
بسم الله الرحمن الرحيم

در فصول گذشته ساختار پروتکلهای TCP و IP را بررسی کردیم و طریقه آدرس‌دهی ماشینها و پروسه های روی هر ماشین را بوسیله آدرس IP و آدرس پورت آموختیم. معمولاً پیاده‌سازی این پروتکلها توسط طراح هر سیستم عامل انجام و بعنوان جزئی از سیستم عامل همراه آن ارائه و نصب می‌شود. در سیستم عامل هایی مثل یونیکس یا MS-Windows که اصل[1] برنامه آن در دسترس نیست این انعطاف وجود ندارد که بتوان در محیطهای آزمایشگاهی برنامه های TCP و IP یا پروتکلهای مرتبط با آنها را تغییراتی داد و نتیجه تغییرات را بررسی و تحلیل کرد لذا نظر علاقمندان به این مورد را به سیستم عامل " لینوکس" جلب می‌نمائیم.

حال فرض می‌کنیم یک برنامه نویس بخواهد در یک محیط برنامه نویسی مثل C بگونه ای برنامه نویسی کند که محتویات یک فایل درون یک کامپیوتر راه دور را تغییر بدهد یا آنرا روی کامپیوتر خودش منتقل نماید یا فرض کنید یک برنامه نویس موظف شده است که یک محیط پست الکترونیکی خاص و با امکانات ویژه برای بکارگیری در یک محیط اداری طراحی نماید. برای طراحی چنین برنامه هایی که تماماً در لایه چهارم یعنی لایه کاربرد تعریف می‌شود برنامه نویس باید به نحوی با مفاهیم برنامه نویسی تحت شبکه آشنا باشد.

در این فصل اصول کلی برنامه نویسی شبکه و مفهوم "سوکت"[۲] را مورد بررسی قرار می‌دهیم و با مثالهای ساده روش نوشتن برنامه های کاربردی تحت پروتکل TCP/IP را تشریح خواهیم کرد. برای سادگی کار و همچنین ارائه دید عمیق ، کدهائی که در این فصل ارائه می‌شوند به زبان C هستند که در محیط سیستم عامل لینوکس و با مترجم **gcc** به زبان ماشین ترجمه شده‌اند.

در حقیقت این فصل نقطه آغازی است برای تمام برنامه نویسانی که به نحوی مجبور خواهند شد برنامه کاربردی تحت شبکه اینترنت بنویسند.

سنگ بنای تمام برنامه های کاربردی لایه چهارم مفهومی بنام سوکت است که این مفهوم توسط طراحان سیستم عامل یونیکس به زیبایی به منظور برقراری ارتباط برنامه های تحت شبکه و تبادل جریان داده بین پروسه ها ابداع شد و در این فصل باید مفهوم آنرا کالبد شکافی کنیم.

---

[۱] Source
[۲] Socket

شاید شما این جمله را شنیده باشید **"در دنیای یونیکس هر چیزی می‌تواند بصورت یک فایل تلقی و مدل شود"**. تمام عوامل و انواع ورودی و خروجیها (I/O) می‌تواند توسط سیستم فایل مدل شود. مثلاً چاپگر می‌تواند یک فایل باشد (مثلاً فایلی با نام PRN). حال وقتی سیستم عامل چاپگر را بصورت یک فایل استاندارد مدل کرده باشد شما با مفاهیمی که از فایلها و چگونگی بکارگیری آنها در محیط برنامه نویسی آموخته اید، برای راه اندازی چاپگر و چاپ یک متن، می‌توانید در برنامه خود عملیات ساده و در عین حال استاندارد زیر را انجام بدهید:

الف: چاپگر را همانند یک فایل با نام استاندارد آن بصورت فایلی "فقط نوشتنی" باز می‌کنید. (با دستورات open() یا fopen()).

ب: اگر نتیجه مرحله قبل موفقیت آمیز بود سیستم عامل یک مشخصه فایل[1] بعنوان اشاره گر فایل برمی‌گرداند.

ج: داده‌هائی که قرار است بر روی چاپگر ارسال شوند را با همان دستورات معمولی نوشتن در فایل (دستور معمولی write() یا fwrite()) ، درون فایل باز شده از مرحله قبل می‌نویسید.

د: پس از اتمام کار فایل را می‌بندید. (دستور close() یا fclose())

کلیت کاری که باید انجام بشود همین چند مرحله است و برنامه نویس به هیچ عنوان درگیر ساختار چاپگر و اعمالی که برای راه اندازی و چاپ یک متن لازم است نخواهد شد. این وظائف را راه انداز چاپگر بعنوان بخشی از پوسته سیستم عامل بعهده دارد.

چهار مرحله‌ای که برای بکارگیری چاپگر معرفی شد دقیقا میتواند برای نوشتن بر روی صفحه نمایش یا خواندن از آن مورد استفاده قرار گیرد ، فقط باید نام فایل صفحه نمایش "کنسول" (con) در نظر گرفته شود.

یونیکس قادر است تمام دستگاههای ورودی و خروجی را بعنوان فایل مدل نماید. بنابراین تمام عملیاتی که برنامه نویس برای بکارگیری دستگاههای مختلف بایستی بداند و بکار بگیرد یکسان و ساده و شفاف خواهد بود. آیا می‌توانید گزاره های زیر را بپذیرید:

- چاپگر فایلی است فقط نوشتنی

---

[1] File Descriptor

- پویشگر[1] فایلی است فقط خواندنی
- صفحه نمایش بعنوان کنسول فایلی است خواندنی و نوشتنی
- پورت سریال فایلی است خواندنی و نوشتنی
- یک فایل واقعی روی دیسک سخت فایلی است خواندنی و نوشتنی
- یک فایل واقعی روی دیسک فشرده فایلی است فقط خواندنی
- صف FIFO یا خط لوله[2] در محیط یونیکس فایلهایی هستند خواندنی و نوشتنی

حال که ذهن شما این نکته را پذیرفت که هر نوع I/O در دنیای سیستم عامل بصورت یک فایل استاندارد قابل عرضه و مدل کردن است ، شما را با یک سوال کلیدی مواجه می‌کنیم :

"آیا ارتباط دو کامپیوتر روی شبکه و مبادله اطلاعات بین آن دو، ماهیت ورودی / خروجی (I/O) ندارد؟"

اگر جوابتان منفی است این فصل را رها کنید ولی اگر تردید دارید یا یقیناً جوابتان مثبت است تا انتها این فصل را دنبال نمایید.

اگر ساختار فایل را برای ارتباطات شبکه ای تعمیم بدهیم آنگاه برای برقراری ارتباط بین دو برنامه روی کامپیوترهای راه دور روال زیر پذیرفتنی است :

الف: در برنامه خود از سیستم عامل بخواهید تا شرایط را برای برقراری یک "ارتباط" با کامپیوتری خاص (با آدرس IP مشخص) و برنامه ای خاص روی آن کامپیوتر (با آدرس پورت مشخص) فراهم کند یا اصطلاحاً سوکتی را بگشاید. اگر این کار موفقیت آمیز بود سیستم عامل برای شما یک اشاره گر فایل برمی‌گرداند و در غیر اینصورت طبق معمول مقدار پوچ (NULL) به برنامه شما برخواهد گرداند.

ب: در صورت موفقیت آمیز بودن عمل در مرحله قبل ، به همان صورتی که درون یک فایل می‌نویسید یا از آن می‌خوانید ، می‌توانید با توابع send() [یا write()] و recv() [یا read()] اقدام به مبادله داده‌ها بنمائید.

ج: عملیات مبادله داده‌ها که تمام شد ارتباط را همانند یک فایل معمولی ببندید. (با تابع close())

برای آنکه در برنامه خود همانند فایل یک "اشاره گر ارتباط" را از سیستم عامل طلب کنید تا برایتان مقدمات یک ارتباط را فراهم کند بایستی تابع سیستمی socket()

---

[۱] Scanner
[۲] Pipe

را در برنامه خود صدا بزنید. درصورتی که عمل موفقیت آمیزبود ، یک اشاره‌گر غیر پوچ بر می‌گردد که از آن برای فراخوانی توابع و روالهای بعدی استفاده خواهد شد.

پس از این هرگاه از "سوکت باز" یا مبادله داده‌ها روی سوکت یاد کردیم منظورمان اشاره به یک ارتباط باز یا مبادله اطلاعات بین دو نقطه TSAP[1] روی دو سیستم شبکه کامپیوتری می‌باشد.

دقیقاً همانند فایلها که میتوان همزمان چندین فایل را در یک برنامه واحد باز کرد و روی هر یک از آنها (با استفاده از اشاره گر فایل) نوشت یا از آنها خواند، در یک برنامه تحت شبکه می‌توان بطور همزمان چندین ارتباط فعال و باز داشت و با مشخصه هر یک از این ارتباطها روی هر کدام از آنها مبادله داده انجام داد.

۲) انواع سوکت و مفاهیم آنها

اگر بخواهیم از نظر اهمیت انواع سوکت را معرفی کنیم دو نوع سوکت بیشتر وجود ندارد. (انواع دیگری هم هستند ولی کم اهمیت ترند). این دو نوع سوکت عبارتند از :

- سوکتهای نوع استریم که سوکتهای اتصال گرا[۲] نامیده می‌شود.
- سوکتهای نوع دیتاگرام که سوکتهای بدون اتصال[۳] نامیده میشود.

اگر عادت به پیش داوری دارید برای تمایز بین مفهوم این دو نوع سوکت، تفاوت بین مفاهیم ارتباط نوع TCP و UDP را مد نظر قرار بدهید. روش ارسال برای سوکتهای نوع استریم همان روش TCP است و بنابراین داده‌ها با رعایت ترتیب و مطمئن با نظارت کافی برخطاهای احتمالی مبادله می‌شوند. سوکتهای نوع دیتاگرام نامطمئن است و هیچگونه تضمینی در ترتیب جریان داده‌ها وجود ندارد.

اکثر خدمات و پروتکلهائی که در لایه چهارم تعریف شده‌اند و در فصول بعدی آنها را بررسی می‌کنیم نیازمند حفظ اعتبار و صحت داده‌ها و همچنین رعایت ترتیب جریان داده‌ها هستند. بعنوان مثال پروتکل انتقال فایل (FTP)، پروتکل انتقال صفحات ابرمتن (HTTP) یا پروتکل انتقال نامه های الکترونیکی (SMTP) همگی

[۱] Transport Service Access Point
[۲] Connection Oriented
[۳] Connectionless

نیازمند برقراری یک ارتباط مطمئن هستند و طبعاً از سوکتهای نوع استریم بهره می‌برند.

همانگونه که قبلاً در مورد پروتکل TCP آموختیم پروتکلی است که داده‌ها را با رعایت ترتیب و خالی از خطا مبادله می‌نماید و پروتکل IP که در لایه زیرین آن واقع است با مسیریابی بسته‌ها روی شبکه سروکار دارد. سوکتهای نوع استریم دقیقاً مبتنی بر پروتکل TCP بوده و طبیعتاً قبل از مبادله داده‌ها باید یک اتصال[1] به روش دست‌تکانی سه‌مرحله‌ای[2] برقرار بشود.

سوکتهای نوع دیتاگرام مبتنی بر پروتکل UDP است و بدون نیاز به برقراری هیچ ارتباط و یا اتصال ، داده‌ها مبادله میشوند و بنابراین تضمینی بررسیدن داده‌ها ، صحت داده‌ها و تضمین ترتیب داده‌ها وجود ندارد ولی با تمام این مشکلات باز هم در برخی از کاربردها مثل انتقال صدا و تصویر یا سیستم DNS که قبلاً آنرا بررسی کردیم مورد استفاده قرار می‌گیرد. تنها حسن این روش سرعت انتقال داده‌ها می‌باشد.

در حقیقت شما با استفاده از سوکتها میخواهید یک ابزار برای استفاده از پروتکلهای TCP یا UDP در اختیار داشته باشید.

" سوکت یک مفهوم انتزاعی از تعریف ارتباط در سطح برنامه‌نویسی خواهد بود و برنامه‌نویس با تعریف سوکت عملاً تمایل خود را برای مبادله داده‌ها به سیستم عامل اعلام کرده و بدون درگیر شدن با جزئیات پروتکل TCP یا UDP از سیستم عامل می‌خواهد تا فضا و منابع مورد نیاز را جهت برقراری یک ارتباط ، ایجاد کند. "

۳) مفهوم سرویس دهنده /مشتری[۳]

دربرنامه نویسی شبکه این نکته قابل توجه است که هر ارتباطی دو طرفه می‌باشد یعنی عملاً ارتباط مابین دو پروسه تعریف می‌شود لذا طرفین ارتباط بایستی در لحظه شروع تمایل خود را برای مبادله داده‌ها به سیستم عامل اعلام کرده باشند. در هر ارتباط یکی از طرفین ، شروع کننده ارتباط تلقی می‌شود و طرف مقابل در صورت

---

[۱] Connection
[۲] Tree Way Handshake
[۳] Client/Server

آمادگی این ارتباط را می‌پذیرد. در صورت پذیرش ارتباط، مبادله داده‌ها امکان پذیر خواهد بود. اگر برنامه‌ای را که شروع کننده ارتباط است ، "برنامه مشتری" بنامیم قاعدتاً برنامه‌ای که این ارتباط را می‌پذیرد (و منتظر آن بوده) "سرویس دهنده" نام خواهد گرفت.

تعریف عمومی: مشتری (Client) پروسه‌ای است که اصولاً نیازمند مقداری اطلاعات است. سرویس دهنده (Server) پروسه ای است که اطلاعاتی را در اختیار دارد و تمایل دارد تا این اطلاعات را به اشتراک بگذارد و منتظر می‌ماند تا یک متقاضی ، واحدی از این اطلاعات را طلب کند و او آنرا تحویل بدهد.

بعنوان مثال وقتی سخن از سرویس دهنده وب[1] در میان است در یک عبارت ساده ، منظور سیستمی است که اطلاعاتی را در قالب صفحات وب[2] در اختیار دارد و در عین حال منتظر است که کسی تقاضای یکی از این صفحات را نموده و او این درخواست را اجابت کرده و داده‌های لازم را در پاسخ به این تقاضا ارسال نماید.

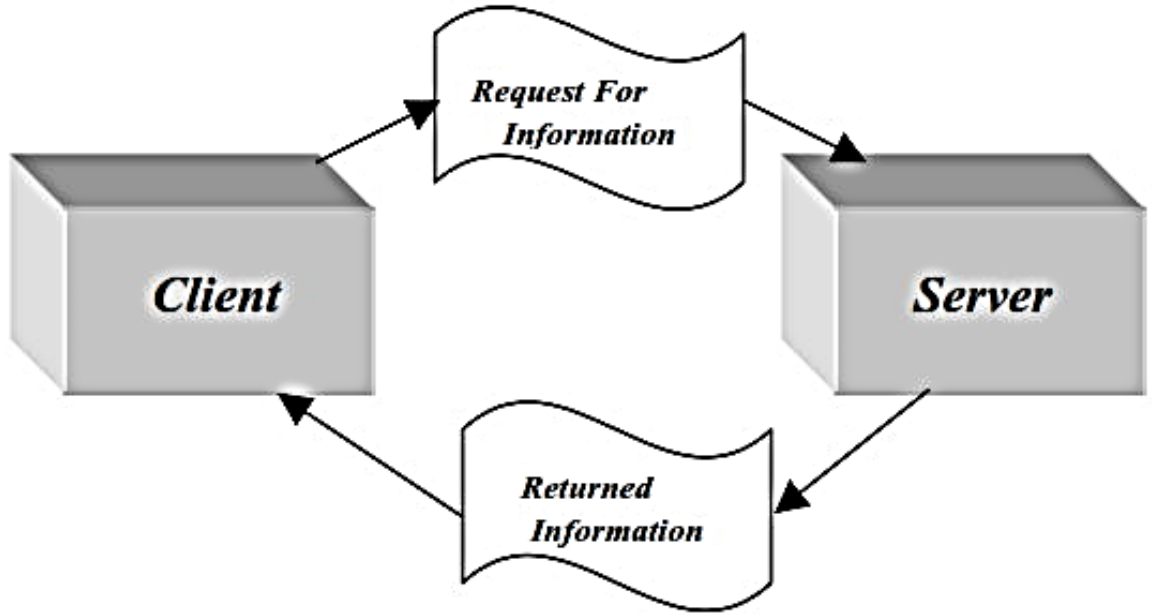
برنامه سمت سرویس دهنده[3] برنامه‌ای است که روی ماشین سرویس‌دهنده نصب میشود و منتظر است تا تقاضائی مبنی بر برقراری یک ارتباط دریافت کرده و پس از پردازش آن تقاضا ، پاسخ مناسب را ارسال نماید بنابراین در حالت کلی برنامه سرویس دهنده شروع کننده یک ارتباط نیست.

در طرف مقابل برنامه های سمت مشتری[4] هستند که بنابر نیاز ، اقدام به درخواست اطلاعات می‌کنند. تعداد مشتریها روی ماشینهای متفاوت یا حتی روی یک ماشین می‌تواند متعدد باشد و لیکن معمولاً تعداد سرویس دهنده ها یکی است. (مگر در سیستم های توزیع شده که مورد بحث ما نیستند.) برای مثال جلسه پرسش و پاسخی را در نظر بگیرید که یکنفر صاحب اطلاعات ، پاسخگو و منتظر سوال است – سمت سرویس‌دهنده –. در طرف دیگر تعدادی سوال کننده هستند که مختارند در رابطه با موضوع مورد بحث سوال نمایند – سمت مشتری–.

---

[1] Web Server
[2] Web Page
[3] Server Side
[4] Client Side

به نظر می‌رسد با دقت در مفهوم سرویس‌دهنده/ مشتری متقاعد بشوید که ساختار برنامه‌ای که در سمت سرویس دهنده در حال اجراست با برنامه ای که در سمت مشتری اجرا می‌شود ، متفاوت خواهد بود.



شکل (۱-۷) ارتباط بین سرویس دهنده و مشتری

قبل از آنکه وارد مقوله برنامه نویسی سوکت بشویم بد نیست الگوریتم کل کاری که بایستی در سمت سرویس دهنده و همچنین در سمت مشتری انجام بشود، بررسی نمائیم :

برنامه شما در سمت سرویس دهنده بـه عملیـات زیـر نیـازمند خواهـد بـود :

الف : یک سوکت را که مشـخصه یـک ارتبـاط اسـت ، بوجـود بیـاورید. تـا اینجـا فقـط به سیستم عامل اعـلام کرده‌ایـد کـه نیـازمند تعریـف یـک ارتبـاط هسـتید. در همیـن مرحلـه به سیستم عامل نوع ارتباط درخواستی خود را (TCP یا UDP) معرفـی می‌نمـائید. این کار در محیط سیسـتم عـامل یونیکـس توسـط تـابع سیسـتمی ()socket انجـام می‌شـود.

ب : بـه سـوکتی کـه بـاز کرده‌ایـد یـک آدرس پـورت نسـبت بدهیـد. ایـن کـار توسـط تـابع سیسـتمی ()bind انجـام می‌شـود و در حقیقـت بـا ایـن کـار بـه سیسـتم عـامل اعـلام می‌کنید کـه تمـام بسـته هـای TCP (یـا UDP) را کـه آدرس پـورت مقصدشـان بـا شـماره مورد نظر شما مطابقت دارد ، بـه سـمت برنامـه شـما هدایـت کنـد. در حقیقـت بـا ایـن کـار خودتان را بعنوان پذیرنـده دسـته ای از بسـته هـای TCP یـا UDP بـا شـماره پـورت خـاص معرفـی کرده‌ایـد. (دقـت کنیـد کـه در برنامـه سـمت سـرویس دهنـده اسـتفاده از دسـتور ()bind الزامـی اسـت)

ج : در مرحله بعد به سیستم عامل اعلام می‌کنید که کارش را برای پذیرش تقاضاهای ارتباط TCP شروع نماید. این کار توسط تابع سیستمی ()listen انجام می‌شود و چون ممکن است تعداد تقاضاهای ارتباط متعدد باشد باید حداکثر تعداد ارتباط TCP را که می‌توانید پذیرای آن باشید ، تعیین نمائید چرا که سیستم عامل باید بداند برای پذیرش ارتباطات TCP چقدر فضا و منابع شامل بافر در نظر بگیرد. دقت کنید که اعلام پذیرش تقاضاهای ارتباط به معنای پذیرش داده‌ها نیست بلکه فضای لازم را جهت ارسال و دریافت داده‌ها ایجاد می‌کنید. معمولاً تعیین تعداد ارتباطات TCP که میتواند بطور همزمان پذیرفته شده و به روش اشتراک زمانی[1] پردازش شود ، در اختیار شماست ولی باید این تعداد کمتر از مقداری باشد که سیستم عامل بعنوان حداکثر تعیین کرده است. بازهم یادآوری می‌کنیم که سرویس دهنده می‌تواند بصورت همزمان، چندین ارتباط متفاوت با چندین برنامه روی ماشینهای متفاوت را بصورت باز و فعال داشته باشد. بعنوان یک مقایسه با سیستم فایل تعداد حداکثر ارتباط باز را تعداد فایلی تصور کنید که می‌تواند توسط برنامه شما بطور همزمان باز شود.

د: نهایتاً با استفاده از تابع accetp() از سیستم عامل تقاضا کنید یکی از ارتباطات معلق را (در صورت وجود) به برنامه شما معرفی کند. تابع accetp() نکات ظریفی دارد که به تفصیل بررسی خواهد شد.

ه: از دستورات send() و recv() برای مبادله داده‌ها استفاده نمائید.

و: نهایتاً ارتباط را خاتمه بدهید. این کار به دو روش امکان پذیر خواهد بود:

- قطع ارتباط دو طرفه ارسال و دریافت (توسط تابع close())
- قطع یکطرفه یکی از عملیات ارسال یا دریافت (توسط تابع shutdown())

در برنامه سمت مشتری بایستی اعمال زیر انجام شود :

الف : یک سوکت را که مشخصه یک ارتباط است ، بوجود بیاورید. تا اینجا فقط به سیستم عامل اعلام شده است که نیازمند تعریف یک ارتباط هستید.

ب : در مرحله بعد لازم نیست همانند برنامه سرویس دهنده به سوکت خود آدرس پورت نسبت بدهید یعنی لزومی به استفاده از دستور bind() وجود ندارد چرا

---

[1] Time sharing

که برنامه سمت مشتری منتظر تقاضای ارتباط از دیگران نیست بلکه خودش متقاضی برقراری ارتباط با یک سرویس دهنده است. بنابراین در مرحله دوم به محض آنکه نیازمند برقراری ارتباط با یک سرویس دهنده شدید آن تقاضا را با استفاده از تابع سیستمی connect() به سمت آن سرویس دهنده بفرستید.

اگر مراحل دست تکانی سه مرحله ای را در برقراری یک ارتباط TCP بخاطر داشته باشید، دستور connect() عملاً متولی شروع و انجام چنین ارتباطی است.

مجدداً تاکید می‌کنیم از تابع bind() زمانی استفاده می‌شود که پذیرای ارتباطات TCP با شماره پورت خاصی باشید ولی در طرف مشتری چنین کاری لازم نخواهد بود چرا که برنامه سمت مشتری شروع کننده ارتباط است.

اگر عمل connect() موفقیت آمیز بود به معنای موفقیت در برقراری یک ارتباط TCP با سرویس دهنده است و می‌توانید بدون هیچ کار اضافی به ارسال و دریافت داده‌ها اقدام نمائید.

ج : از توابع send() , recv() برای ارسال یا دریافت داده‌ها اقدام نمائید.

د : ارتباط را با توابع close() یا shutdown() بصورت دوطرفه یا یکطرفه قطع نمائید.

پس از بررسی الگوریتم کلی برنامه های سمت سرویس دهنده و سمت مشتری وقت آن رسیده است که ساختمان داده‌ها و همچنین توابع و روالهای مورد نیاز در برنامه نویسی را با دقت بیشتری مورد بررسی قرار بدهیم.

۴) ساختمان داده‌های مورد نیاز در برنامه نویسی مبتنی بر سوکت

برای آغاز برنامه نویسی بهترین کار آنست که متغیرها و انواع ساختمان داده مورد نیاز در برنامه نویسی سوکت، تحت بررسی قرار بگیرد. (تمام قطعه کدها با C هستند)

اولین نوع داده ، "مشخصه سوکت[۱]" است که همانند اشاره‌گر فایل ، برای ارجاع به یک ارتباط باز مورد استفاده قرار می‌گیرد و یک عدد صحیح دوبایتی است یعنی با تعریف زیر ، متغیر **a** می‌تواند مشخصه یک سوکت باشد:

```
int    a;
```

---

[۱] Socket Descriptor

دومین نوع داده برای برقراری ارتباط ، یک استراکچر است که آدرس پورت پروسه و همچنین آدرس IP ماشین طرف ارتباط را درخود نگه می‌دارد. فعلاً در تعریفی ساده ساختار آن بصورت زیر است:

```
struct sockaddr {
        unsigned short sa_family;       /* address family, AF_xxxx */
        char sa_data[14];               /* 14 bytes of protocol address */
    };
```

sa_family : خانواده یا نوع سوکت را مشخص می‌کند. در حقیقت این گزینه تعیین می‌کند که سوکت مورد نظر را در چه شبکه و روی چه پروتکلی بکار خواهید گرفت؛ لذا در سیستمی که با پروتکلهای متفاوت و سوکتهای متفاوت سروکار دارد ، باید نوع سوکت درخواستی را تعیین کنید. فعلاً در کل این فصل که بحث ما شبکه اینترنت با پروتکل TCP/IP است خانواده سوکت را با ثابت AF_INET مشخص می‌کنیم. در مورد شبکه های دیگر مثل Appletalk این گزینه متفاوت خواهد بود.

sa_data : این چهارده بایت مجموعه ای است از آدرس پورت ، آدرس IP و قسمتی اضافی که باید با صفر پر شود و دلیل آنرا بعداً اشاره می‌کنیم.

همانگونه که اشاره شد این استراکچر بایستی آدرس پورت و آدرس IP را نگه دارد ولی در تعریف بالا چنین فیلدهائی مشاهده نمی‌شود. برای سادگی در برنامه نویسی ، استراکچر دیگری معرفی میشود که دقیقاً معادل استراکچر قبلی است ولی تعریف متفاوتی دارد و شما می‌توانید از هر کدام به دلخواه بهره بگیرید:

```
struct sockaddr_in {
        short int sin_family;           /* Address family */
        unsigned short int sin_port;  /* Port number */
        struct in_addr sin_addr;      /* Internet address */
        unsigned char sin_zero[8];   /* Same size as struct sockaddr */
        };
```

sin_family : همانند ساختار قبلی خانواده سوکت را تعیین می‌کند و برای شبکه اینترنت بایستی مقدار ثابت AF_INET داشته باشد.

sin_port : این فیلد دو بایتی ، آدرس پورت پروسه مورد نظر را مشخص می‌نماید.

in_addr : آدرس IP ماشین مورد نظر را مشخص می‌کند. این فیلد خودش یک استراکچر است که در ادامه تعریف خواهد شد ، فقط بدانید که کلاً عددی صحیح ، بدون علامت و چهاربایتی است.

sin_zero[8] : این هشت بایت در کاربردهای مهندسی اینترنت کلاً باید مقدار صفر داشته باشد. دلیل وجود این فیلد ، آنست که مفهوم سوکت برای تمام شبکه‌ها با پروتکلهای متفاوت ، بصورت معادل استفاده شده و بنابراین استراکچر فوق باید بگونه ای تعریف شود که برای تمام پروتکلهای شبکه قابل استفاده باشد. در شبکه اینترنت فعلاً آدرس IP چهاربایتی و آدرس پورت دوبایتی است در حالی که در برخی دیگر از شبکه ها طول آدرس بیشتر است. بنابراین هنگامی که از استراکچر فوق در کاربردهای برنامه نویسی شبکه اینترنت بهره می‌گیرید این هشت بایت اضافی است ولی حتماً باید با توابعی مثل memset() تماماً صفر شود.

دقت نمایید کـه دو اسـتراکچر قبلـی دقیقـاً معـادلند و می‌تـوان در فراخوانـی توابـع ، هـر کدام از آنها را با تکنیک "تطـابق نـوع[1]" بجـای دیگـری بکـار بـرد ولـی در مجمـوع اسـتفاده از تعریـف دوم راحـت ترخواهدبـود.

در تعریـف اسـتراکچر دوم یـک اسـتراکچر دیگـر بنـام in_addr تعریـف شـده کـه ساختار آن بصورت زیـر اسـت:

```
/* Internet IP address (a structure for historical reasons) */
 struct in_addr {
                unsigned long s_addr;
               };
```

ایـن فیلـد چـهاربایتی بـرای نگـهداری آدرس IP کـاربرد دارد و تعریـف آن بصـورت فوق کمی عجیب به نظـر می‌رسـد چـرا کـه می‌توانسـتیم در اسـتراکچر قبلـی بطـور مسـتقیم

---

[1] Casting

آنـرا unsigned long معرفـی کنیـم ولـی بـهر حـال بصـورت بالا تعریـف شـده اسـت. بـرای مقداردهـی بـه فیلدهـای بـالا می‌توانیـد از هـر روشـی کـه دلخـواه شماسـت اسـتفاده کنیـد ولیکن توابعی ساده برای این کار وجـود دارنـد کـه در ادامـه معرفـی خواهنـد شـد.

۵) مشکلات ماشینها از لحاظ ساختار ذخیره سازی کلمات در حافظه

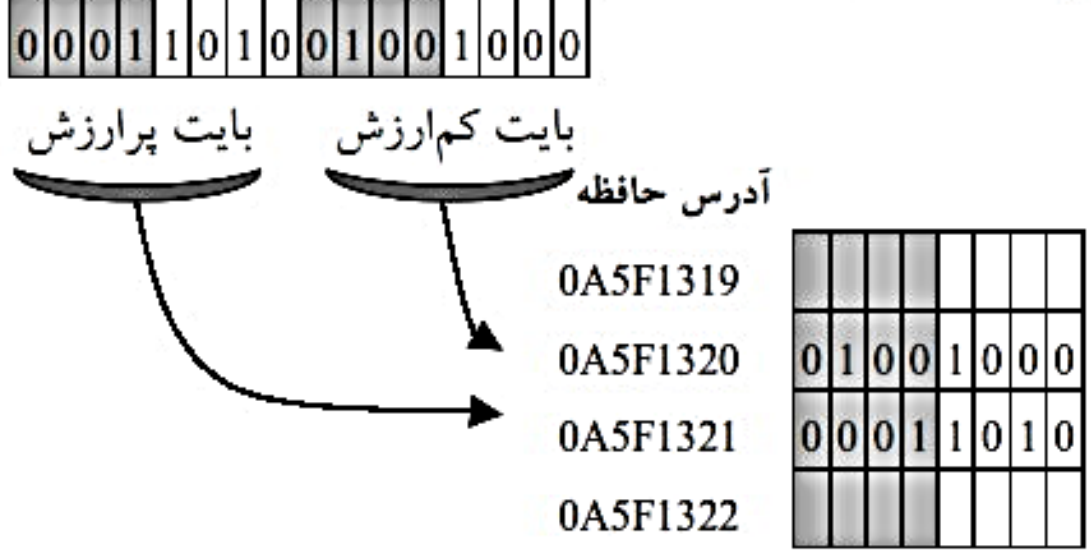
در گذشته تفاوت ماشـینهای نـوع BE[1] و نـوع LE[2] را بررسـی کردیـم و اشـاره شـد کـه در پروتکـل TCP/IP ترتیـب بایتـها بصـورت BE توافـق شـده اسـت لـذا وقتـی قـرار اسـت برنامـه شـما روی ماشـینی کـه سـاختار LE دارد نصـب شـود ترتیـب بایتـهای ارسـالی روی شـبکه بـهم خواهـد خـورد. بعنـوان مثـال وقتـی کـه روی ماشـینی از نـوع LE دسـتور زیـر اجرا می‌شـود:

```
struct sockaddr_in   as;
as.sin_port=0xB459;
```

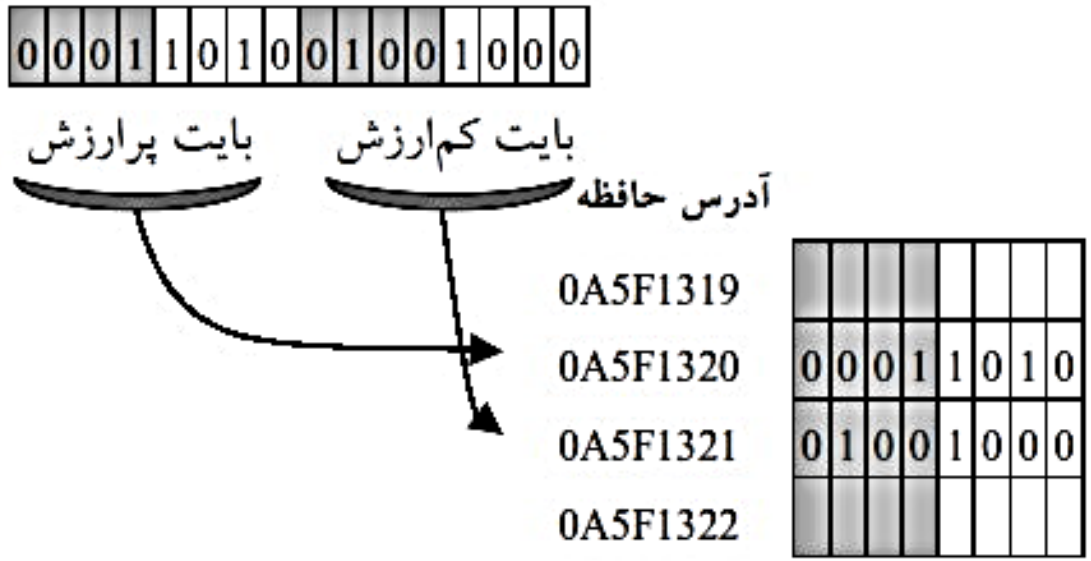
ماشینهای Big Endian و Little Endian

ساختار بسته‌های IP و TCP در قالب کلمات ۳۲ بیتی سازماندهی می‌شوند. ماشینهای متفاوتی که در دنیا وجود دارد کلمات دوبایتی و چهاربایتی (۳۲ بیتی) را به یک نحو درون حافظه ذخیره نمی‌کنند . در برخی از ماشینها برای ذخیره یک کلمه در حافظه ابتدا بایت کم ارزش در حافظه ذخیره شده و پشت سر آن بایت پرارزش ذخیره می‌شود ، در حالی که در برخی دیگر دقیقاً برعکس است. بعنوان مثال فرض کنید کلمهٔ ۲ بایتی (0x1A48) با دستور Mov به حافظهٔ اصلی منتقل شود. ماشینها به دو روش آنرا در حافظهٔ خود ذخیره می‌کنند :

- ماشینهایی که ابتدا بایت کم ارزش و سپس بایت پرارزش را ذخیره می‌کنند ، ماشینهای Little Endian نامیده می‌شوند. کامپیوترهای شخصی با پردازنده سری 80X86 و پنتیوم از این دسته هستند. در مثال زیر طریقهٔ ذخیره‌سازی یک کلمهٔ ۱۶ بیتی در فضای حافظهٔ یک ماشین .L.E به تصویر کشیده شده است.



➢ ماشینهایی که ابتدا بایت پرارزش و سپس بایت کم‌ارزش را ذخیره می‌کنند ، ماشینهای Big Endian نامیده می‌شود. کامپیوترهای سری SUN از این دسته هستند. در مثال زیر طریقهٔ ذخیره‌سازی یک کلمهٔ ١٦ بیتی در فضای حافظهٔ یک ماشین .B.E به تصویر کشیده شده است.



چون بایت کم ارزش اول ذخیره می‌شود و بعد از آن بایت پرارزش قرار میگیرد لذا پس از قرار گرفتن این دو بایت در بسته TCP آدرس پورت بصورت زیر (و قطعاً اشتباه) تنظیم خواهد شد:

59	B4

بنابراین وقتی قرار است برنامه نویس مقداری را درون فیلدی قرار بدهد که دو بایتی یا چهار بایتی است بایستی نگران نوع ماشین و ترتیب بایتها باشد. بهمین دلیل معرفی توابع زیر بعنوان ابزار کار برنامه نویس شبکه اینترنت ضروری است:

htons() : تابع تبدیل کلمات دوبایتی به حالت BE

htonl() : تابع تبدیل کلمات چهاربایتی به حالت BE

ntohs() : تابع تبدیل کلمات دوبایتی از BE به حالت فعلی ماشین

ntohl() : تابع تبدیل کلمات چهاربایتی از BE به حالت فعلی ماشین

---

[۱] Big Endian
[۲] Little Endian

برنامه نویس لازم است ساختار ماشین مورد استفاده جهت نصب نهایی برنامه اش را بداند تا در صورت LE بودن حتماً قبل از قرار دادن مقادیر در فیلدهای دوبایتی یا چهاربایتی از توابع فوق استفاده کند.

تذکر: فقط وقتی از توابع فوق استفاده می‌شود که نهایتاً فیلد مورد نظر در بسته TCP یا IP تنظیم شود. بعنوان مثال در استراکچر sock_addr_in در فیلد sin_family مقدار AF_INET که مقداری ثابت است قرار می‌گیرد و این فیلد فقط برای سیستم عامل تعریف شده و روی شبکه منتقل نخواهد شد لذا برای مقدار دهی به این فیلد لازم نیست از توابع فوق استفاده نمائیم. در ادامه با مثالهائی که خواهیم داشت با موارد استفاده توابع فوق آشنا می‌شویم.

۱-۵) مشکلات تنظیم آدرس IP درون فیلد آدرس

در مبحث آدرسهای IP آموختید که آدرسهای IP درقالب چهار فیلد هشت‌تایی[۱] ده دهی نوشته می‌شوند :

192.140.11.211

در حالی که در استراکچر sock_addr_in فیلد آدرس IP عددی است چهاربایتی که با یک عدد از نوع long پر می‌شود. بنابراین دو تابع زیر جهت انتساب آدرسهای IP با ساختار فوق الذکر کاربرد دارد:

تابع inet_addr() : این تابع یک رشته کاراکتری بفرم "187.121.11.44" را گرفته و به یک عدد چهاربایتی با قالب BE تبدیل می‌کند. مثال:

```
struct sockaddr_in  ina;
ina.sin_addr.s_addr=inet_addr("130.421.5.10")
```

در مثال بالا آدرس IP رشته‌ای است و پس از تبدیل به عددی چهاربایتی در قالب BE در فیلد مربوطه قرار می‌گیرد.

تابع inet_ntoa() : این تابع عکس عمل تابع قبلی را انجام می‌دهد یعنی یک آدرس چهاربایتی درقالب BE را گرفته و آنرا بصورت یک رشتهٔ کاراکتری که آدرس IP را

بصورت نقطه‌دار تعریف کرده ، تبدیل می‌نماید. پارامتر ورودی تابع فوق از نوع struct in_addr و خروجی آن نوع رشته ای است. به مثال زیر دقت کنید:

```
printf("%s",inet_ntoa(ina.sin_addr));
```

در مثال فوق محتوای آدرس IP بصورت رشته‌ای نقطه دار و در مبنای ده روی خروجی چاپ خواهد شد. مثلاً خروجی به فرم زیر است:

130.141.5.10

در بخشهای آتی چگونگی تبدیل آدرسهای حوزه بفرم www.ibm.com را به آدرس IP در محیط برنامه نویسی توضیح خواهیم داد. قبل از آن باید توابع لازم برای تعریف و برقراری ارتباط تعریف شوند.

۶) توابع مورد استفاده در برنامه سرویس دهنده (مبتنی بر پروتکل TCP)

۶-۱) تابع socket()

فرم کلی این تابع بصورت زیر است:

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

domain : این پارامترنشان‌دهندهٔ خانواده سوکت است و به نحوی که قبلاً اشاره شد در برنامه‌نویسی شبکهٔ اینترنت ، با مقدار ثابت AF_INET تنظیم می‌شود.

type : با این پارامتر نوع سوکت دلخواهتان را اعلام می‌کنید که می‌تواند نوع استریم یا از نوع دیتاگرام باشد. اگر سوکت دلخواهتان نوع استریم بود در فیلد type مقدار ثابت SOCK_STREAM قرار بدهید و اگر نوع دیتاگرام خواستید درآن مقدار SOCK_DGRAM تنظیم کنید.

protocol : در این فیلد شماره شناسائی پروتکل مورد نظرتان را تنظیم می‌کنید که برای کاربردهای شبکه اینترنت همیشه مقدار آن صفر است.

مقادیری که در فیلدهای اول و سوم قرار می‌دهید در برنامه نویسی تحت شبکه اینترنت همیشه ثابت خواهند بود.

مقدار بازگشتی توسط تابع socket() همان مشخصه سوکت است که از آن برای توابع بعدی استفاده خواهد شد (دقیقاً مثل اشاره گر یک فایل) لذا مشخصه سوکت بایستی تا زمانی که ارتباط خاتمه می‌یابد بدقت نگهداری شود.

اگر مقدار برگشتی تابع socket() ، ۱- باشد عمل موفقیت آمیز نبوده و روند کار باید متوقف شود و شما بعنوان برنامه نویس موظفید حتماً خروجی این تابع را بررسی کنید چرا که عملیات بقیه توابع که در ادامه معرفی خواهند شد به خروجی همین تابع بستگی دارد.

وقتی مقدار برگشتی تابع socket() مقدار ۱- باشد متغیر سراسری errno شماره خطای رخداده می‌باشد. برای پردازش شماره خطا تابع سیستمی perror() می‌تواند استفاده شود که روش بکارگیری آن در مثالها آمده است. این دو متغیر و تابع نیاز به تعریف ندارد و سیستمی هستند.

۲-۶) تابع bind()

وقتی سیستم عامل برای شما یک سوکت باز کرد در حقیقت شما فقط سنگ بنای یک ارتباط را بنا نهاده‌اید ولی هنوز هیچ کاری برای مبادله داده‌ها انجام نشده است. تابع bind() که معمولاً در برنامه سمت سرویس دهنده معنا می‌**یابد "عملی است جهت نسبت دادن آدرس پورت به یک سوکت باز شده**". این تعریف احتمالاً ابهام دارد پس به تعریف ساده زیر دقت کنید:

از طریق تابع bind() از سیستم عامل خواهش می‌کنید که تمام بسته های TCP یا UDP و همچنین تقاضاهای ارتباط با شماره پورت خاص را به سمت برنامه شما هدایت نماید.

بعنوان مثال وقتی گفته می‌شود که پروتکل HTTP به پورت 80 گوش می‌دهد به این معناست که برنامهٔ سرویس‌دهنده ، تمام بسته های TCP را که وارد ماشین محلی می‌شوند و شماره پورت مقصد آنها 80 است ، تحویل می‌گیرد و پردازش می‌نماید.

فرم کلی تابع bind() بصورت زیر است:

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

sockfd : همان مشخصه سوکتی است که قبلاً با استفاده از تابع socket() بازکرده‌اید. در حقیقت شما می‌خواهید به سوکت باز شده یک آدرس پورت نسبت بدهید.

my_addr : یک استراکچر که خانواده سوکت، آدرس پورت و آدرس IP ماشین محلی را در خود دارد. ساختار این استراکچر قبلاً تعریف شد.

addr_len : طول استراکچر my_addr بر حسب بایت

برای آشنایی با چگونگی استفاده از توابع فوق به قطعه کد زیر دقت کنید:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>

#define MYPORT 3490

main()
    {
    int sockfd;
    struct sockaddr_in my_addr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0))!= NULL){
      my_addr.sin_family = AF_INET;         /* host byte order */
      my_addr.sin_port = htons(MYPORT);  /* short, network byte order */
      my_addr.sin_addr.s_addr = inet_addr("132.241.5.10");
      bzero(&(my_addr.sin_zero), 8);          /* zero the rest of the struct */

     if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))!=-1){
      .
      .
      .
```

در مورد تابع bind() نکاتی وجود دارد که اشاره به آنها خالی از لطف نیست:

الف: در محیط یونیکس اگر فیلد آدرس پورت با مقدار صفر تنظیم شود آنگاه سیستم عامل در بین آدرسهای پورت از شماره ۱۰۲٤ تا ٦٥٥٣٥ ، یک شمارهٔ تصادفی انتخاب کرده و آنرا بعنوان شماره پورت در نظر می‌گیرد.

ب: برنامه کاربردی شما نباید شماره پورتی را بر گزیند که بین صفر تا ۱۰۲۳ باشد چرا که این شماره پورتها برای سرویس دهنده های استاندارد و سرویس دهنده

های یونیکس رزرو شده است و سیستم عامل اجازه استفاده از این شماره پورتها را به برنامه های کاربران نخواهد داد.

ج: در محیط یونیکس اگر فیلد آدرس IP را با مقدار ثابت INADDR_ANY تنظیم کنید ، آنگاه سیستم عامل بصورت خودکار آدرس IP ماشین محلی شما را استخراج و در آن قرار خواهد داد.

د: نکته ای که ممکن است بر آن خرده بگیرید آن است که چرا در مقداردهی فیلدهای بالا سعی نکردیم با بهره‌گیری از توابع htons() آن را به حالت BE تبدیل کنیم در حالی که چنین کاری لازم می‌باشد. دلیل آن بسیار ساده است: هر دو مقدار صفر دارند و حالت صفر نیازی به تبدیل ندارد.

ه: اگر شماره پورتی که انتخاب می‌کنید برنامه دیگری قبل از شما برای خود رزرو کرده باشد یعنی آنرا در برنامه خود به سوکتی bind() کرده باشد آنگاه عمل bind() موفقیت آمیز نبوده ومقدار (۱-) به برنامه شما باز خواهد گشت. برای پردازش نوع خطا ، متغیر سراسری errno شماره خطا و تابع perror() مشخصات خطا را بر می‌گرداند.

۳-۶) تابع listen()

این تابع فقط در برنامه سرویس دهنده معنا می‌یابد و در یک عبارت ساده اعلام به سیستم عامل برای پذیرش تقاضاهای ارتباط TCP است. به عبارت بهتر توسط این تابع به سیستم عامل اعلام می‌کنید که از این لحظه به بعد (یعنی زمان اجرای تابع) تقاضاهای ارتباط TCP ماشینهای راه دور با شماره پورت مورد نظرتان را به صف کرده و منتظر نگه دارد.

با توجه به آنکه ممکن است پس از راه اندازی برنامه سرویس‌دهنده ، در لحظاتی چندین پروسه متفاوت بطور همزمان تقاضای برقراری ارتباط TCP به یک آدرس پورت بدهند بنابراین سیستم عامل باید بداند که حداکثر چند تا از آنها را بپذیرد و ارتباط آنها را به روش دست تکانی سه مرحله ای برقرار نموده و آنها را در صف سرویس دهی قرار بدهد. توسط تابع listen() باید به سیستم عامل اعلام شود که حداکثر تعداد ارتباطات فعال و باز روی یک شماره پورت خاص چند تا باشد. فرم کلی تابع بصورت زیر است:

```
int listen(int sockfd, int backlog);
```

sockfd : همان مشخصه سوکت است که در ابتدا آنرا ایجاد کرده‌اید.
backlog : حداکثر تعداد ارتباطات معلّق و به صف شدهٔ منتظر. در بسیاری از سیستمها مقدار backlog به ۲۰ محدود شده است.

همانند توابع قبلی در صورت بروز خطا مقدار برگشتی این تابع ۱- خواهد بود و متغیر errno شماره خطای رخداده می‌باشد.

۴-۶) تابع accept()

این تابع اندکی مرموز به نظر می‌رسد و بایستی به مفهوم آن دقت شود :
پس از آنکه تابع listen() اجرا شد تقاضای ارتباط TCP پروسه های روی ماشینهای راه دور (درصورت وجود) پذیرفته ، به صف شده و معلق نگاه داشته میشود. وقتی که تابع accept() اجرا می‌شود در حقیقت برنامه شما از سیستم عامل تقاضا می‌کند که از بین تقاضاهای به صف شده یکی را انتخاب کرده و آنرا با مشخصات پروسه طرف مقابل تحویل برنامه بدهد. بنابراین برنامه باید از بین ارتباطات معلق یکی را به

حضور بطلبد تا عملیات لازم را انجام بدهد. بهمین دلیل سیستم عامل یک مشخصه سوکت جدید ایجاد کرده و آنرا به برنامه بر می‌گرداند. در اینجا شما یک سوکت جدید دارید. مشخصه سوکت اول که توسط تابع socket() بدست آمده و مشخصه سوکت دوم که با تابع accept() به برنامه شما برگشته است. تفاوت این دو سوکت در چیست؟

الف: از سوکت اول برای پذیرش یکی از ارتباطات معلق در دستور accept() استفاده می‌کنید. در حقیقت این سوکت مشخصه کل ارتباطات به صف شدهٔ منتظر میباشد.

ب: از سوکت دوم برای دریافت و ارسال اطلاعات روی یکی از ارتباطات معلق استفاده می‌کنید. این سوکت مشخصه یکی از ارتباطات به صف شده می‌باشد.

فرم کلی تابع به صورت زیر است :

```
#include <sys/socket.h>

int accept(int sockfd, void *addr, int *addrlen);
```

sockfd : مشخصه سوکت است که در ابتدا با تابع socket() بدست آمده است.

addr : اشاره‌گر به استراکچری است که شما آنرا بعنوان پارامتر به این تابع ارسال می‌کنید تا سیستم عامل پس از پذیرش یک ارتباط معلق آدرس پورت و آدرس IP طرف مقابل ارتباط را در آن به برنامه شما برگرداند. ساختار این استراکچر قبلاً معرفی شد.

addrlen : طول استراکچر addr بر حسب بایت

مقدار برگشتی این تابع یک مشخصه سوکت است که در روالهای بعدی مورد استفاده قرار می‌گیرد. اگر مقدار برگشتی (۱-) باشد خطائی رخداده است که شماره آن خطا در متغیر سراسری errno قابل بررسی است.

مثال ناتمام زیر برای روشن شدن کلیت کار بسیار سودمند خواهد بود:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#define MYPORT 3490    /* the port users will be connecting to */
#define BACKLOG 10     /* how many pending connections queue will hold */

main()
{
   int sockfd, new_fd;  /* listen on sock_fd, new connection on new_fd */
   struct sockaddr_in my_addr;    /* my address information */
   struct sockaddr_in their_addr; /* connector's address information */
   int sin_size;

   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0))!= NULL){

   my_addr.sin_family = AF_INET;         /* host byte order */
   my_addr.sin_port = htons(MYPORT);     /* short, network byte order */
   my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
   bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the struct */

   if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))!=-1){

   listen(sockfd, BACKLOG);

   sin_size = sizeof(struct sockaddr_in);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   .
   .
   .
```

بـار دیگـر تـأکید می‌کنیـم کـه بـرای ارسـال یـا دریـــافت داده‌هـــا بایســـتی از ســـوکت جدیدی که مشخصه آن توســـط تـابع ()accept برمـی‌گـردد ، اسـتفاده کنیـد.

۶-۵) توابع ()send و ()recv

ایـن دو تـابع در برنامـه سـمت سـرویس دهنـده و برنامـه سـمت مشـتری قـابل اسـتفاده بوده و برای مبادله داده‌ها کاربرد دارند. فـرم کلـی دو تـابع بـه صـورت زیـر اسـت :

```
int send(int sockfd, const void *msg, int len, int flags);

int recv(int sockfd, void *buf, int len, unsigned int flags);
```

sockfd : مشخصه سـوکتی کـه از تـابع ()accept بدسـت آمـده اسـت.
msg : محلی در حافظه (مثـل آرایـه یـا اسـتراکچر) کـه داده‌هـای ارسـالی از آنجـا اسـتخراج شده و درون فیلد داده[1] از یـک بسـته TCP قـرار گرفتـه و ارسـال می‌شـوند.
len : طول داده‌های ارسالی یـا دریـافتی بـر حسـب بـایت
flag : برای پرهـیز از پیچیـدگـی بحـث در ایـن مـورد توضیـح نمی‌دهیـم. فقـط در آن صفـر بگذاریـد.

buf : این پارامتر در تابع recv() آدرس محلی در حافظه است که داده‌های دریافتی در آنجا قرار گرفته و به برنامه باز گردانده می‌شود.

مقدار برگشتی این دو تابع در صورت بروز هر گونه خطا ۱- خواهد بود ولی در صورت برگشت یک عدد مثبت ، تعداد بایتهای ارسالی یا دریافتی را بر حسب بایت مشخص می‌کند. دقت کنید که ممکن است تعداد بایتهای ارسالی یا دریافتی با تعدادی که در متغیر len تقاضا داده‌اید یکسان نباشد. بعنوان مثال فرض کنید شما در متغیر len مقدار ۱۰۰۰ قرار داده‌اید ولی مقداری که تابع برگردانده است ۲۰۰ باشد. در این صورت ۸۰۰ بایت از کل داده‌های ارسالی (یا دریافتی) باقی مانده است که برنامه شما باید تکلیف آنها را مشخص کند.

---

[۱] Payload

توصیه : در هر مرحله سعی کنید حجم داده‌هائی که توسط تابع send() ارسال می‌کنید حول و حوش یک کیلو بایت باشد.

نکته : توابع send() , recv() فقط برای ارسال و دریافت روی سوکتهای نوع استریم کاربرد دارد ولی اگر می‌خواهید به روش UDP و با سوکتهای دیتاگرام داده‌هایتان را ارسال کنید اندکی صبر کنید؛

۶-۶) توابع close() و shutdown()

تا زمانی که نیاز داشتید میتوانید یک ارتباط را باز نگه‌داشته و داده ارسال یا دریافت نمائید ولیکن همانند فایلها هرگاه نیازتان برطرف شد باید ارتباط را ببندید.

فرم کلی تابع close() بصورت زیر است :

```
close(int sockfd);
```

sockfd : مشخصه سوکت مورد نظر. این سوکت همان مشخصه ای است که تابع accept() برگردانده است. دقت کنید که اگر sockfd مشخصه ای باشد که توسط تابع socket() برگشته است تمام ارتباطات معلق و منتظر نیز بسته خواهد شد.

ارتباطی که توسط تابع close() بسته می‌شود دیگر برای ارسال و دریافت قابل استفاده نخواهد بود.

هر گاه سوکتی را ببندید در حقیقت یکی از ارتباطات TCP را بسته اید و سیستم عامل می‌تواند بجای آن تقاضای ارتباط دیگری را قبول کرده ، برای پردازش به صف ارتباطات معلق اضافه کند.

راه دیگر بستن یک سوکت تابع shutdown() می‌باشد که فرم کلی آن بصورت زیر است :

```
int shutdown(int sockfd, int how);
```

sockfd : مشخصه سوکت مورد نظر

how : روش بستن سوکت که یکی از سه مقدار زیر را می‌پذیرد:

- مقدار صفر: دریافت داده را غیر ممکن می‌سازد ولی سوکت برای ارسال داده ، همچنان باز است. سیستم عامل بافر ورودی[1] مربوط به آن سوکت را آزاد می‌کند.
- مقدار ۱ : ارسال داده را غیر ممکن می‌سازد در حالی که سوکت برای دریافت داده‌ها همچنان باز است. سیستم عامل بافر خروجی[2] مربوط به آن سوکت را آزاد می‌کند.

- مقدار ۲: ارسال و دریافت را غیر ممکن کرده سوکت کاملاً بسته می‌شود. این حالت دقیقاً همانند تابع close() عمل می‌نماید.

همانند توابع قبلی در صورت بروز خطا مقدار برگشتی این توابع ۱- خواهد بود و متغیر سراسری errno شماره خطا را برای پردازش مشخص می‌کند.

۷) توابع مورد استفاده در برنامه مشتری (مبتنی بر پروتکل TCP)

تا اینجا توابعی که معرفی شدند توابع پایه‌ای بودند که در سمت سرویس دهنده به نحوی استفاده می‌شوند. حال باید ببینیم در سمت مشتری چه توابعی مورد استفاده قرار می‌گیرند:

الف : ابتدا دقیقاً مانند برنامه سرویس دهنده یک سوکت بوجود بیاورید. برای اینکار از تابع socket() که در بخش قبلی معرفی شد استفاده کنید. تا اینجا هیچ تفاوتی برای بکارگیری این تابع در سمت سرویس دهنده و سمت مشتری وجود ندارد.

ب : در هنگام نیاز مستقیماً تقاضای برقراری ارتباط را به سمت سرویس دهنده بفرستید و آنقدر منتظر شوید تا این تقاضا پذیرفته شود. این عمل توسط تابع connect() انجام می‌شود که در ادامه توضیح داده خواهد شد.

ج – از توابع send() و recv() برای ارسال و دریافت داده‌ها استفاده کنید.

د– نهایتاً ارتباط ایجاد شده را توسط تابع close() یا shutdown() ببندید.

---

[۱] Inbound buffer
[۲] Outbound buffer

۱-۷) تابع connect()

برای برقراری ارتباط با یک سرویس دهنده از تابع connect() استفاده می‌شود و در صورتی که برنامه سرویس دهنده روی ماشین مورد نظر اجرا شده باشد و توابع listen() و accept() در برنامه فراخوانی شده باشند آنگاه نتیجه تابع connect() موفقیت آمیز خواهد بود.

فرم کلی تابع connect() به صورت زیر است :

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

sockfd : مشخصه سوکتی است که با فراخوانی تابع socket() بدست آمده است.

serv_addr : استراکچری از نوع sockaddr است که قبلاً معرفی شد. در این استراکچر آدرس IP ماشین مقصد و آدرس پورت برنامه مقصد تعیین خواهد شد.

addrlen : اندازه استراکچر قبلی را بر حسب بایت معرفی می‌کند و می‌توان براحتی در این پارامترمقدار sizeof(struct sockaddr) قرار داد.

به این نکته دقت کنید که شما آدرس پورت خودتان را تنظیم نمی‌کنید بلکه سیستم عامل بطور خودکار یک شماره پورت تصادفی برای شما انتخاب می‌کند و مقدار این شماره برای برنامه سمت مشتری اصلاً مهم نیست چرا که وقتی شما به یک سرویس دهنده متصل می‌شوید و سرویس دهنده این تقاضا را می‌پذیرد پاسخ سرویس دهنده به همان آدرس پورتی خواهد بود که سیستم عامل برای سوکت انتخاب کرده است. در حقیقت برنامه شما بعنوان شروع کنندهٔ ارتباط ، آدرس پورت خود را نیز به طرف مقابل اعلام می‌کند. در مقابل آدرس پورت برنامه سرویس دهنده قطعاً باید ثابت و مشخص باشد تا برنامه مشتریها بتوانند ارتباط را شروع نمایند.
در صورت عدم موفقیت در برقراری یک ارتباط TCP مقدار برگشتی این تابع ۱- خواهد بود و متغیر errno شماره خطای رخداده می‌باشد.

مثال ناتمام زیر تا حدودی این دیدگاه را به شما عرضه می‌کند:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#define DEST_IP   "132.241.5.10"
#define DEST_PORT 23

main()    {

      int sockfd;
      struct sockaddr_in dest_addr;   /* will hold the destination addr */

      if (( sockfd = socket(AF_INET, SOCK_STREAM, 0))!=NULL) {

       dest_addr.sin_family = AF_INET;             /* host byte order */
       dest_addr.sin_port = htons(DEST_PORT);  /* short, network byte order */
       dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
       bzero(&(dest_addr.sin_zero), 8);              /* zero the rest of the struct */

     if ((connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)))!=-1) {
       .
       .
       .
```

۸) ارسال و دریافت به روش UDP با سوکتهای دیتاگرام

توابع ارسال ، دریافت و پذیرش برای سوکتهای نوع استریم کاربرد دارد. حال باید دید که به چه صورت می‌توان ارسال و دریافت را به روش UDP روی سوکتهای نوع دیتاگرام انجام داد.

- **برنامه سمت سرویس دهنده**

الف : یک سوکت از نوع دیتاگرام ایجاد کنید. این کار با فراخوانی تابع socket() با پارامتر SOCK_DGRAM انجام می‌شود.

ب : به سوکت ایجاد شده آدرس پورت مورد نظرتان را نسبت بدهید. (با تابع bind())

ج : بدون هیچ کار اضافی میتوانید منتظر دریافت داده‌ها بشوید. (تا موقعی که داده‌ای دریافت نشود ارسال معنی نمی‌دهد.) وقتی داده‌ای دریافت و پردازش شد آدرس برنامه مبدا (آدرس IP و پورت) مشخص شده و ارسال امکان پذیر خواهد بود.

ارسال و دریافت روی سوکتهای نوع دیتاگرام بوسیله توابع recvfrom() و sendto() انجام می‌شود.

د: نهایتاً سوکت ایجاد شده را ببندید.

- **برنامه سمت مشـــتری**

الـف: یـک سـوکت از نـوع دیتـــاگـــرام ایجـــاد کنیـــد. (بـــا تـــابع ()socket و پـــارامتر SOCK_DGRAM)

ب: هر گاه نیـاز شـد بـدون هیـچ کـار اضـافی داده‌هایتـان را بـه سـمت سـرویس دهنـده ارسـال نمـایید. تـا وقتـی کـه بـه سـمت سـرویس دهنـده ارسـالی نداشـته باشـید، دریـــافت داده‌ها معنا نمی‌دهد چرا کـــه شـما بـرای سـرویس دهنـده شـناخته شـده نیسـتید مگـر اینکـه داده‌ای را ارسال نمائید. ارسال و دریافت را تـا زمـانی کـه نیـاز اسـت انجـام بدهیـد.

ج: سوکت ایجـاد شـده را ببندیـد.

فرم کلی تابع ارسال داده مبتنی بر ســوکتهای دیتـاگـرام بصـورت زیـر اسـت:

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to,
    int tolen);
```

sockfd : مشخصه سوکت دیتاگرام کـــه بـا تـابع ()socket بوجـود آمـده اسـت.

msg : آدرس محـل قـرارگرفتـن پیـام در حافظـه کـه داده‌هـــای ارســـالی بایســـتی از آنجـــا استخراج شده و درون یـک بسـته UDP قـرار گرفتـه و ارسـال شـود.

len : طول پیام ارسالی بـــر حسـب بـایت

flags : برای پرهیز از پیچیدگی بحث فعلاً آنرا صفر در نظر بگیرید.
to : استراکچری از نوع sockaddr که قبلاً ساختار آنرا مشخص کردیم. در این استراکچر باید آدرس IP مربوط به ماشین مقصد و همچنین شماره پورت سرویس دهنده تنظیم شود.
tolen : طول استراکچرsockaddr است که به سادگی می‌توانید آنرا به مقدار sizeof(struct sockaddr) تنظیم نمایید.

مقدار برگشتی این تابع همانند تابع send() تعداد بایتی است که سیستم عامل موفق به ارسال آن شده است. دقت کنید که اگر مقدار برگشتی (1-) باشد خطائی بروز کرده که می‌توانید شماره خطا را در متغیر سراسری errno بررسی نمائید. باز هم تکرار می‌کنیم دلیلی ندارد تعداد بایتی که تقاضای ارسال آنها را داده‌اید با تعداد بایتی که ارسال شده یکی باشد. بنابراین حتماً این مورد را در برنامه خود بررسی کرده و همچنین تقاضاهای ارسال در هر مرحله را نزدیک یک کیلو بایت در نظر بگیرید.

فرم کلی تابع دریافت داده مبتنی بر سوکتهای دیتاگرام بصورت زیر است:

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,struct sockaddr *from,
        int  *fromlen);
```

sockfd : مشخصه سوکت دیتاگرام که با تابع socket() بوجود آمده است.
buf : آدرس محلی از حافظه که سیستم عامل داده‌های دریافتی را در آن محل قرار خواهد داد.
len : طول پیامی که باید دریافت شود (بر حسب بایت)
from : استراکچری است از نوع sockaddr که قبلاً ساختار آن بررسی شد و سیستم عامل آنرا با مشخصات آدرس IP و آدرس پورت برنامه مبداء تنظیم و به برنامه شما برمی‌گرداند.
flag : آنرا به صفر تنظیم کنید.
len : طول استراکچری است که سیستم عامل آنرا برگردانده است.

مقدار برگشتی این تابع نیز تعداد بایتی است که دریافت شده است. این پارامتر برای پردازش داده‌های دریافتی اهمیت حیاتی دارد.

۹) توابع مفید در برنامه نویسی شبکه

بغیر از توابع سیستمی معرفی شده توابع دیگری هم هستند که برای برنامه نویسی شبکه بسیار مفید و کارآمد هستند. در ادامه برخی از مهمترین آنها را تشریح خواهیم کرد:

۹-۱) تابع getpeername()

```
#include <sys/socket.h>

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

با استفاده از این تابع می‌توانید هویت طرف مقابل ، شامل آدرس IP و آدرس پورت پروسهٔ طرف مقابل ارتباط را استخراج نمائید. پارامترهای این تابع بصورت ذیل تعریف شده است :

sockfd : مشخصه سوکت مورد نظر

addr : استراکچری است از نوع sockaddr که قبلاً ساختار آن تعریف شده است. این استراکچر توسط سیستم عامل با آدرس IP و آدرس پورت طرف مقابل پر شده است.

addrlen : طول استراکچرsockaddr

در صورت عدم موفقیت تابع فوق مقدار برگشتی (۱-) خواهد بود و در متغیر سراسری errno شماره خطا برای بررسی نوع خطا تنظیم خواهد شد.

نکته ای که ممکن است برنامه نویس فراموش کند آن است که ترتیب آدرس IP و آدرس پورت بصورت BE است و اگر ماشین شما از نوع LE است باید حتماً آنرا تبدیل کنید.

۲-۹) تابع gethostname()

این تابع نام ماشینی را که برنامهٔ شما روی آن اجرا می‌شود ، بر خواهد گرداند. این نام یک رشته کاراکتری معادل با نام نمادین ماشین است نه آدرس IP آن (مثلاً www.ibm.com). فرم کلی تابع بصورت زیر است:

```
#include <unistd.h>

int gethostname(char *hostname, size_t size);
```

hostname : یک آرایه از کاراکترها (یا بعبارت بهتر یک رشته کاراکتری) است که پس از بازگشت تابع نام ماشین در آنجا ذخیره خواهد شد.

size : طول رشته کاراکتری بر حسب کاراکتر

اگر مقدار برگشتی (۱-) باشد خطائی بروز کرده و مقدار errno همانند قبل شماره خطا را نگه می‌دارد ولی اگر تابع فوق موفق عمل کند مقدار برگشتی صفر خواهد بود.

۳-۹) بکارگیری سیستم DNS برای ترجمه آدرسهای حوزه

قبلاً در مورد سیستم DNS و طرز عملکرد آن بحث شد. در اینجا وقت آن فرا رسیده است که بتوانید در محیط برنامه نویسی تقاضای ترجمه نام حوزه[۱] یک سرویس دهنده را به این سیستم ارائه کرده و نتیجه را در برنامه خود استفاده نمائید.

---

[۱] Domain Name

مثالهای کوچکی که تا اینجا داشته ایم همگی برای برقراری یک ارتباط با ماشین خاص مستقیما از آدرس IP آن استفاده می‌کردند و لیکن فرض کنید که شما بخواهید برنامه ای بنویسید که کاربر بتواند آدرس نام حوزه یک سرویس دهنده را بعنوان آدرس مقصد وارد نماید. تابعی که در این مورد بکار می‌آید دارای فرم کلی زیر است:

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

name : رشته کاراکتری نام حوزه یک سرویس دهنده

مقدار برگشتی تابع ، آدرس استراکچری است از نوع hostent که ساختار آن بصورت زیر تعریف شده است :

```
struct hostent {
      char   *h_name;
      char   **h_aliases;
      int    h_addrtype;
      int    h_length;
      char   **h_addr_list;
   };

#define h_addr h_addr_list[0]
```

hname : نام رسمی ماشین (برای شبکه اینترنت این رشته نام حوزه خواهد بود مثلاً (www.ibm.com

h_aliases : نام مستعار ماشین (این رشته با \0 ختم می‌شود)

h_addrtype : خانواده آدرس (همانگونه که اشاره شد در شبکه اینترنت این فیلد مقدار AF_INET خواهد داشت.)

h_length : طول آدرس بر حسب بایت

h_addr_list : یک رشته کاراکتری که در آن آدرس IP مربوط به ماشین سرویس دهنده قرار دارد. این رشته با \0 ختم می‌شود.

دقت کنید که در تابع بالا در صورت موفقیت آمیز بودن، یک اشاره گر به استراکچر بر می‌گرداند و در غیر اینصورت مقدار NULL برخواهد گشت و برخلاف

توابع قبلی متغیر errno تنظیم نخواهد شد و بجای آن متغیر سراسری herror که متغیری سیستمی است تنظیم می‌شود و در ضمن تابع سیستمی herror() برای کشف نوع خطا بکار گرفته می‌شود.

برای رفع ابهاماتی که در این تابع وجود دارد طرح یک مثال ضروری به نظر می‌رسد. به برنامه کوچک و اجرائی زیر دقت نمائید :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
```

```
    struct hostent *h;

    if (argc != 2) {  /* error check the command line */
       fprintf(stderr,"usage: getip address\n");
       exit(1);
    }

    if ((h=gethostbyname(argv[1])) == NULL) {  /* get the host info */
       herror("gethostbyname");
       exit(1);
    }

    printf("Host name  : %s\n", h-h_name);
    printf("IP Address : %s\n",inet_ntoa(*((struct in_addr *)h-h_addr)));

    return 0;
}
```

نام این برنامه getip است که یک آدرس نام حوزه را بعنوان ورودی دریافت کرده و نتیجه ترجمه آن را به آدرس IP و بقیه مشخصات را روی خروجی چاپ می‌کند.
نکات زیر درمورد برنامه بالا ارزش بازگوئی دارد:

الف: طریقه بکارگیری برنامه فوق بدینصورت است که نام برنامه را روی خط فرمان تایپ کرده و سپس در جلوی آن نام حوزه را با یک فاصله خالی نوشته و کلید Enter را فشار می‌دهید. مثال :

ب: آدرس IP معادل با آدرس نام حوزه در متغیر h_addr_list واقع است و هر چند که بصورت یک رشته است که با کد 0\ ختم می‌شود ولی برای شبکه اینترنت که آدرسهای IP فعلاً چهار بایتی هستند شما فقط به چهار بایت اول آن که بصورت BE ذخیره شده‌اند نیازمندید. در برنامه فوق برای تبدیل آدرس چهاربایتی به حالت رشته ای نقطه دار بفرم (مثلاً 213.190.140.187) ازتابع inet_ntoa() برای چاپ روی خروجی بهره گرفته شده است.

ج: عمل "تطبیق نوع" در تابع inet_ntoa() به آن دلیل بوده است که طبق تعریف اصلی متغیر h→h_addr بصورت رشته معمولی تعریف شده ولی در تابع inet_ntoa() آرگومان ورودی آن یک استراکچر از نوع in_addr است که در ابتدای فصل ساختار آن تعریف شد و چهاربایتی است. بنابراین مجبوریم با عمل "تطبیق نوع" سازگاری پارامتر ورودی را تضمین کنیم ولی در عمل اتفاق خاصی نمی‌افتد.