برنامه نویسی سوکت

مهرداد قدیری

مقدمه

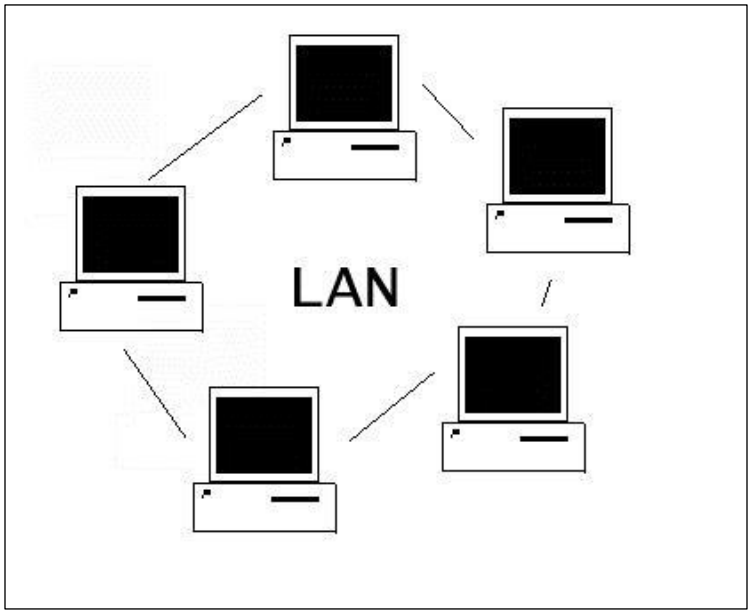
همه ما انسانها در طول زندگی خود روش های ارتباطی زیادی را تجربه کرده‌ایم. هنگامی که در یک کلاس درس حاضر می شویم و به صحبت های استاد گوش می دهیم به نوعی اقدام به ایجاد ارتباط و تعامل برای اهداف مشخصی (مانند بهره وری از دانسته های استاد) می کنیم. این مثال ساده نشان دهنده نوعی ارتباط انسانی است که در داخل آن می توان اجزای مهم ارتباطی از جمله زبان و خط استاندارد شده را در بین دانشجو و استاد در نظر گرفت. این زبان و خط استاندارد شده که فرآیند طولانی را در دوره ی حیات بشر تاکنون طی کرده است، توجه ذاتی و فطری انسان‌ها به ارتباطات را نشان می دهد. انسان این نیاز ذاتی خود به ارتباطات و تعامل را به سایر حوزه های کاری خود از جمله علوم مهندسی نیز کشانده است که در این میان علم کامپیوتر توجه ویژه ای به مقوله ی ارتباطات دارد. پیشرفت روز افزون علم کامپیوتر که منجر به افزایش توانایی کامپیوترها در پردازش و ذخیره سازی اطلاعات شده است، بدون ایجاد یک ارتباط کارآمد بین کامپیوترها ناقص خواهد ماند. از این رو دانشمندان و فعالان حوزه ی کامپیوتر اقدام به طراحی و استاندارد سازی هر چه بیشتر ابزار و وسایل و پروتکل های ارتباطی نموده اند که به کمک آنها می توان اقدام به دسترسی به داده ها و منابع و به اشتراک گذاری سودمند و هدفمند آنها نمود. در بخش اول به توضیحی اجمالی پیرامون برخی از مباحث پایه ای شبکه های کامپیوتری می پردازیم.

۱.شبکه های کامپیوتری

شبکه های کامپیوتری با هدف به اشتراک گذاری اطلاعات و منابع، دسترسی به آنها را حتی از راه دور میسر می سازند. این شبکه ها با پیشرفت روزافزون خود بیشتر جنبه های زندگی ما را تحت تاثیر قرار داده اند به طوری که نمی توان بدون آنها زندگی کرد حتی اگر به طور مستقیم کاربر آنها نباشیم و حضور آنها را حس نکنیم. به طور مثال وقتی اقدام به ارسال یک نامه الکترونیکی می کنیم از مجموعه ای از پروتکل ها بهره می گیریم و پیام خود را از طریق تعداد زیادی تجهیزات شبکه به مقصد مورد نظر می رسانیم. و یا هنگامی که یک صورت حساب را در بانک توسط متصدی آن پرداخت می کنیم شاید حضور شبکه های کامپیوتری برای ما کمرنگتر باشد اما به هر حال این عمل هم توسط مجموعه ای از پروتکل ها و ابزار و وسایل ارتباطی شبکه محقق می شود. در ادامه به بررسی شبکه های کامپیوتری از منظر مقیاس جغرافیایی می پردازیم.

۱-۱.شبکه های کامپیوتری از منظر مقیاس جغرافیایی

شبکه های کامپیوتری را می توان از دید مقیاس جغرافیایی به سه دسته شبکه های محلی(LAN[۱])، شبکه های بین شهری(MAN[۲]) و شبکه های گسترده(WAN[۳]) تقسیم نمود. هر یک از این شبکه ها تجهیزات مورد نیاز خود را به همراه دارند که برای آن نوع ارتباط در آن مقیاس مورد استفاده قرار می گیرد. واضح است که بسیاری از تجهیزات مورد استفاده در شبکه های گسترده در شبکه های محلی بدون کاربرد است و بلعکس. شبکه های محلی بیشتر برای ایجاد شبکه ای از ایستگاه های کاری[۴] و منابع دیگر در یک مکان کوچک مثلا در یک ساختمان استفاده می شوند. در این شبکه ها معمولا تعداد کامپیوتر ها و منابع کم بوده و براحتی از یک محل قابل مدیریت می باشند.



شکل ۱-۱ شبکه محلی

شبکه های بین شهری بیشتر به منظور ایجاد ارتباط بین دو یا چند مکان در یک شهر استفاده می شوند. به طور مثال اگر سازمانی قصد دارد شعبات خود را که در مکان های متفاوتی از شهر قرار دارند را به یکدیگر متصل کند، از این رویکرد استفاده می کند. این نوع شبکه ها معمولا از تعداد ایستگاه های کاری بیشتری تشکیل شده اند و از تجهیزات ویژه ای استفاده می کنند که به مراتب هزینه راه اندازی و نگهداری آن بیشتر از شبکه های محلی می باشد. همچنین در این شبکه ها ممکن است برای کاهش هزینه ها از تجهیزات بیسیم استفاده شود.

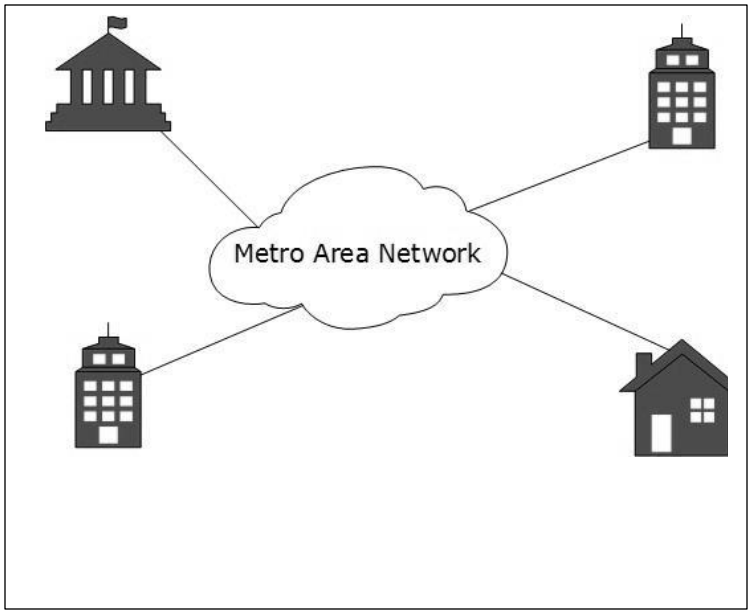
---

[۱] Local Area Network

[۲] Metropolitan Area Network

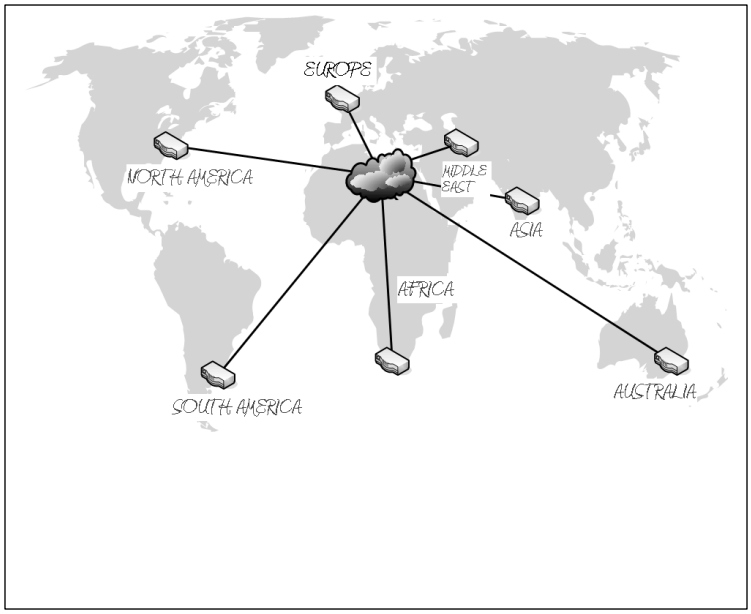
[۳] Wide Area Network

[۴] Work Station



شکل ۲-۱ شبکه بین شهری

شبکه های گسترده چیزی بیشتر از اتصال چند کامپیوتر و یا چند شعبه از یک سازمان می باشند. این شبکه ها بیشتر برای ایجاد ارتباط میان چند شهر، چند کشور و یا حتی چندین قاره می باشد که معمولا توسط سازمانهای تامین زیرساخت از سوی خود دولت ها صورت می گیرد. این شبکه ها و در راس آنها اینترنت وظیفه ی سرویس دهی به تعداد بسیار زیادی از کاربران را در نقاط مختلف دنیا را بر عهده دارند.

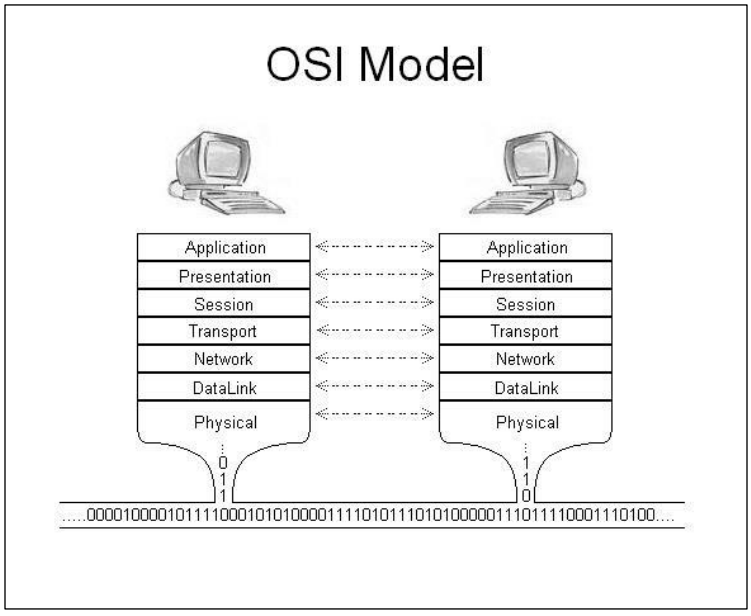


شکل ۳-۱ شبکه گسترده

حال در ادامه قصد داریم به دو دسته بندی مهم در حوزه شبکه های کامپیوتری اشاره کنیم که هر کدام به توضیح و بخش بندی لایه ای شبکه می پردازند. هر یک از این لایه ها شامل پروتکل هاییست که وظیفه ای را بر عهده دارند و خدمتی را برای لایه بالاتر خود ارائه می‌دهند.

۲-۱.مدل هفت لایه شبکه OSI

مدل هفت لایه OSI[۱] که در شکل ۴-۱ قابل مشاهده است مدل استانداردی از شبکه های کامپیوتریست اما طراحان شبکه به این مدل توجه شایانی نکرده اند و در عمل از مدل دیگری که در ادامه به توضیح آن خواهیم پرداخت، بهره برده اند. در اینجا به یک توضیح اجمالی پیرامون هر یک از این لایه ها بسنده می کنیم.



شکل ۴-۱ مدل هفت لایه OSI

۱-۲-۱.لایه فیزیکی

همانطور که در شکل پیداست لایه فیزیکی[۲] با داده های بیتی سروکار داشته و هیچ اطلاعی در رابطه با محتوای پیام ارسالی یا دریافتی ندارد. وظیفه این لایه دریافت اطلاعات و تبدیل آنها به سیگنال الکتریکی و ارسال آنها بر روی کانال ارتباطی با نرخ بیت[۳] و فرکانس مشخص است.

[۱] Open System Interconnection

[۲] Physical Layer

[۳] Bit Rate

۲-۲-۱.لایه پیوند داده

لایه فیزیکی تنها عهده دار ارسال داده ها به صورت سیگنال های الکتریکیست ولی صحت ارسال داده ها را تضمین نمی کند. کانال های ارتباطی قطعا بدون خطا نبوده و به دلایل زیادی از جمله نویز[۱]، تصادم[۲] و ... باعث عدم اطمینان در صحت داده ها می شوند. وظیفه تامین امنیت داده ها در برابر مشکلات مذکور را لایه پیوند داده ها[۳] بر عهده دارد. این لایه داده ها را در قالب هایی به نام فریم قرار می‌دهد و در صورتی که خطایی به وقوع پیوست ارسال و یا دریافت اطلاعات را مجددا از سر می گیرد. این لایه همچنین جریان ارسال فریم ها را کنترل می کند تا یک دستگاه کند بتواند از یک دستگاه سریعتر فریم ها را بدون از دست رفتن دریافت کند.

۳-۲-۱.لایه شبکه

مسئولیت عمده لایه شبکه[۴] کنترل ازدحام و یافتن بهینه ترین مسیر برای ارسال اطلاعات است. این لایه که داده ها را در بسته[۵] هایی قرار می دهد، با تشخیص و انتخاب بهترین مسیر موجود بین فرستنده و گیرنده از ایجاد ازدحام و ترافیک در مسیریاب ها جلوگیری می کند. لایه شبکه که بدون اتصال[۶] است بدون تضمین دریافت بسته ها با همان ترتیب ارسالی از فرستنده، بسته ها را دریافت می کند. همچنین این لایه هیچ اطلاعی از اینکه گیرنده آماده ی دریافت بسته ها می باشد یا نه، ندارد.

۴-۲-۱.لایه انتقال

لایه انتقال[۷] مشکلات و کاستی های موجود در لایه شبکه را برطرف می کند. در این لایه داده ها در بخش هایی به نام قطعه[۸] دسته بندی می شوند و قبل از ارسال، بسته های موجود در هر قطعه شماره گذاری می شوند تا ترتیب بسته های ارسالی محفوظ بماند و از دریافت تکراری و گم شدن بسته ها پرهیز شود. لایه انتقال که لایه ای اتصال گراست[۹] قبل از شروع ارسال، از آماده بودن گیرنده برای دریافت اطلاعات اطمینان حاصل می کند.

---

[۱] Noise
[۲] Collision
[۳] Data Link Layer
[۴] Network Layer
[۵] Packet
[۶] Connectionless
[۷] Transport Layer
[۸] Segment
[۹] Connection Oriented

۵-۲-۱.لایه جلسه

لایه جلسه[۱] وظیفه ایجاد، مدیریت و خاتمه دادن به یک جلسه را بر عهده دارد. هر جلسه برای دو برنامه که بر روی دو دستگاه برقرار می شود می تواند این فرآیند را مدیریت کند و در صورت امکان قطع شدن ارتباط آن را از سر بگیرد.

۶-۲-۱.لایه ارائه

لایه ارائه[۲] بوسیله متدهای استانداردی به رمزگذاری و رمزگشایی پیام ها می پردازد و اطمینان خاطر ایجاد می کند که ماشین هایی که از استانداردهای متفاوتی برای متن استفاده می کنند، می توانند بدون هیچ مشکلی با یکدیگر ارتباط برقرار نمایند. همچنین فعالیت هایی مانند فشرده سازی فایل ها در این لایه صورت می گیرد.

۷-۲-۱.لایه کاربرد

لایه کاربرد[۳] در واقع واسطی میان کاربر و شبکه می باشد. برنامه های کاربردی از جمله مرورگر وب، برنامه ارسال پیام الکترونیکی و... در این لایه قرار دارند که از پروتکل های استانداردی مانند HTTP[۴]، FTP[۵]، SMTP[۶] و... استفاده می کنند.

همانطور که پیشتر عنوان شد این مدل هفت لایه در بین توسعه دهندگان شبکه مورد توجه چندانی قرار نگرفته است و بیشتر طراحان به سراغ مدل چهار لایه ای TCP/IP[۷] رفته اند که بسیار پرکاربرد است. در ادامه به شرح این مدل می پردازیم.

۳-۱.مدل چهار لایه شبکه TCP/IP

لایه های این مدل را در شکل ۵-۱ مشاهده می کنید. لازم به ذکر است دو لایه ی جلسه و ارائه موجود در مدل هفت لایه OSI در مدل چهار لایه TCP/IP وجود ندارند و وظایف مربوط به این لایه ها در صورت لزوم توسط لایه کاربرد انجام می شود. در ادامه به شرح مختصری در مورد هر لایه می پردازیم.

---

[۱] Session Layer

[۲] Presentation Layer

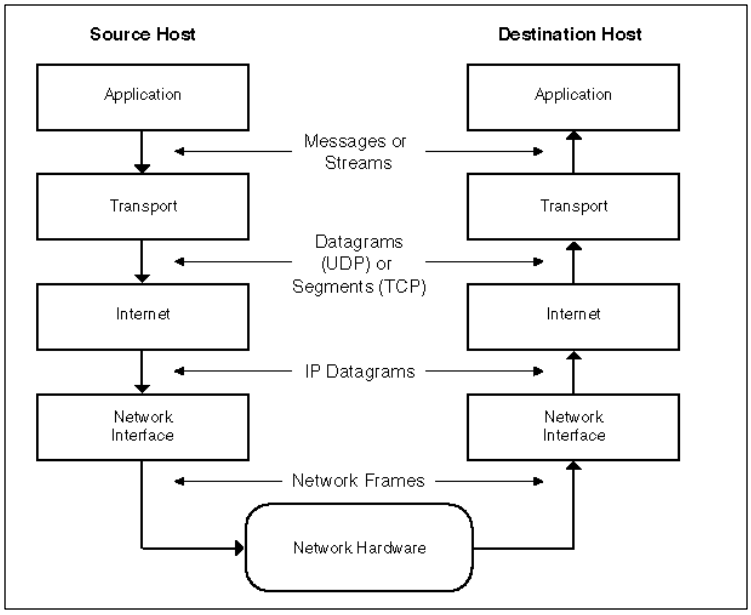
[۳] Application Layer

[۴] Hypertext Transfer Protocol

[۵] File Transfer Protocol

[۶] Simple Mail Transfer Protocol

[۷] Transmission Control Protocol/Internet Protocol



شکل ۵-۱ مدل چهارلایه TCP/IP

۱-۳-۱.لایه واسط شبکه

این لایه مشابه آنچه در رابطه با لایه فیزیکی مدل هفت لایه OSI عنوان شد با داده های بیتی و تبدیل آنها به سیگنال های الکتریکی و ارسال آنها بر روی کانال های ارتباطی سروکار دارد. لایه واسط شبکه[۱] که کاملا با سخت افزارهای شبکه و راه اندازهای مورد نیاز آنها سروکار دارد امکان ایجاد ارتباط بین سخت افزارهای غیر همگن را بوسیله مجموعه ای از استانداردها، بوجود می آورد. شناسه هر دستگاه در این لایه آدرس MAC[۲] می باشد که برای تمام دستگاه های ارتباطی شناسه ای منحصر به فرد است و هنگام ساخت آن دستگاه توسط شرکت سازنده به آن اختصاص می یابد.

۲-۳-۱.لایه اینترنت(شبکه)

وظیفه اصلی لایه اینترنت[۳] هدایت داده ها در قالب بسته هایی موسوم به بسته های IP[۴] در مسیرهای موجود بین مسیریاب ها می باشد. این لایه پس از بسته بندی اطلاعات و اختصاص شناسه منحصر به فردی به نام IP اقدام به گزینش مسیر مناسب برای ارسال بسته ها می کند. این ها بخشی از وظایف پروتکل IP می باشد که در این لایه کار می کند. در جدول ۱-۱ می توانید سایر پروتکل های مهم این لایه را مشاهده نمایید.

---

[۱] Network Interface Layer

[۲] Media Access Control

[۳] Internet Layer

[۴] Internet Protocol

پروتکل پیام های کنترلی، مانند پیغام های خطا	ICMP
پروتکلی برای تبدیل آدرس IP به آدرس MAC	ARP
معکوس پروتکل ARP	RARP
پروتکلی برای مدیریت ارسال پیام به صورت گروهی	IGMP

جدول ۱-۱

۳-۳-۱.لایه انتقال

همانند لایه شبکه در مدل هفت لایه OSI، در لایه اینترنت مدل چهار لایه TCP/IP مشکلاتی از قبیل عدم اطمینان از ارسال بدون نقص بسته ها و ترتیب دریافتی صحیح بسته ها و یا آماده بودن گیرنده برای دریافت داده ها وجود دارد. این مشکلات توسط لایه انتقال و در راس آنها بوسیله پروتکل TCP مرتفع می گردد. لایه انتقال که لایه ای اتصال گراست به وسیله تمهیداتی از بروز مشکلات فوق الذکر و مشکلات دیگری مانند رسیدن بسته ای قدیمی که زمان حیات آن منقضی شده و بسته دوباره ارسال شده ی جدید به طور همزمان جلوگیری می نماید. همچنین لایه انتقال امکان استفاده چندین برنامه به صورت همزمان و مجزا از شبکه را مهیا می کند. این امکان با اختصاص یک شماره منحصر به فرد به نام شماره پورت[۱] به برنامه ها محقق می شود. پروتکل های استاندارد شبکه هر کدام شماره پورت رزرو شده ای دارند که در جدول ۲-۱ می توانید تعدادی از آنها را مشاهده کنید. پورت های رزرو شده در دامنه ۰ تا ۱۰۲۳ می باشند و پورت های بالای این دامنه را می توان به برنامه های کاربردی دیگر اختصاص داد.

۴-۳-۱.لایه کاربرد

این لایه دقیقا مشابه آنچه که برای لایه کاربرد مدل هفت لایه OSI شرح دادیم می باشد با این تفاوت که قادر است وظایف مربوط به رمزگذاری، رمزگشایی و فشرده سازی داده ها را نیز در صورت نیاز انجام دهد.

در این بخش به توضیح اجمالی پیرامون برخی از مباحث پایه ای شبکه پرداختیم. در بخش بعد قصد داریم با بهره گیری از برخی مفاهیم شبکه و برنامه نویسی تحت شبکه یک سرویس چت پایه ای را طراحی و پیاده سازی نماییم تا به طور کاربردی با این مفاهیم بیشتر آشنا شویم.

[۱] Port Number

20 & 21	FTP
22	SSH
23	Telnet
25	SMTP
53	DNS
67	BOOTP (Server)
68	BOOTP (Client)
69	TFTP
70	Gopher Protocol
80	HTTP
110	POP3
119	NNTP
143	IMAP
161	SNMP
194	IRC
443	HTTPS
444	SNPP

جدول ۲-۱

۲.سوکت چیست؟

سوکت[۱] یک مفهوم انتزاعی از تعریف ارتباط در سطح برنامه نویسی خواهد بود و برنامه نویس با تعریف سوکت عملا تمایل خود را برای مبادله داده ها به سیستم عامل اعلام کرده و بدون درگیر شدن با جزئیات پروتکل TCP یا UDP از سیستم عامل می خواهد تا فضا و منابع مورد نیاز را جهت برقراری یک ارتباط، ایجاد کند.[1] سوکت ها به دو دسته کلی TCP و UDP تقسیم می شوند. نوع TCP نوع قابل اعتماد این سوکت هاست به این معنی که برای هر اتصال دست تکانی سه مرحله ای[۲] انجام داده و از صحت ارسال اطلاعات اطمینان حاصل می نماید. اما نوع UDP یا همان سوکت های دیتاگرام، اطلاعات را بدون هیچ اطلاع قبلی ارسال می کنند. از این رو این سوکت ها غیر قابل اعتماد بوده و صحت ارسال اطلاعات و ترتیب آنها را تضمین نمی کنند اما در هر حال از سرعت بالاتری به نسبت سوکت های TCP برخوردارند و برای ارسال صدا و تصویر مناسبند. در این مقاله سعی داریم با استفاده از سوکت و مفاهیمی مانند چند نخی[۳] و انحصار متقابل[۴] مراحل ساخت یک سرویس چت پایه ای را شرح دهیم. این برنامه شامل یک سرویس دهنده می باشد که کلاینت ها بعد از اتصال با سرور می توانند به صورت عمومی با یکدیگر ارتباط برقرار نمایند به این صورت که پیغام ارسالی از یک کلاینت برای تمامی کلاینت ها ارسال می شود. در ادامه مفصلا به شرح برنامه سرور و برنامه کلاینت و همچنین توابع و کلاس های استفاده شده می پردازیم.لازم به ذکر است، زبان برنامه نویسی انتخابی ما در این مقاله جاوا می باشد.

۱-۲.سوکت ها در جاوا

در جاوا سوکت های UDP و TCP وجود داشته و می توان آنها را در پکیج `java.net` یافت. ما در این مقاله از سوکت های نوع TCP استفاده می کنیم. سوکت های TCP شامل دو شیئ با نام های `ServerSocket` و `Socket` می باشند. شیئ `ServerSocket` در واقع وظیفه انتظار و پذیرش اتصال های درخواستی به یک ماشین را دارد و پس از پذیرش هر یک از اتصالات، آن را به شیئ `Socket` تحویل می دهد.

---

[۱] Socket

[۲] Tree Way Handshake

[۳] Multithreading

[۴] Mutual Exclusion

```
public class ChatServer
{
    private ServerSocket ss;
    private void listen(int port) throws IOException
    {
        ss = new ServerSocket(port);
        System.out.println("We are listen to : "+ss);
        while(true)
        {
        Socket s = ss.accept();
        System.out.println("Connection from : "+s);
        }
    }
}
```

قطعه کد ۱-۲

همانطور که در قطعه کد بالا مشاهده می کنید کلاس ChatServer شامل یک تابع به نام listen می باشد که این تابع با دریافت یک شماره پورت و ارسال آن به شیئ ServerSocket اعلام می دارد که این برنامه به شماره پورت مذکور گوش می دهد و اتصالات درخواستی به این شماره پورت را سرویس می دهد. سپس درون یک حلقه بینهایت اقدام به پذیرش یکایک اتصالات درخواستی نموده و هر یک از این اتصالات را به یک شیئ Socket اختصاص می دهد.

۳.جریان داده ها[۱] در جاوا

در جاوا و در پکیج java.io یک کلاس با نام DataOutputStream وجود دارد که با استفاده از آن می توان اطلاعاتی را به خروجی مورد نظر ارسال کرد. در این مثال خروجی، شیئ Socket می باشد و تنها کاری که باید انجام دهیم، الحاق این خروجی به شیئ DataOutputStream است. در پکیج java.util کلاسی با نام Hashtable وجود داشته که نوعی ساختمان داده را معرفی می کند که شامل دو بخش کلید و مقدار است و هر رکورد آن شامل یک زوج کلید و مقدار است. ما از این ساختمان داده برای ذخیره هر سوکت و دیتا استریم ساخته شده برای آن استفاده می کنیم که برای مراحل بعدی مورد استفاده قرار می گیرد. در قطعه کد۱-۳ مراحل این کار قابل مشاهده است.

```
public class ChatServer
{
    private ServerSocket ss;
    private Hashtable outputStreams = new Hashtable();
    private void listen(int port) throws IOException
    {
        ss = new ServerSocket(port);
        System.out.println("We are listen to : "+ss);
        while(true)
        {
```

[۱] Data Streams

```
        Socket s = ss.accept();
        System.out.println("Connection from : "+s);
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        outputStreams.put(s, dout);
        }
    }
}
```

قطعه کد ۱-۳

۴.چند نخی

همانطور که پروسه های مختلف درون یک سیستم عامل به صورت همروند و موازی اجرا می شوند، گاهی انتظار داریم تا اجزای یک پروسه نیز به صورت موازی اجرا شوند. در اکثر زبان های برنامه نویسی (از جمله جاوا) مکانیزمی به نام چند نخی وجود داشته که به کمک آن می توان اجزای یک پروسه را به صورت همروند و موازی اجرا کرد. در این برنامه ما وظیفه سرویس دادن به یکایک اتصالات را به یک نخ[۱] منحصر به فرد محول می کنیم. هر نخ وظیفه انتظار و دریافت اطلاعات از یک اتصال را بر عهده دارد. برای این منظور یک کلاس با نام SrvrIncomingMsgThread ایجاد می کنیم. نکته قابل توجه در این کلاس این است که برای ساخت اشیاء همروند، باید این کلاس خواص کلاسی با نام Thread را به ارث ببرد. کلاس Thread در پکیج java.lang وجود داشته و هر کلاسی که از آن ارث بری کند باید تابع run() موجود در این کلاس را دوباره نویسی[۲] نماید. در این تابع، کدهایی را که می خواهیم به عنوان یک نخ مجزا اجرا شوند را قرار می دهیم. برای اجرای این تابع باید تابع start() را درون تابع سازنده‌ی[۳] کلاس فرخوانی کرد. قطعه کدهای ۱-۴ و ۲-۴ مراحل انجام این کار را نشان می دهند.

```
public class ChatServer {
    private ServerSocket ss;
    private Hashtable outputStreams = new Hashtable();
    private void listen(int port) throws IOException
    {
        ss = new ServerSocket(port);
        System.out.println("We are listen to : "+ss);
        while(true)
        {
        Socket s = ss.accept();
        System.out.println("Connection from : "+s);
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        outputStreams.put(s, dout);
        new SrvrIncomingMsgThread(this,s);
        }
    }
}
```

قطعه کد ۱-۴

[۱] Thread

[۲] Overwrite

[۳] Constructor

```
public class SrvrIncomingMsgThread extends Thread
{
    ChatServer chatserver;
    Socket socket;
    public SrvrIncomingMsgThread(ChatServer chatserver,Socket socket)
    {
        this.chatserver = chatserver;
        this.socket = socket;
        start();
    }
    public void run()
    {
      //Your Thread Code
    }
}
```

قطعه کد ۲-۴

۵.انحصار متقابل

استفاده کردن از چند نخی گاهی باعث بروز مشکلاتی می شود که یکی از مهمترین آن ها استفاده دو یا چند نخ از یک داده اشتراکی به صورت همزمان است. اگر این داده اشتراکی حاوی اطلاعات مهمی باشد و یکی یا بیشتر نخ ها قصد تغییر محتوای آن را داشته باشند، ممکن است عملکرد برخی از نخ ها و یا همگی آنها دچار اشکال شود. در این برنامه نیز چنین احتمالی وجود داشته و همین امر ما را بر آن می دارد تا از مکانیزمی با نام انحصار متقابل استفاده نماییم. انحصار متقابل در واقع ضامن دسترسی انحصاری هر نخ به یک داده اشتراکی مشخص شده می باشد. تا مادامی که داده ای در انحصار یک نخ باشد نخ دیگری حق دسترسی به آن داده را نخواهد داشت. در بخش های گذشته ساختمان داده ای تحت عنوان Hashtable را معرفی کردیم که هر رکورد آن حاوی شیئ سوکت و شیئ دیتا استریم معادل آن می باشد. هنگامی که پیامی از سوی یکی از کلاینت ها به سرور ارسال می شود، سرور باید آن را به باقی کلاینت ها ارسال کند. سرور ارسال پیام را در داخل یک حلقه برای تمام دیتا استریم های موجود در شیئ outputStreams انجام می دهد. از طرفی اگر کلاینتی اتصال خود را با سرور قطع نماید، سرور، سوکت مربوط به آن کلاینت را از داخل شیئ outputStreams حذف می کند. حال در نظر بگیرید این عمل حذف کردن از داخل شیئ outputStreams همزمان با ارسال پیام به تمامی اعضای این شیئ صورت گیرد. این عمل می تواند حامل پیش آمدهای غیر قابل پیش بینی باشد و اجرای برنامه را دچار مشکل نماید. راه حل این مشکل ایجاد انحصار متقابل بر روی این شیئ می باشد تا از دسترسی همزمان نخ ها به این شیئ جلوگیری شود. در جاوا روش های گوناگونی برای ایجاد انحصار متقابل وجود داشته که یکی از آنها استفاده از تابع synchronized() است. با استفاده از این تابع می توان بلوکی از کد را که قصد دسترسی به داده خاصی را دارد قرار دهیم تا فقط در صورتی اجازه ی دسترسی به آن شیئ را داشته باشد که آن شیئ توسط نخ دیگری در حال استفاده نباشد. شیئ مورد نظر را باید به صورت ورودی به تابع synchronized() ارسال کنیم. در قطعه کد ۱-۵ چگونگی استفاده از این تابع را مشاهده می کنید. دو تابع sendToAll() و closeConnction() در ادامه معرفی خواهند شد.

```
public class ChatServer
{
    private ServerSocket ss;
    private Hashtable outputStreams = new Hashtable();
    private void listen(int port) throws IOException
    {
        ss = new ServerSocket(port);
        System.out.println("We are listen to : "+ss);
        while(true)
        {
        Socket s = ss.accept();
        System.out.println("Connection from : "+s);
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        outputStreams.put(s, dout);
        new SrvrIncomingMsgThread(this,s);
        }
    }
    public void sendToAll(String msg)
    {
        synchronized(outputStreams)
        {
            //For All Sockets in outputStreams Send msg
        }
    }
   public void closeConnction(Socket s)
    {
        synchronized(outputStreams)
        {
            //Remove s From outputStreams
        }
    }
}
```

قطعه کد ۱-۵

۶.تحلیل و بررسی برنامه

در بخش های گذشته با مفاهیمی آشنا شدیم که پایه و اساس یک سرویس دهنده چت را تشکیل می دهند. حال در این قسمت به تحلیل و تشریح کد برنامه‌ی سرویس دهنده و سرویس گیرنده که در پیوست این مقاله موجود می باشند، می پردازیم.

۱-۶.تحلیل و بررسی کد سرویس دهنده

تابع `main()` در کلاس `ChatServer` شامل یک شیئ `Scanner` می باشد که بوسیله تابع `nextInt()`، شماره پورت برنامه را از کاربر دریافت می کند. این شماره پورت باید شماره ای منحصر به فرد باشد و برنامه ی دیگری در حال استفاده از آن نباشد. در ادامه بوسیله کلمه کلیدی `new` یک شیئ جدید از کلاس

ChatServer می سازیم و شماره پورت را به تابع سازنده کلاس ارسال می کنیم. این کار درون یک بلوک try…catch() انجام می گیرد تا در صورت بروز خطا پیغام مناسبی به کاربر نمایش داده شود. تابع سازنده کلاس شماره پورت مذکور را گرفته و آن را برای تابع listen() ارسال می کند. در بخش های گذشته جزئیات این تابع را بررسی کردیم و دیدیم که بخش های مختلف آن چه عملکردی دارند. همانطور که در کد این تابع مشخص است به ازای هر درخواست پذیرفته شده، یک شیئ از کلاس SrvrIncomingMsgThread ایجاد شده و آدرس شیئ کلاس ChatServer (بوسیله کلمه کلیدی this) و آدرس سوکت مورد نظر برای تابع سازنده آن ارسال می شود. همانطور که پیشتر گفته شد تابع sendToAll() وظیفه ارسال همگانی پیام های دریافتی را بر عهده دارد. این تابع درون یک حلقه و با استفاده از ساختمان داده ی خاصی به نام نوع شمارشی[1] اقدام به ارسال پیام به تمامی کلاینت ها می کند. این ساختمان داده می تواند در هر بار یکی از عناصر موجود در یک شیئ را تولید کند. به عنوان مثال در این برنامه در هر بار آدرس یکی از سوکت های موجود در شیئ outputStreams را بوسیله تابع getOutputStreams() که از نوع ساختمان داده شمارشی می باشد، برمی گردانیم. سپس شیئ دیتا استریم معادل آن سوکت را برای ارسال پیام در شیئ dout ذخیره کرده و بوسیله تابع writeUTF() پیام را ارسال می کنیم. تابع دیگری که در این کلاس وجود دارد تابع closeConnction() می باشد که وظیفه ی حذف آدرس سوکت و دیتا استریم معادل آن از داخل شیئ outputStreams را بر عهده دارد و به عبارتی دیگر به ارتباط خود با کلاینت مورد نظر پایان می دهد. حال به بررسی کلاس SrvrIncomingMsgThread می پردازیم. همانطور که پیشتر گفته شد به ازای هر سوکت یک شیئ از کلاس SrvrIncomingMsgThread ایجاد می شود که مسئولیت دریافت پیام از آن سوکت مورد نظر را بر عهده دارد. تابع سازنده این کلاس شیئ ChatServer و شیئ سوکت را به عنوان ورودی دریافت کرده و پس از انتساب آنها به دو شیئ محلی دیگر تابع start() را برای ایجاد یک نخ و اجرای دستورات داخل تابع run() فراخوانی می کند. در تابع run() ابتدا یک شیئ از نوع DataInputStream ایجاد کرده و شیئ سوکت را به عنوان جریان ورودی داده به آن الحاق می کنیم. سپس درون یک حلقه بینهایت برای ورود پیام انتظار می کشیم. در صورت ورود پیام، آن را داخل یک رشته ذخیره کرده و برای تابع sendToAll() شیئ chatserver، ارسال می نماییم. تمامی این مراحل درون یک بلوک try…catch()…finally قرار داده شده است که در صورت بروز خطا، ارتباط با کلاینت مورد نظر خاتمه یابد. این کار با فراخوانی تابع closeConnction() شیئ chatserver انجام می شود.

۲-۶.تحلیل و بررسی کد سرویس گیرنده

برنامه سرویس گیرنده یا به عبارت دیگر کلاینت، شامل سه کلاس با نام های ChatClient، ReceiveMsgThread و SendMsgThread می باشد. در تابع main() از کلاس ChatClient ابتدا آدرس IP ماشین سرویس دهنده و سپس شماره پورت آن را از کاربر دریافت می کنیم. سپس یک شیئ سوکت

[1] Enumeration

بوسیله ی این آدرس IP و شماره پورت، ایجاد کرده و آن را به تابع سازنده دو شیئ از کلاس ReceiveMsgThread و SendMsgThread ارسال می کنیم. این دو کلاس که هر دو از کلاس Thread ارث بری می کنند به صورت دو نخ همزمان بر روی سیستم کلاینت اجرا می شوند که یکی از آنها وظیفه سرویس دهی به پیام های دریافتی و نمایش آنها بر روی خروجی را بر عهده دارد و دیگری به ارسال پیام از سیستم کلاینت می پردازد. واضح است که برای همروندی در ارسال و دریافت پیام ها ما مجبور به استفاده از چندنخی می باشیم. شیئ ایجاد شده از کلاس ReceiveMsgThread که عهده دار سرویس دهی به پیام های ورودی می باشد، درون یک حلقه بینهایت بوسیله تابع readUTF() به انتظار برای پیام ورودی از سمت سرور می نشیند و پس از دریافت پیام آن را بر روی خروجی چاپ می کند. شیئ ساخته شده از کلاس SendMsgThread موظف به انتظار و دریافت پیام های کاربر از صفحه کلید و ارسال آن برای سرویس دهنده است.

۷.خروجی برنامه

در این قسمت خروجی برنامه را بررسی می کنیم. در خروجی ۷-۱ ChatServer ابتدا برنامه از کاربر شماره پورت برنامه را می گیرد و به انتظار برای درخواست از سوی کلاینت می نشیند.

```
C:\>java ChatServer.class
Enter port number : 2222
We are listen to : ServerSocket[addr=0.0.0.0/0.0.0.0,localport=2222]
```

خروجی ۷-۱ ChatServer

حال برنامه کلاینت را بر روی دو سیستم دیگر اجرا می کنیم. حال کلاینت ها می توانند به تبادل پیام بپردازند. خروجی زیر برای همه کلاینت ها یکسان است.

```
C:\>java ChatClient.class
Enter server IP address : 192.168.1.2
Enter server port number : 2222
```

خروجی ۷-۲ ChatClient

خروجی برنامه ChatServer پس از اتصال کلاینت ها به صورت خروجی ۷-۳ ChatServer می باشد.

```
C:\>java ChatServer.class
Enter port number : 2222
We are listen to : ServerSocket[addr=0.0.0.0/0.0.0.0,localport=2222]
Connection from : Socket[addr=/192.168.1.3,port=49514,localport=2222]
Connection from : Socket[addr=/192.168.1.4,port=49568,localport=2222]
```

خروجی ۳-۷ ChatServer

در این مثال سرویس دهنده بر روی کامپیوتری با آدرس آی پی 192.168.1.2 می باشد که دو کلاینت با آدرس آی پی های 192.168.1.3 و 192.168.1.4 به آن متصل شده اند. حال هر پیامی که از سوی یکی از این کلاینت ها فرستاده شود برای کلاینت دیگر نیز ارسال می شود. در صورتی که کلاینتی اتصال خود را قطع کند و یا به هر دلیلی ارتباط آن با سرویس دهنده خاتمه یابد، به طور مثال اگر هر دو کلاینت اتصال خود را خاتمه دهند، خروجی برنامه سرویس دهنده به صورت خروجی ۴-۷ ChatServer خواهد بود.

```
C:\>java ChatServer.class
Enter port number : 2222
We are listen to : ServerSocket[addr=0.0.0.0/0.0.0.0,localport=2222]
Connection from : Socket[addr=/192.168.1.3,port=49514,localport=2222]
Connection from : Socket[addr=/192.168.1.4,port=49568,localport=2222]
Close connection to : Socket[addr=/192.168.1.3,port=49514,localport=2222]
Close connection to : Socket[addr=/192.168.1.4,port=49568,localport=2222]
```

خروجی ۴-۷ ChatServer

منابع

[1] . اصول مهندسی اینترنت، دکتر احسان ملکیان

[2] . http://www.ibm.com/developerworks/edu/j-dw-javachat-i.html

[3] . Computer Networks, Andrew S. Tanenbaum, 5th Edition

[4] . Modern Operating Systems, Andrew S. Tanenbaum, 3rd Edition

ChatServer.java

```
/*
 * ChatServer.java
 * author Mehrdad Ghadiri
 */

package chatserver;
import java.io.*;
import java.net.*;
import java.util.*;
public class ChatServer
{
    private ServerSocket ss;
    private Hashtable outputStreams = new Hashtable();
    public ChatServer(int port) throws IOException
    {
        listen(port);
    }
    private void listen(int port) throws IOException
    {
        ss = new ServerSocket(port);
        System.out.println("We are listen to : "+ss);
        while(true)
        {
        Socket s = ss.accept();
        System.out.println("Connection from : "+s);
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        outputStreams.put(s, dout);
        new SrvrIncomingMsgThread(this,s);
        }
    }
    Enumeration getOutputStreams()
    {
       return outputStreams.elements();
    }
    public void sendToAll(String msg)
    {
        synchronized(outputStreams)
        {
            for(Enumeration e = getOutputStreams();e.hasMoreElements();)
            {
                DataOutputStream dout = (DataOutputStream) e.nextElement();
                try
                {
                    dout.writeUTF(msg);
                }
                catch(Exception ee)
                {
                    //do nothing
                }
            }
        }
    }
    void closeConnction(Socket s)
    {
        synchronized(outputStreams)
        {
            System.out.print("Close connection to :"+s+"\n");
            outputStreams.remove(s);
```

```
            try
            {
                s.close();
            }
            catch(Exception ee)
            {
                //do nothing
            }

        }
    }
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
      System.out.print("Enter port number : ");
        int port = in.nextInt();
        try
        {
        new ChatServer(port);
        }
        catch(IOException e)
        {
            System.out.println("There is a fatal server Error!");
        }
    }

}
```

SrvrIncomingMsgThread.java

```
/*
 * SrvrIncomingMsgThread.java
 * author Mehrdad Ghadiri
 */

package chatserver;
import java.net.*;
import java.io.*;
import java.util.*;
public class SrvrIncomingMsgThread extends Thread
{
    ChatServer chatserver;
    Socket socket;
    public SrvrIncomingMsgThread(ChatServer chatserver,Socket socket)
    {
        this.chatserver = chatserver;
        this.socket = socket;
        start();
    }
    public void run()
    {
        try
        {
            DataInputStream din = new
DataInputStream(socket.getInputStream());
            while(true)
            {
                String msg = din.readUTF();
                chatserver.sendToAll(msg);
            }
```

```
        }
        catch (Exception ex)
        {
            //IOException
        }
        finally
        {
            chatserver.closeConnction(socket);
        }
    }
}
```

ChatClient.java

```
/*
 * ChatClient.java
 * author Mehrdad Ghadiri
 */

package chatclient;
import java.io.*;
import java.net.*;
import java.util.*;
import chatclient.*;
public class ChatClient
{
    public static void main(String[] args) throws IOException
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter server IP address : ");
        String address = in.next();
        System.out.print("Enter server port number : ");
        int port = in.nextInt();
        Socket socket = new Socket(InetAddress.getByName(address),port);
        new SendMsgThread(socket);
        new ReceiveMsgThread(socket);
    }
}
```

ReceiveMsgThread.java

```
/*
 * ReceiveMsgThread.java
 * author Mehrdad Ghadiri
 */

package chatclient;
import java.io.*;
import java.net.*;
import java.util.*;
public class ReceiveMsgThread extends Thread
{
    Socket socket;

    public ReceiveMsgThread(Socket socket)
    {
        this.socket = socket;
        start();
```

```
    }
    public void run()
    {
        try
        {
        DataInputStream din = new DataInputStream(socket.getInputStream());

        while(true)
        {
            String msg = din.readUTF();
            System.out.print(msg+"\n");
        }
        }
        catch(Exception e)
        {
            //do nothing
        }
    }
}
```

SendMsgThread.java

```
/*
 * SendMsgThread.java
 * author Mehrdad Ghadiri
 */

package chatclient;
import java.io.*;
import java.net.*;
import java.util.*;
public class SendMsgThread extends Thread
{
    Socket socket;
    public SendMsgThread(Socket socket)
    {
        this.socket = socket;
        start();
    }
    public void run()
    {
        try
        {
        DataOutputStream dout = new
DataOutputStream(socket.getOutputStream());
        while(true)
        {
            Scanner in = new Scanner(System.in);
            String msg = in.nextLine();
            dout.writeUTF(InetAddress.getLocalHost().getHostAddress()+" :
"+msg);
        }
        }
        catch(Exception e)
        {
            //do nothing
        }

    }
}
```