرور رخ میدهد:

۱. سرور سوکتی را تعریف می کند
۲. سرور سوکت را به یک IP که همان IP خودش است و یک پورت Bind میکند یا اختصاص می دهد
۳. سرور به پورت گوش می دهد
۴. کلاینت سوکتی را تعریف میکند و IP و پورت سرور را به آن اختصاص می دهد
۵. کلاینت درخواست اتصال یا کانکت شدن به سرور را می دهد
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 می گیرد
۱۰. سرور بسته می شود
۱۱. کلانیت بسته می شود

پیاده سازی اصول اولیه

حال که ۱۱ مرحله اصلی لازم برای اتصال بین کلاینت و سرور را توضیح دادیم سعی می کنیم مراحل گفته شده را در C# پیاده سازی کنیم. در این مستند پیاده سازی را یک بار به کمک سوکت های سنکرون و بار دیگر به کمک سوکت های آسنکرون انجام داده و مراحل را نشان می دهیم.

پیاده سازی با سوکت های سنکرون

ابتدا برنامه سمت سرور را می نویسم.

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 دستور using به کامپایلر سی شارپ معرفی کنیم:

```
System.Net
System.Net.Socket
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 یا توابع ایستا 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.

به کمک دستور زیر یک شیء از کلاس سوکت به صورت سراسری و static ایجاد می کنیم:

```
static Socket sktListener;
```

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

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

AddressFamily.InterNetworkبه معنای این است که از شبکه ای استفاده می کنیم که دارای IP نسخه ۴ است.

SocketType.Streamبرای سوکت هایی است که می خواهند به صورت Stream داده ها را تبادل کنند و ProtocolType.Tcp نوع پروتکل ما را مشخص می کند.

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

```
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 نمی دهیم و پورت ۱۸۰۰ را باز میکنیم. کلاس IPEndPoint برای مشخص نمودن یک نود یا یک کامپیوتر در شبکه به کار می رود.

اکنون زمان گوش دادن به پورت است:

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

عدد ۱۰۰ نشانه آن است که حداکثر ۴ connection می توانند 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 می باید تقاضای کانکت شدن کلاینت را بپذیریم:

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

حال برای گرفتن داده ها ، می بایست یک بافر تعریف 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

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

و متد زیر آن را رمزگشایی میکند:

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

ما عمل رمزنگاری را موقع ارسال داده ها و عمل رمز گشایی را موقع دریافت آنها انجام می دهیم.
اکنون می خواهیم داده ها را دریافت کرده و رمز گشایی کنیم:

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

حال میتوانیم داده ها را پردازش کنیم:

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

پیاده سازی با 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 می کنیم. در این مستند ، یک برنامه سمت 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. ولی اگر بخواهیم در یک جمله Delegate ها را تعریف کنیم میتوانیم بگوییم delegate در 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 می کنند ، 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 :

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

سپس عملیات معمول را بر روی سوکت انجام میدهیم:

```
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 AcceptCallBackنام دارد ، منتسب کنیم .

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

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

```
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 AcceptCallBackاجرا گردد و به اموری که تعیین می کنیم رسیدگی کند.

نکته : پارامتر دوم تابع BeginAccept ، شیء ای است برای ارسال داده های وضعیت سوکت به تابعی که به سوکت رسیدگی می کند . در این جا این شیء خود سوکت است . اگر سوکت را به صورت سراسری تعریف می کردیم ، 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 .

```
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 می کند. ابتدا این اطلاعات را استخراج می کنیم:

```
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:

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

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

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

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 تعریف می کنیم 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 استفاده کنیم. این بافر که در حقیقت آرایه ای از بایتها است به صورت یک آرایه سراسری تعریف میکنیم:

```
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 نمی کند، اما چنانچه قصد داشتیم با چند کلاینت کار کنیم برای هریک می بایست بافر مخصوص به خودش را تعریف میکردیم که اصولا پیاده سازی آن برنامه به گونه ای دیگر خواهد بود.

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

```
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:

```
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 می کنیم:

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

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

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

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

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

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

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

در تکمیل مطالب گفته شده در اینجا یک مثال ساده آورده ایم که یک Server و یک Client را برای شما می سازد و با استفاده از Threadingبه ازای هر اتصال از سمت کلاینت یک Thread ایجاد می کند:

```
class TcpServer
{
   private TcpListener _tcpListener;


   public TcpServer(int port)
   {
      _tcpListener = new TcpListener(port);
   }

   public void BeginToListen(int intMaxconn)
   {
      _tcpListener.Start(intMaxconn);

      while (true)
      {
         if (!_tcpListener.Pending())
         {
            Console.WriteLine("Waiting for new request...");
            Thread.Sleep(1000);
         }
         else
         {
            Console.WriteLine("Recieve new pending..");
            ClientConnectionThread clientConn = new ClientConnectionThread(_tcpListener);
            Thread thread = new Thread(new ThreadStart(clientConn.HandleConnection));
            thread.Start();
         }
      }
   }
}
```

یک کد ساده

حال که مراحل اصلی برنامه نویسی سوکت شرح داده شد، در ادامه یک کد ساده به زبان C# برای ارسال متن در یک شبکه محلی با استفاده از مطالب گفته شده آورده شده است.

مهم ترین کلاس به کار رفته در این کد، کلاس TcpListener است که یکی از مهم ترین وظایف ارتباط بین دو کامپیوتر را با استفاده از پروتکل TCP را فراهم می نماید. همان گونه که در مطالب قبلی نیز ذکر شد باید یک شی از این کلاس تعریف نماییم که این شی به کمک یک آدرس IP و یک پورت به پیغام های رسیده از آدرس و از آن پورت گوش داده و منتظر برقراری ارتباط می ماند.

نمونه کد زیر به کمک این کلاس به پیام های رسیده از پورت ۱۳۰۰۰ از آدرس Localhost گوش می کند و ارتباط با سایر کلاینت ها را برقرار می کند:

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

class MyTcpListener
{
  public static void Main()
  {
    TcpListener server=null;
    try
    {
      // Set the TcpListener on port 13000.
      Int32 port = 13000;
      IPAddress localAddr = IPAddress.Parse("127.0.0.1");

      // TcpListener server = new TcpListener(port);
      server = new TcpListener(localAddr, port);

      // Start listening for client requests.
      server.Start();

      // Buffer for reading data
```

```
Byte[] bytes = new Byte[256];
String data = null;

// Enter the listening loop.
while(true)
{
  Console.Write("Waiting for a connection... ");

  // Perform a blocking call to accept requests.
  // You could also user server.AcceptSocket() here.
  TcpClient client = server.AcceptTcpClient();
  Console.WriteLine("Connected!");

  data = null;

  // Get a stream object for reading and writing
  NetworkStream stream = client.GetStream();

  int i;

  // Loop to receive all the data sent by the client.
  while((i = stream.Read(bytes, 0, bytes.Length))!=0)
  {
    // Translate data bytes to a ASCII string.
    data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
    Console.WriteLine("Received: {0}", data);

    // Process the data sent by the client.
    data = data.ToUpper();

    byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);

    // Send back a response.
    stream.Write(msg, 0, msg.Length);
    Console.WriteLine("Sent: {0}", data);
  }

  // Shutdown and end connection
  client.Close();
}
```

```
  }
  catch(SocketException e)
  {
    Console.WriteLine("SocketException: {0}", e);
  }
  finally
  {
     // Stop listening for new clients.
     server.Stop();
  }


  Console.WriteLine("\nHit enter to continue...");
  Console.Read();
}
```

ضمیمه: معرفی مختصر فضاهای نامی استفاده شده

System.Net ، شامل نوع هائی بمنظور دستیابی به چندین پروتکل متداول نظیر: HTTP و DNS است . namespace فوق ، همچنین شامل کلاس های WebRequest و WebResponse بوده که امکان ایجاد برنامه هائی را فراهم می نماید که قادرند مستقل از نوع پروتکل استفاده شده ، با یکدیگر ارتباط برقرار نمایند . این نوع برنامه ها می توانند بسادگی درخواستی را ایجاد و پاسخ آن را از طریق یک URL بدون آگاهی از جزئیات مربوطه ، دریافت نمایند. System.Net ، همچنین شامل namespace زیرمجموعه System.Net.Sockets است . نوع ها ی موجود در namespace فوق ، یک پیاده سازی مدیریت یافته از اینترفیس های مبتنی بر سوکت سنتی در ارتباط با TCP و یا UDP مربوط به WinSock برای تولیدات مبتنی بر دات نت ، می باشد .System.Net ، امکانات حمایتی لازم بمنظور دستیابی به HTTP,TCP و سایر پروتکل ها را فراهم می نماید.

System.IO، مجموعه ای گسترده از نوع ها را بمنظور خواندن و نوشتن فایل ها و دایرکتوری ها ، ارائه می نماید. با اینکه دستیابی به سیستم های مدیریت بانک اطلاعاتی ، اغلب بعنوان اصلی ترین رویکرد بمنظور دستیابی به داده ها مطرح می گردد ، ولی امکان انجام عملیات مرتبط با فایل ها ، همچنان مفید و ضروری خواهد بود.

System.Threading، یکی دیگر از namespaceهای مهم فریمورک دات نت محسوب می گردد. نوع های موجود در namespaceفوق ، روشی استاندارد برای پیاده کنندگانی که با هر یک از زبانهای دات نت کار می نمایند را بمنظور ایجاد برنامه های multithreadedارائه می نماید. قبل از دات نت ، ویژوال بیسیک ، C++ و سایر زبانها جملگی دارای رویکرد مختص بخود بمنظور threading بودند. در فریمورک دات نت تمامی زبان های مبتنی بر CLR، قادر به استفاده از محتویات System.Threading بمنظور کار بار threads بصورت یکپارچه می باشند . شاید مهمترین نوع در این namespace، کلاس Thread باشد که متدهای لازم بمنظور شروع فعالیت یک thread، توقف آن و سایر عملیات مربوطه را ارائه می نماید.