دانشکده فنی و مهندسی

طراحی و پیاده‌سازی کلاس ارتباط شبکه به کمک سوکت‌ها در سیستم عامل Linux با زبان برنامه‌نویسی C++

پایان‌نامه برای دریافت درجه کارشناسی

در رشته مهندسی تکنولوژی کامپیوتر گرایش نرم‌افزار

بن‌ایل عیسوی

استاد راهنما:

دکتر علیرضا مهینی

شهریورماه ۱۳۹۰

دانشکده فنی و مهندسی

طراحی و پیاده‌سازی کلاس ارتباط شبکه به کمک سوکت‌ها در سیستم عامل Linux با زبان برنامه‌نویسی C++

پایان‌نامه برای دریافت درجه کارشناسی

در رشته مهندسی تکنولوژی کامپیوتر گرایش نرم‌افزار

بن‌ایل عیسوی

استاد راهنما:

دکتر علیرضا مهینی

شهریورماه ۱۳۹۰


بسم الله الرحمن الرحيم

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه/رساله

نام دانشکده: دانشکده فنی و مهندسی

نام دانشجو: بن‌ایل عیسوی

عنوان پایان‌نامه یا رساله: طراحی و پیاده‌سازی کلاس ارتباط شبکه به کمک سوکت‌ها در سیستم عامل Linux با زبان برنامه‌نویسی C++

تاریخ دفاع:

رشته: مهندسی تکنولوژی کامپیوتر

گرایش: نرم‌افزار

ردیف	سمت	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امضا
۱	استاد راهنما				
۲	استاد راهنما				
۳	استاد مشاور				
۴	استاد مشاور				
۵	استاد مدعو خارجی				
۶	استاد مدعو خارجی				
۷	استاد مدعو داخلی				
۸	استاد مدعو داخلی				

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب بن‌ایل عیسوی به شماره دانشجویی ۸۸۴۴۱۲۵۵۱۷ دانشجوی رشته مهندسی تکنولوژی کامپیوتر گرایش نرم‌افزار در مقطع تحصیلی کارشناسی ناپیوسته تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی:

امضا و تاریخ:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

☐ بهره‌برداری از این پایان‌نامه/ رساله برای همگان بلامانع است.

☐ بهره‌برداری از این پایان‌نامه/ رساله با اخذ مجوز از استاد راهنما، بلامانع است.

☐ بهره‌برداری از این پایان‌نامه/ رساله تا تاریخ ..... ممنوع است.

نام استاد یا اساتید راهنما:

تاریخ:

امضا:

تقدیم به مادرم یگانه فردی که؛

در این مدت حمایتش را از من دریغ نکرد و

موجب انگیزه‌ای در درونم برای اتمام کاری شد

که آغازگرش بودم.

چکیده

برنامه‌نویسی تحت شبکه یکی از مقولات بسیار قدیمی و مهم در زمینه کاربردهای کامپیوتر می‌باشد. با گسترش فناوریهای روز مبتنی بر تبادل اطلاعات، روند رو به رشد این نوع برنامه‌نویسی مشهود است. پیشرفتهای به وجود آمده در این زمینه باعث افزایش توجه به دو گزینه مهم یعنی امنیت و کاربر پسند بودن شده است. تولید نرم‌افزارهای با کیفیت بالا بسیار آسان شده است. عناصر اجتماعی موجب گرایش برنامه‌نویسان به سمت برنامه‌نویسی تحت شبکه شده است. موضوعی که باعث شد این پروژه آغاز شود...

واژه‌های کلیدی: برنامه‌نویسی تحت شبکه، برنامه‌نویسی سوکت، امنیت در شبکه، لینوکس و برنامه نویسی تحت آن

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل ۱:

مقدمه

۱-۱- شبکه کامپیوتری

شبکه‌های کامپیوتری مجموعه‌ای از کامپیوترهای مستقل متصل به یکدیگرند که با یکدیگر ارتباط داشته و تبادل داده می‌کنند. مستقل بودن کامپیوترها بدین معناست که هر کدام دارای واحدهای کنترلی و پردازشی مجزا بوده و بود و نبود یکی بر دیگری تاثیرگذار نیست.

متصل بودن کامپیوترها یعنی از طریق یک رسانه فیزیکی مانند کابل، فیبر نوری، ماهواره‌ها و ... به هم وصل می‌باشند.

۱-۱-۱- انواع شبکه‌های رایانه‌ای از نظر اندازه

۱-۱-۱-۱- شبکه شخصی (PAN)

«شبکه شخصی» (Personal Area Network) یک «شبکه رایانه‌ای» است که برای ارتباطات میان وسایل رایانه‌ای که اطراف یک فرد می‌باشند (مانند «تلفن»ها و «رایانه‌های جیبی» (PDA) که به آن «دستیار دیجیتالی شخصی» نیز می‌گویند) بکار می‌رود.

۱-۱-۱-۲- شبکه محلی (LAN)

«شبکه محلی» (Local Area Network) یک «شبکه رایانه‌ای» است که محدوده جغرافیایی کوچکی مانند یک خانه، یک دفتر کار یا گروهی از ساختمان‌ها را پوشش می‌دهد.

۱-۱-۱-۳- شبکه کلان‌شهری (MAN)

«شبکه کلان‌شهری» (Metropolitan Area Network) یک «شبکه رایانه‌ای» بزرگ است که معمولاً در سطح یک شهر گسترده می‌شود. در این شبکه‌ها معمولاً از «زیرساخت بی‌سیم» و یا اتصالات «فیبر نوری» جهت ارتباط محل‌های مختلف استفاده می‌شود.

۱-۱-۱-۴- شبکه گسترده (WAN)

«شبکه گسترده» (Wide Area Network) یک «شبکه رایانه‌ای» است که نسبتاً ناحیه جغرافیایی وسیعی را پوشش می‌دهد (برای نمونه از یک کشور به کشوری دیگر یا از یک قاره به قاره‌ای دیگر).
شبکه‌های گسترده برای اتصال شبکه‌های محلی یا دیگر انواع شبکه به یکدیگر استفاده می‌شوند.

۱-۱-۱-۵- شبکه متصل (Internetwork)

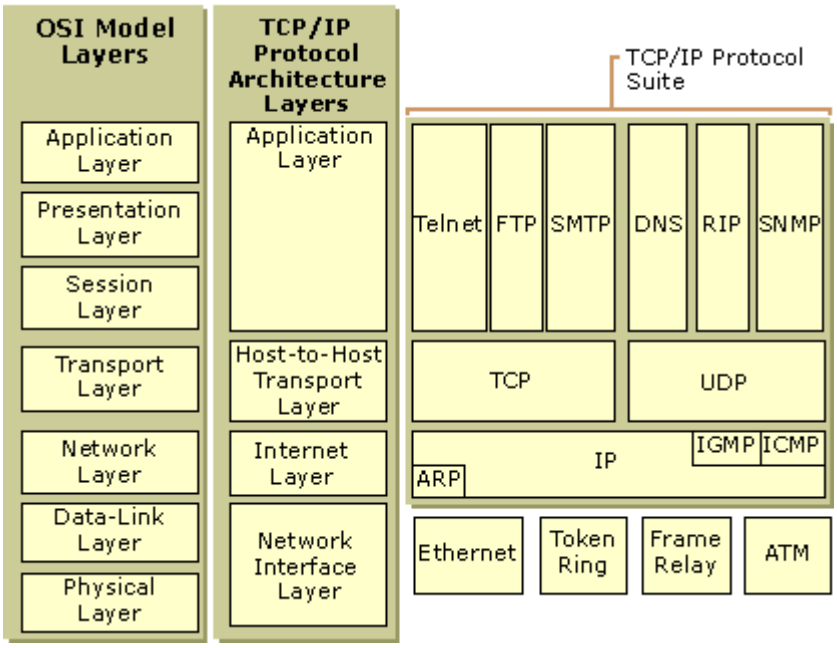
دو یا چند «شبکه» یا «زیرشبکه» (Subnet) که با استفاده از تجهیزاتی که در لایه ۳ یعنی «لایه شبکه» «مدل مرجع OSI» عمل می‌کنند مانند یک «مسیریاب»، به یکدیگر متصل می‌شوند تشکیل یک شبکه از شبکه‌ها یا «شبکه متصل» را می‌دهند.
می‌توان سه نوع «شبکه متصل» دسته بندی نمود:

- شبکه داخلی یا اینترانت (Intranet)
- شبکه خارجی یا اکسترانت (Extranet)
- شبکه‌اینترنت (Internet)

۱-۲- مفاهیم اولیه برنامه‌نویسی تحت شبکه

۱-۲-۱- مدل TCP/IP

مدل TCP/IP یا مدل مرجع اینترنتی که گاهی به مدل DOD (وزارت دفاع)، مدل مرجع ARPANET نامیده می‌شود، یک توصیف خلاصه لایه TCP/IP برای ارتباطات و طراحی پروتکل شبکه کامپیوتراست. (شکل ۱)



شکل ۱ مقایسه مدلهای لایه‌ای شبکه و پروتکلهای آنها

۱-۲-۱-۱- لایه کاربرد[1]

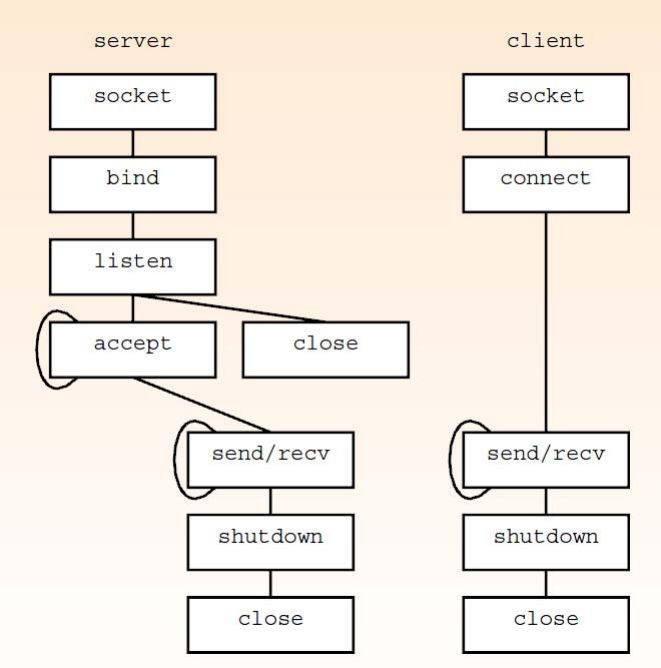
لایه کاربردی بیشتر توسط برنامه‌ها برای ارتباطات شبکه استفاده می‌شود. به سرویس‌های استاندارد شبکه از جمله http، ftp، telnet و روش‌های ارتباط سایر برنامه‌ها اشاره دارد.

۱-۲-۱-۲- لایه انتقال[2]

[1] Application Layer
[2] Transport Layer

مسئولیتهای لایه انتقال، قابلیت انتقال پیام را END-TO-END و مستقل از شبکه، به اضافه کنترل خطا، قطعه قطعه کردن و کنترل جریان را شامل می‌شود. دو نوع برقراری ارتباط در این لایه عبارت اند از: [1]

- اتصال گرا[1] (TCP): نیاز به برقراری اتصال بین سیستم‌ها قبل از تبادل داده‌ها است (روش Handshake). (شکل ۲)



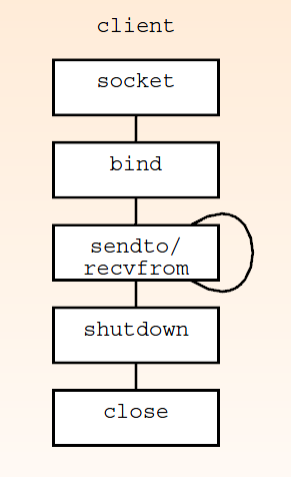
شکل ۲ نمودار روند سوکتهای اتصال‌گرا

- بدون اتصال[2] (UDP): بدون برقراری اتصال می‌توان به تبادل داده‌ها پرداخت. (شکل ۳)

۱-۲-۱-۳- لایه اینترنت[3]

این لایه آدرس‌دهی بسته‌ها و تحویل آن‌ها بین شبکه‌ها را مدیریت می‌کند. همچنین بسته‌ها را به تکه‌های[4] کوچکتر تقسیم کرده تا در لایه پایین‌تر پردازش شوند.

1 Connection-Oriented
2 Connectionless
3 Internet layer
4 Fragment



شکل ۳ نمودار روند سوکتهای بدون اتصال

۱-۲-۱-۴- لایه رابط شبکه[1]

وظیفه تحویل داده‌ها به وسیله رابط فیزیکی را بر عهده دارد. این لایه همچنین بسته‌ها را خطایابی می‌کند.

Layer	Description	Protocols
Application	Defines TCP/IP application protocols and how host programs interface with transport layer services to use the network.	HTTP, Telnet, FTP, TFTP, SNMP, DNS, SMTP, X Windows, other application protocols
Transport	Provides communication session management between host computers. Defines the level of service and status of the connection used when transporting data.	TCP, UDP, RTP
Internet	Packages data into IP datagrams, which contain source and destination address information that is used to forward the datagrams between hosts and across networks. Performs routing of IP datagrams.	IP, ICMP, ARP, RARP
Network interface	Specifies details of how data is physically sent through the network, including how bits are electrically signaled by hardware devices that interface directly with a network medium, such as coaxial cable, optical fiber, or twisted-pair copper wire.	Ethernet, Token Ring, FDDI, X.25, Frame Relay, RS-232, v.35

جدول ۱ پروتکلها و لایه‌های مدل TCP/IP

[1] Network Interface layer

۱-۲-۲- سوکت برکلی

برنامه رابط نرم‌افزاری سوکت‌های برکلی شامل کتابخانه‌ای برای تولید نرم‌افزارها به زبان برنامه‌نویسی C می‌باشد که بیشتر ارتباطات بین شبکه‌ها برای ارتباط میان-پردازه‌ای را انجام می‌دهد.

سوکت برکلی (شناخته شده به عنوان BSD socket API) در BSD 4.2 سیستم عامل Unix (منتشر شده در سال ۱۹۸۳) به عنوان برنامه رابط نرم‌افزاری ساخته شد. از این مجموعه برای کدنویسی برنامه‌هایی که بین دو میزبان در شبکه یا دو پردازه در یک کامپیوتر عملیات ارتباطی دارند استفاده می‌شود. این کاربرد حتی با دستگاه‌های ورودی/خروجی متفاوتی نیز کار می‌کند. پیاده‌سازی از این رابط، همان رابط نرم‌افزاری اصلی مجموعه پروتکل اینترنتی (TCP/IP) است. برای اولین بار در دانشگاه کالیفرنیا، برکلی، برای استفاده در سیستم‌های Unix ساخته شد. هم اکنون تقریبا تمام سیستم عامل‌های مدرن به دنبال استاندارد شدن رابط سوکت برکلی، پیاده‌سازی‌هایی از آن را دارند. [2]

فصل ۲:

برنامه‌نویسی سوکت و مفاهیم آن در سیستم عامل Linux

۲-۱- مقدمه

هدف این پروژه برقراری ارتباط بین کامپیوترهای درون یک شبکه محلی که در واقع منظور همان برنامه‌نویسی تحت شبکه است، می‌باشد. پیاده‌سازی این ارتباط تحت مفاهیم مدل چهار لایه‌ای TCP/IP انجام شده است. این پروژه در سیستم عامل Linux و با زبان برنامه‌نویسی C++ کدنویسی شده است. در ادامه به اهمیت این گزینش‌ها می‌پردازیم. برای برنامه‌نویسی تحت شبکه از مفهومی به نام سوکت ، معروف به سوکت برکلی استفاده می‌شود.

۲-۲- مدل TCP/IP و ارتباط آن با Unix

قدرت Unix در توکار بودن شبکه‌بندی فراهم شده در آن است. در اوایل سال ۱۹۸۰، دانشگاه کالیفرنیا در برکلی، در نسخه اصلی System 7 از Unix، تغییرات اساسی قابل توجهی از جمله پشتیبانی برای حافظه مجازی و نسخه ابتدایی از مدل TCP/IP انجام داد. این نسخه به عنوانBSD 4.2 شناخته شد. در سال ۱۹۸۶ برکلی نسخه جدید Unix را با اصلاح و بهبود در کد شبکه‌بندی TCP/IP به عنوان BSD 4.3 به عرضه گذاشت. [2]

از زمانیکه این نرم‌افزار با امتیاز دولت ایالات متحده امریکا در برکلی ساخته شد، در تمام فروشگاه‌ها و دانشگاه‌ها به کمترین قیمت موجود بود. کد TCP/IP طراحی شده در برکلی به سیستم عامل‌های دیگر از جمله DEC VMS، Macintosh، DOS، Windows، IBM CMS و بسیاری دیگر انتقال یافت. به موجب وجود سیستم‌های کامپیوتری در جاهاییکه TCP/IP قابل استفاده است، این مدل در به هم پیوستن

سیستم‌ها در محیط‌های کامپیوتری غیر یکنواخت یک عامل ضروری محسوب می‌شود. Unix سکوی تولید TCP/IP شد و برکلی شرکت کننده اصلی این در این کوشش بود؛ این کار سیستمی برای شبکه‌بندی تولید کرد که خود را تاکنون اثبات کرده است. تاکنون تخمین زده شده است که بیش از ۵ میلیون سیستم از مجموعه نرم‌افزاری TCP/IP استفاده می‌کنند. Unix به عنوان سکویی آغازین، راهی را برای مجتمع سازی تمام این سیستم‌های متفاوت به عنوان ساختار مفید هموار ساخت.

۲-۳- آشنایی با برخی مفاهیم مورد نیاز

۲-۳-۱- سوکت و انواع آن

شاید شما این جمله را شنیده باشید «در دنیای یونیکس هر چیزی میتواند بصورت یک فایل تلقی و مدل شود». تمام عوامل و انواع ورودی و خروجیها (I/O) میتواند توسط سیستم فایل مدل شود. [3]

«آیا ارتباط دو کامپیوتر روی شبکه و مبادله اطلاعات بین آن دو، ماهیت ورودی/خروجی ندارد؟»

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

الف) در برنامه خود از سیستم عامل بخواهید تا شرایط را برای برقراری یک «ارتباط» با کامپیوتری خاص (با آدرس IP مشخص) و برنامه ای خاص روی آن کامپیوتر (با آدرس پورت مشخص) فراهم کند یا اصطلاحاً سوکتی را بگشاید.

ب) در صورت موفقیت آمیز بودن عمل در مرحله قبل ، به همان صورتی که

درون یک فایل می‌نویسید یا از آن می‌خوانید ، می‌توانید با توابع send() (یا write()) و recv() (یا read()) اقدام به مبادله داده‌ها بنمائید.

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

در این پروژه با دو نوع سوکت سروکار خواهیم داشت:

- سوکتهای نوع استریم که سوکتهای اتصال گرا نامیده میشود.
- سوکتهای نوع دیتاگرام که سوکتهای بدون اتصال نامیده میشود.

روش ارسال برای سوکتهای نوع استریم همان روش TCP است و بنابراین داده‌ها با رعایت ترتیب و مطمئن با نظارت کافی برخطاهای احتمالی مبادله می‌شوند. سوکتهای نوع دیتاگرام نامطمئن است و هیچگونه تضمینی در ترتیب جریان دادهها وجود ندارد.

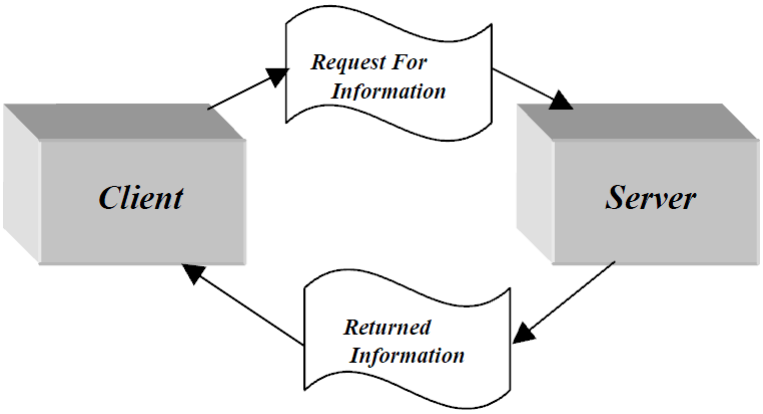
۲-۳-۲- مفهوم سرویس دهنده/مشتری

- **برنامه سرویس دهنده (Server)**

برنامه سرویس دهنده برنامه ای است که بر روی سرور شبکه نصب شده و در خواست هایی را دریافت و پس از پردازش، پاسخ مناسبی به سرویس گیرنده ارسال می کند. در حالت کلی می توان اینگونه نتیجه گرفت که برنامه سمت سرور نمی تواند آغاز کننده ارتباط باشد.

- **برنامه سرویس گیرنده یا مشتری (Client)**

برنامه سرویس گیرنده بر روی Client قرار گرفته و درخواست هایی را به ماشین سرور ارسال می کند و سپس منتظر دریافت پاسخ می ماند. لازم به اشاره است که برنامه سرویس گیرنده را می توان ماشین آغاز کننده ارتباط عنوان کرد زیرا شروع کننده درخواست است.



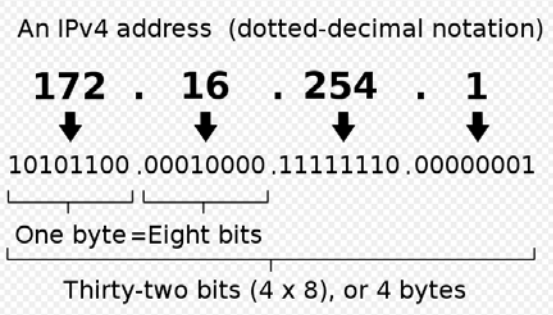
شکل ۴ ارتباط بین سرویس دهنده و مشتری

۲-۴- IP آدرسها، structها و حمل داده‌ها

قبل از اینکه وارد مباحث کدنویسی شویم بگزاریدکمی بیستر در مورد تئوری‌ها بحث کنیم. ابتدا IP آدرس‌ها و دو نوع آن توضیح داده می‌شوند. در ادامه به توابع کتابخانه Socket مورد نیاز برای ذخیره‌سازی و پردازش IPها و داده‌های دیگر صحبت می‌کنیم.

۲-۴-۱- IP آدرسها؛ دو نسخه ۴ و ۶

IP نسخهٔ چهارم یک عدد ۳۲ بیتی است که برای سادگی آن را به شکل چهار بخش عددی در مبنای ده می‌نویسند که با نقطه از هم جدا می‌شوند (مانند 192.168.0.1). این روش نشانی‌دهی را ده‌دهی نقطه‌دار می‌نامند هر یک از چهار بخش را یک هشتایی (Octet) می‌گویند زیرا طول آن ۸ بیت (یا ۱ بایت) است و می‌تواند عددی از ۰ تا ۲۵۵ باشد. پس ۲ به توان ۳۲ آدرس مختلف داریم.



شکل ۵ آدرس IP نسخه ۴

گسترش روز افزون اینترنت و نیاز به آدرس های بسیار بیشتر تیم Internet Engineering Task Force را برآن داشت تا به فکر تکنولوژی های جدیدی باشند تا امکان تعریف آدرس های آی پی بیشتری فرآهم گردد. [4] [5] بهترین راه ساخت مجدد نشانی پروتکل اینترنت بود . در سال ۱۹۹۵ میلادی نسخه جدید نشانی پروتکل اینترنت با نام آی پی نسخه ۶ معرفی گردید . اندازه آدرس از ۳۲ بیت به ۱۲۸ بیت افزایش یافت وامکان آدرس دهی تا ۲به توان ۱۲۸ آدرس افزایش یافت. این کار تنها تعداد آدرس های اینترنتی را گسترش نداد، بلکه باعث خواهد شد جدول مسیریاب‌های اینترنتی (روترها) کوچکتر شود . کلیه سیستم‌عامل‌های جدید سرور و خانگی از جمله ویندوز ویستا به طور کامل پشیبانی می‌شود ولی متاسفانه هنوز توسط بسیاری از مسیریاب های شبکه های خانگی و تجهیزات شبکه عادی پشتیبانی نشده است.

چند مثال از نحوه نمایش IPv6

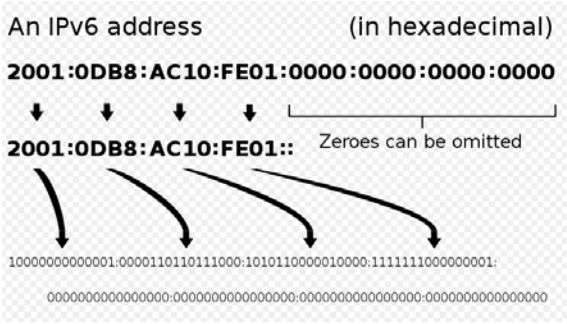
i)	**2001:0db8:c9d2:0012:0000:0000:0000:0051**
	2001:db8:c9d2:12::51
ii)	2001:0db8:ab00:0000:0000:0000:0000:0000
	2001:db8:ab00::
iii)	0000:0000:0000:0000:0000:0000:0000:0001
	::1

جدول ۲ نمونه‌هایی از IP نسخه ۶ و نمایش معادل آنها

نکته جالب آدرس 1:: است که معادل آن در نسخه ۴، همان 127.0.0.1 یا آدرس Loopback می‌باشد.

نکته آخر اینکه یک حالت سازگاری نسخه ۴ در نسخه ۶ وجود دارد. به عنوان مثال

برای نشان دادن آدرس 192.0.2.33 نسخه ۴ به صورت نسخه ۶، حاصل به صورت ::ffff:192.0.2.33 خواهد بود.



شکل ۶ آدرس IP نسخه ۶

۲-۴-۲- کلاسهای مختلف IP و subnetها

سه کلاسِ پایه‌ایِ مختلفِ نشانی‌دهیِ آی‌پی، برای شبکه‌های بزرگ، متوسط و کوچک وجود دارد. کلاس A برای شبکه‌های بزرگ، کلاس B برای شبکه‌های متوسط و کلاس C برای شبکه‌های کوچک است. (شکل ۷) [6]

کلاس	طول بر حسب بیت	شروع	پایان	CIDR	subnet mask
Class A	0	0.0.0.0	127.255.255.255	8/	255.0.0.0
Class B	10	128.0.0.0	191.255.255.255	16/	255.255.0.0
Class C	110	192.0.0.0	223.255.255.255	24/	255.255.255.0
Class D [[multicast]]	1110	224.0.0.0	239.255.255.255	4/	Not Defined
Class E [[reserved]]	1111	240.0.0.0	255.255.255.255	4/	Not Defined

شکل ۷ کلاسهای آدرس IP نسخه ۴

برای جلوگیری از هدردهی IP در هر کلاس، از مفهومی به نام subnet mask استفاده می‌شود. در نسخه ۴ به صورت 192.0.2.12, 255.255.255.252 یا 192.0.2.12/30 و در نسخه ۶ به صورت 2001:db8::/32 یا مثال دیگر مانند 2001:db8:5413:4028::9db9/64 خواهد بود.

۲-۴-۳- شماره پورت[1]

کنار IP آدرس، آدرس دیگری نیز توسط TCP (سوکتهای اتصال‌گرا) و UDP (سوکتهای بدون اتصال) مورد استفاده قرار می‌گیرد که شماره پورت نام دارد. این شماره یک عدد ۱۶ بیتی همانند یک آدرس محلی برای اتصال است. IP آدرس را به عنوان آدرس خیابان هتلی در نظر بگیرید و شماره پورت به عنوان شماره اتاق خواهد بود.

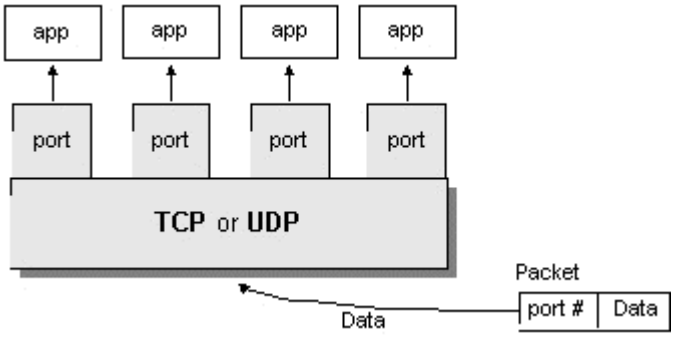
پورتها به دو گروه رزرو شده (پورت های بین ۱ تا ۱۰۲۴) و غیر رزرو شده (سایر پورت ها) تقسیم می شوند. پورت های رزرو شده برای کاربردهای استاندارد مورد استفاده قرار می گیرند. مانند :

PORT NUMBER	USAGE
۲۵	SMTP
۸۰	HTTP
۱۱۰	POP
۱۱۹	News Server

جدول ۳ نمونه‌ای از شماره پورتهای رزرو شده

در این پروژه از پورتهای غیر رزرو شده که آزاد باشند (مورد استفاده سایر برنامه ها نباشند) می توان جهت برقراری ارتباطات مورد نیاز، استفاده نمود. یعنی می تواند به ازای هر پورت با یک برنامه ارتباط برقرار کند (مانند پورت ۳۰۰۰).

[1] Port Number



شکل ۸ نمایشی از اتصال و عملکرد پورت

Byte Order -۴-۴-۲

باید این نکته را بدانید که، دو نوع مدل برای مرتب سازی بایت وجود دارد: [7]

- Big Endian

به عنوان مثال عدد هگزادسیمال b34f را در نظر بگیرید. در حالت عادی شما به ترتیب b3 و بعد 4f را ذخیره می‌کنید. این همان روشی است که در شبکه‌ها استفاده می‌شود و به عنوان Network Byte Order شناخته شده است.

- Little Endian

در این روش تمام بایتها به صورت عکس ذخیره می‌شوند. یعنی در عدد b34f ابتدا 4f و بعد b3 ذخیره می‌شود و این همان روشی نیز هست که در سیستمهای سازگار با اینتل استفاده می‌شود که به روش Host Byte Order معروف است.

بعضی ماشین ها، اعدادشان را ذاتا به صورت ترتیب بایت شبکه ای ذخیره می کنند و بعضی دیگر خیر. هنگامی که ذکر می‌شود که یک چیز باید در یک ترتیب بایت شبکه‌ای باشد، شما مجبور هستید تابعی (از قبیل htons())را برای تغییر آن از ترتیب بایت میزبان ۱۵ فراخوانی کنید.

دو نوع تابع برای تبدیل این دو نوع مدل وجود دارد:

- Short: اعداد ۲ بایتی
- Long: اعداد ۴بایتی

htons()	**h**ost **to** **n**etwork **s**hort
htonl()	**h**ost **to** **n**etwork **l**ong
ntohs()	**n**etwork **to** **h**ost **s**hort
ntohl()	**n**etwork **to** **h**ost **l**ong

جدول ۴ توابع تبدیل بایت شبکه‌ای

در حالت کلی برای ارسال داده‌ها روی رسانه انتقال ابتدا باید آنها را به صورت Network Byte Order تبدیل کنید و برعکس برای گرفتن داده‌ها از رسانه انتقال باید آنها را به صورت Host Byte Order درآوردید.

۲-۴-۵- ساختارها[1]

خوب؛ حالا زمان صحبت در مورد برنامه‌نویسی است. در این فصل توضیحات مختصری برای آشنایی با کاربرد در مورد نوعهای داده مورد استفاده در واسط سوکت داده خواهد شد. در ابتدا باید این نکته را یادآور شوم که هر مشخصه سوکت[2] (همانند اشاره‌گر فایل) یک عدد صحیح از نوع داده int است.

❖ **struct addrinfo**

این ساختمان داده یکی از ابتکارات جدید است. تمام اطلاعات آدرسهای سوکت برای استفاده‌های بعدی در این ساختار ذخیره می‌شوند. همچنین در یافتن نام ماشین میزبان یا نام سرویس نیز مورد استفاده قرار می‌گیرد. اولین قدم برای برقراری یک ارتباط استفاده از همین ساختمان داده است.

1 structs
2 Socket descriptor

```
struct addrinfo {
    int              ai_flags;      // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;     // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;   // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;   // use 0 for "any"
    size_t           ai_addrlen;    // size of ai_addr in bytes
    struct sockaddr *ai_addr;       // struct sockaddr_in or _in6
    Char            *ai_canonname;  // full canonical hostname
    struct addrinfo *ai_next;       // linked list, next node
};
```

این ساختمان داده با استفاده از تابع getaddrinfo() مقداردهی می‌شود. در صورت وجود چند نتیجه، ساختار به واسطه لیست‌های پیوندی (به وسیله ai_next*) دستردسی را میسر می‌کند. برای تحمیل نسخه IPv4 یا IPv6 به مقادیر بازگشتی تابع می‌توان در بخش ai_family از ثابتهای بیان شده استفاده کرد. جالب است که می‌توان برای عدم تحمیل نسخه خاصی می‌توان از ثابت AF_UNSPEC استفاده کرد. برای به دست آوردن اطلاعات اصلی IP آدرس، از اشاره‌گر ai_addr که به ساختمان داده IP آدرس اشاره می‌کند استفاده می‌کنیم.

❖ struct sockaddr

این ساختمان داده اطلاعات مربوط به بیشتر انواع سوکت را در خود نگه می‌دارد:

```
//holds socket address information for many types of sockets
struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

بخش sa_family مربوط به تعیین نسخه IP که منجر به نحوه استفاده از ساختار می‌شود، است. به موازات این ساختمان داده، دو ساختار sockaddr_in و sockaddr_in6 برای cast اشاره‌گر داده‌های آن طبق نسخه IP مورد استفاده قرار می‌گیرند.

```
// IPv4 only
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};

// Internet address
struct in_addr {
    uint32_t s_addr;                // that's a 32-bit int (4 bytes)
};
```

```
// IPv6 only
struct sockaddr_in6 {
    u_int16_t       sin6_family;    // address family, AF_INET6
    u_int16_t       sin6_port;      // port number, Network Byte
Order
    u_int32_t       sin6_flowinfo;  // IPv6 flow information
    struct in6_addr sin6_addr;      // IPv6 address
    u_int32_t       sin6_scope_id;  // Scope ID
};

// Internet address
struct in6_addr {
    unsigned char   s6_addr[16];    // IPv6 address
};
```

❖ struct sockaddr_storage

این ساختمان داده به اندازه کافی بزرگ است و به این منظور طراحی شده که هم IPv4 و هم IPv6 را با هم در خود نگه دارد. در بعضی مواقع نمی‌دانیم که قرار است کدام نسخه از IP را ذخیره کنیم و این همان ساختمان داده‌ای است که مورد استفاده قرار می‌گیرد. بعد از ذخیره اطلاعات، می‌توان آن‌ها را با ساختارهای یاد شده به نوع مورد نظر cast کرد.

```
//another simple structure, struct sockaddr_storage
struct sockaddr_storage {
    sa_family_t  ss_family;             // address family

    // all this is padding, implementation specific, ignore it:
```

```
    char      __ss_pad1[_SS_PAD1SIZE];
    int64_t   __ss_align;
    char      __ss_pad2[_SS_PAD2SIZE];
};
```

برای این کار فقط نیاز به تشخیص نوع داده آن دارید که در فیلد ss_family می‌توانید AF_INET یا AF_INET6 بودن آنرا تشخیص داده و cast مورد نظر خود را انجام دهید.

۲-۴-۶- تبدیلات آدرس شبکه

توابع زیادی برای تبدیل IP ادرس‌ها به Byte Orderهای بیان شده وجود دارد. اکنون به معرفی دو تابع پر کاربرد مورد استفاده در این پروژه می‌پردازم.

- تبدیل IP آدرس رشته‌ای به شبکه‌ای با استفاده از تابع inet_pton()
- تبدیل IP آدرس شبکه‌ای به حالت رشته‌ای با استفاده از تابع inet_ntop()

مثالی از تبدیلات:

```
//convert string IP addresses to their binary representations
struct sockaddr_in sa;          // IPv4
struct sockaddr_in6 sa6;        // IPv6

inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr));                 // IPv4
inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr)); // IPv6
```

```
//convert binary representations to string IP addresses
// IPv4:
char ip4[INET_ADDRSTRLEN];  // space to hold the IPv4 string
struct sockaddr_in sa;      // pretend this is loaded with something

inet_ntop(AF_INET, &(sa.sin_addr), ip4, INET_ADDRSTRLEN);
printf("The IPv4 address is: %s\n", ip4);

// IPv6:
char ip6[INET6_ADDRSTRLEN]; // space to hold the IPv6 string
struct sockaddr_in6 sa6;    // pretend this is loaded with something

inet_ntop(AF_INET6, &(sa6.sin6_addr), ip6, INET6_ADDRSTRLEN);
printf("The address is: %s\n", ip6);
```

۲-۵- توابع مربوط

در این بخش به توضیح فراخوانی توابع مورد نیاز در پروژه می‌پردازیم. کتابخانه‌های مربوط به این توابع نیز بیان می‌شوند.

❖ getaddrinfo()

تابعی دارای گزینه‌های زیاد با کاربرد بسیار ساده است. برای تنظیم کردن و انتساب به ساختمان داده‌هایی مورد استفاده قرار می‌گیرد که بعدا مورد استفاده واقع می‌شوند.

```
//Prepare the structs
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node,    // e.g. "www.example.com" or IP
                const char *service, // e.g. "http" or port number
                const struct addrinfo *hints,
                struct addrinfo **res);
```

خروجی تابع، اشاره‌گری به لیست پیوندی res حاصل از نتایج اجرا است. بخش node، نام ماشین میزبان یا IP آدرس است. پارامتر servise می‌تواند شماره پورت مانند 80 یا نام سرویس مانند http باشد. پارامتر hints ساختاری از نوع addrinfo است که با اطلاعات مورد نظر پر شده است.

مثالی از نحوه استفاده این تابع برای سرور میزبانی که بر روی آدرس ماشین با پورت ۳۰۰۰ تنظیم می‌شود:

```
// only sets up structures we'll use later
int status;
struct addrinfo hints;
struct addrinfo *servinfo;       // will point to the results

memset(&hints, 0, sizeof hints); // make sure the struct is empty
hints.ai_family = AF_UNSPEC;     // don't care IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
hints.ai_flags = AI_PASSIVE;     // fill in my IP for me

if ((status = getaddrinfo(NULL, "3000", &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
    exit(1);
}
```

```
// servinfo now points to a linked list of 1 or more struct addrinfos

// ... do everything until you don't need servinfo anymore ....

freeaddrinfo(servinfo);          // free the linked-list
```

❖ socket()

و بالاخره نقطه آغاز یک برنامه تحت شبکه!

خروجی این تابع در صورت موفقیت آمیز بودن یک مشخصه سوکت (یا به قولی مشخصه فایل) است.

```
// get socket file descriptor
#include <sys/types.h>
#include <sys/socket.h>

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

- domain
  - AF_INET: برای تعیین نسخه ۴ از IP آدرس
  - AF_INET6: برای تعیین نسخه ۶ از IP آدرس
- type
  - SOCK_STREAM: ارتباط اتصال‌گرا
  - SOCK_DGRAM: ارتباط بدون اتصال
- protocol: TCP یا UDP بودن را تعیین می‌کند. معمولا با عدد 0 مقداردهی می‌شود، تا به صورت خودکار نسبت به type تعیین شود.

❖ bind()

در طرف سرور استفاده می‌شود و سوکت را به پورت تعیین شده (مانند 3000) مقید می‌کند.

```
// which port I'll listen
#include <sys/types.h>
#include <sys/socket.h>

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

sockfd، مشخصه سوکت حاصل از تابع socket است. My_addr اشاره‌گری به ساختمان داده sockaddr است که حاوی اطلاعاتی درباره آدرس و شماره پورت می‌باشد. طول این ساختمان داده به باین نیز addrlen است.

❖ connect()

در طرف ماشین مهمان (Guest) برای وصل شدن به سرور از این دستور استفاده می‌شود.

```
// connect to server
#include <sys/types.h>
#include <sys/socket.h>

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

sockfd همان مشخصه سوکت است که در طرف ماشین مهمان حاصل تابع socket() است. آدرس و شماره پورت ماشین مقصد یا میزبان در ساختمان داده serv_addr به طول addrlen بایت ذخیره شده است.

❖ listen()

برای اینکه ماشین مهمان بتواند به میزبان connect() شود، لازم است که ماشین میزبان listen() کند تا بتواند درخواست را اصطلاحا بشنود.

```
// listen for incoming connections
int listen(int sockfd, int backlog);

// sequence of use
getaddrinfo();
socket();
bind();
listen();
/* accept() goes here */
```

سوکتی که می‌خواهیم روی آن listen() انجام دهیم به وسیله مشخصه سوکت sockfd و تعداد درخواستهای connect() که پذیرفته می‌شوند (مثلا 10 درخواست) در backlog تعیین می‌شوند.

❖ accept()

درخواست connect() از سوی ماشین مهمان را می‌پذیرد.

```
// accept incoming connection
#include <sys/types.h>
#include <sys/socket.h>

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

همان مشخصه سوکت sockfd که listen() می‌کند، اکنون درخواست اتصال از سوی ماشین مهمان را می‌پذیرد و اطلاعات آنرا درون اشاره‌گر addr به طول addrlen بایت قرار می‌دهد.

❖ send() و recv()

این دو تابع در ارتباط اتصال‌گرا برای دریافت و ارسال داده‌ها مورد استفاده قرار می‌گیرند.

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

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

char *buf = "Project…";
int len, bytes_sent, bytes_recv;
.
.
len = strlen(buf);
bytes_sent = send(sockfd, msg, len, 0);
.
.
bytes_recv = recv(sockfd, buf, len, 0);
.
.
```

❖ sendto() و recvfrom()

در حالت ارتباط بدون اتصال مورد استفاده قرار می‌گیرند. در این حالت برای ارسال و دریافت داده‌ها، علاوه بر داده، نیاز به ساختمان داده حاوی آدرس ماشین مقصد برای

ارسال داده‌ها (در زمان sendto()) و ساختمان داده برای ذخیره اطلاعات ماشینی که از آن داده دریافت کرده‌ایم (در زمان recvfrom()، است.

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

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

❖ close() و shutdown()

توابعی که در نهایت برای بستن ارتباط نیاز است.

در صورتی که مایل به قطع ارتباط به صورت کامل و دوطرفه هستید از تابع close() استفاده کنید.

```
// Close connection now
close(sockfd)
```

در ضمن برای اینکه کنترل بیشتری بر روی نحوه قطع ارتباط داشته باشید می‌توانید از تابع shutdown() استفاده نمایید.

```
// Control how to close connection
int shutdown(int sockfd, int how)
```

❖ how

- SHUT_RD: قطع دریافت داده
- SHUT_WR: قطع ارسال داده
- SHUT_RDWR: قطع ارسال و دریافت داده

نگاهی اجمالی بود بر توابعی که تقریبا در اکثر برنامه‌های تحت شبکه مورد استفاده قرار می‌گیرند. در فصل پیاده‌سازی در صورت نیاز توضیحات بیشتری در رابطه با آنها

داده خواهد شد.

در ضمن باید خاطر نشان کنم که این توابع فقط مربوط به بحثهای شبکه بوده و در زمینه گرافیک برنامه و چند نخی[1]، کتابخانه‌ها و توابع مربوط به خود را دارند؛ آنها نیز در فصل پیاده‌سازی توضیح داده خواهند شد.

[1] multithreading

فصل ۳:

تعریف مساله و طراحی

۳-۱- مقدمه

در این فصل، نرم‌افزار پروژه را با اصول مهندسی بیان می‌کنم. ابتدا به تعریف نیازها می‌پردازم و در ادامه نمودارها و... توضیح داده خواهند شد.

۳-۲- نیازها

۳-۲-۱- نیازهای کاربر

طراحی و پیاده‌سازه کلاس ارتباطات شبکه در سیستم عامل Linux با زبان C++ به طوریکه هر شیء نمونه‌سازی شده از این کلاس باید دارای قابلیتهای زیر باشد:

- ❖ تعیین پروتکل در لایه انتقال (TCP/UDP)
- ❖ امکان کار به صورت Blocking و یا NonBlocking
- ❖ اعمال مکانیزمهای حفاظت
- ❖ امکان Log شدن تراکنشهای ارتباطی جهت دیباگ

۳-۲-۲- نیازهای سیستم

- ❖ استفاده از پوسته گرافیکی برای کاربر پسند بودن سیستم
- ❖ لزوم استفاده از مفاهیم چند نخی[1]
- ❖ نیاز به تعیین رمز عبور برای امنیت سیستم
- ❖ استفاده از کتابخانه‌های استاندارد برنامه‌نویسی

[1] Multi-threading

- ❖ تعیین خصوصیات سیستم توسط کاربر در نحوه اجرای نرم‌افزار
  - نوع پروتکل
  - IP آدرس
  - شماره پورت و ...
- ❖ توانایی به تعلیق انداختن سیستم
- ❖ توانایی به کار انداختن دوباره سیستم
- ❖ توانایی تعیین خصوصیات سیستم در حین اجرا
- ❖ دریافت اطلاعات کاربری از کاربران و ذخیره آنها
- ❖ دریافت و ارسال پیام به صورت پخشی به تمام کاربران
- ❖ اعمال فیلتر در ارسال پخشی پیام به کاربران
- ❖ نمایش پیامها برای کاربران
- ❖ ثبت و نمایش لیست کاربران حاضر
- ❖ ذخیره تمام اطلاعات و تراکنشها در یک فایل متنی

۳-۲-۳- مدلهای سیستم

- اجرای سیستم در حالت Server
  - دریافت و ثبت ورود کاربر جدید
  - درخواست برای Authentication کاربر
  - ثبت و نمایش تراکنشهای کاربران
  - ثبت و نمایش نام کاربران
  - ارسال نام کاربران به دیگر کاربرها
  - توانایی قطع ارتباط کاربر
  - اعلان قطع ارتباط کاربر

- اجرای سیستم در حالت Client
  - ارسال اطلاعات کاربری از جمله نام و رمز عبور
  - امنیت در ارسال رمز عبور
  - ارسال پیام به کاربران با اعمال فیلترینگ

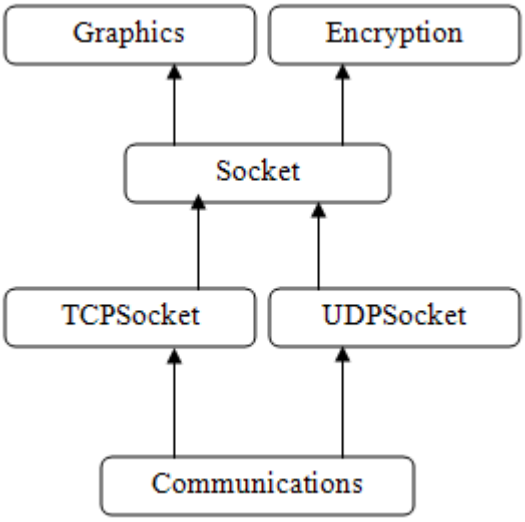
۳-۲-۴- تکامل سیستم

- ❖ تغییر پوسته گرافیکی سیستم به دلیل وجود باگ در سیستم عامل Linux
- ❖ برداشتن محدودیت سیستم از سقف ۱۰ کاربر به تعداد نامحدود
- ❖ امکان ارسال فایل
- ❖ امکان تعیین مکانیزمهای حفاظت در سطح کاربر
- ❖ افزودن اطلاعات کاربری بیشتر
- ❖ امکان برقراری ارتباط به صورت صوتی و تصویری

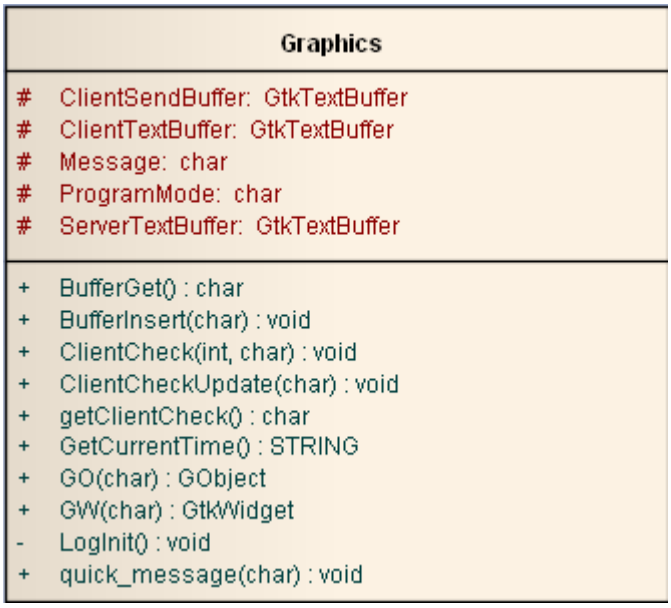
۳-۳- نمودار کلاس

در این قسمت نحوه ارثبری کلاسها (شکل ۹) نمایش داده خواهد شد. تمام ارثبریها به صورت Public و Virtual انجام می‌شوند.

در نهایت از کلاس Communication، شیء گرفته می‌شود.



شکل ۹ نحوه ارثبری کلاسها



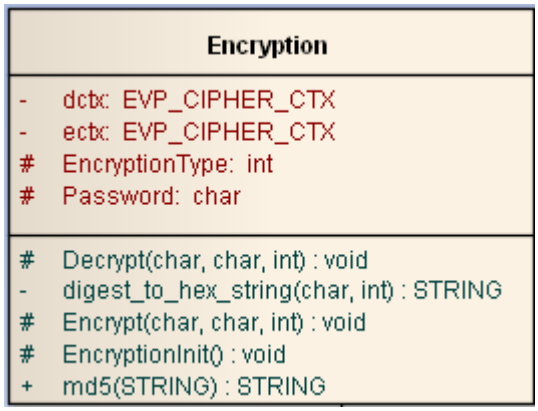
شکل ۱۰ خصوصیات و متدهای کلاس Graphics

❖ کلاس Graphics

لایه گرافیکی نرم‌افزار را تشکیل می‌دهد که دارای خواص و متدهای زیر است:

❖ کلاس Encryption

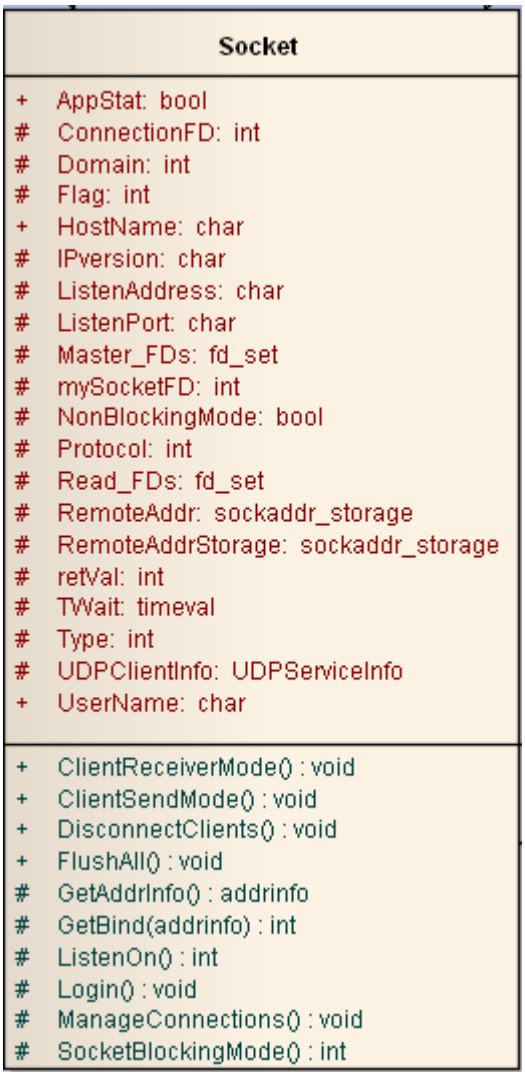
پیاده‌سازی توابع برای امنیت داده‌ها در این کلاس صورت گرفته است.



شکل ۱۱ خصوصیات و متدهای کلاس **Encryption**

❖ کلاس Socket

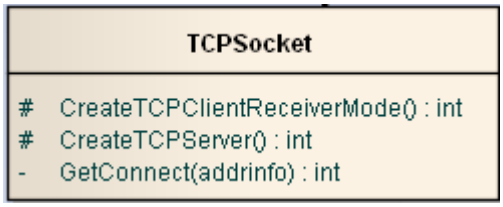
Socket پایه‌ای‌ترین کلاس به شمار می‌رود. تمام توابع اساسی در آن پیاده‌سازی شده‌اند. از دو کلاس Graphics و Encryption مشتق می‌شود. این کلاسها به دلیل ارثبری سلسله مراتبی و جلوگیری از کپی شدن دوباره خصوصیات و متدها به صورت Virtual به ارث برده شده‌اند. در شکل ۱۲به دلیل عدم تکرار متدها و خواص از ذکر آنها خودداری شده است؛ اما برای یادآوری باید بیان کنم که تمام خواص و متدهای Public و یا protected به ارث برده می‌شوند.



شکل ۱۲ خصوصیات و متدهای کلاس Socket

❖ کلاس TCPSocket

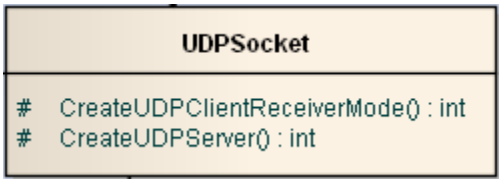
از کلاس Socket به صورت Virtual مشتق شده است. برای ساختن ارتباط اتصال‌گرا، کلاس مشتق شده از این کلاس به توابع به ارث برده شده‌اش ارجاع می‌کند.



شکل ۱۳ خصوصیات و متدهای کلاس TCPSocket

❖ کلاس UPDSocket

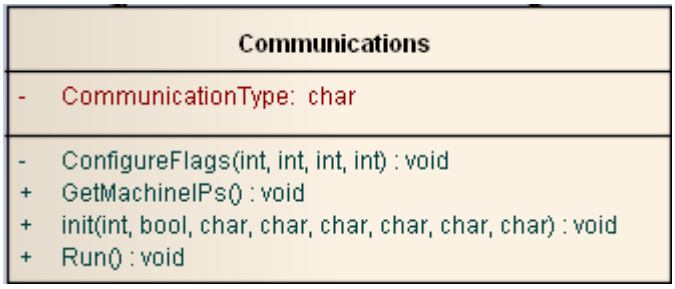
از کلاس Socket به صورت Virtual مشتق شده است. برای ساختن ارتباط بدون اتصال، کلاس مشتق شده از این کلاس به توابع به ارث برده شده‌اش ارجاع می‌کند.



شکل ۱۴ خصوصیات و متدهای کلاس UDPSocket

❖ کلاس Communications

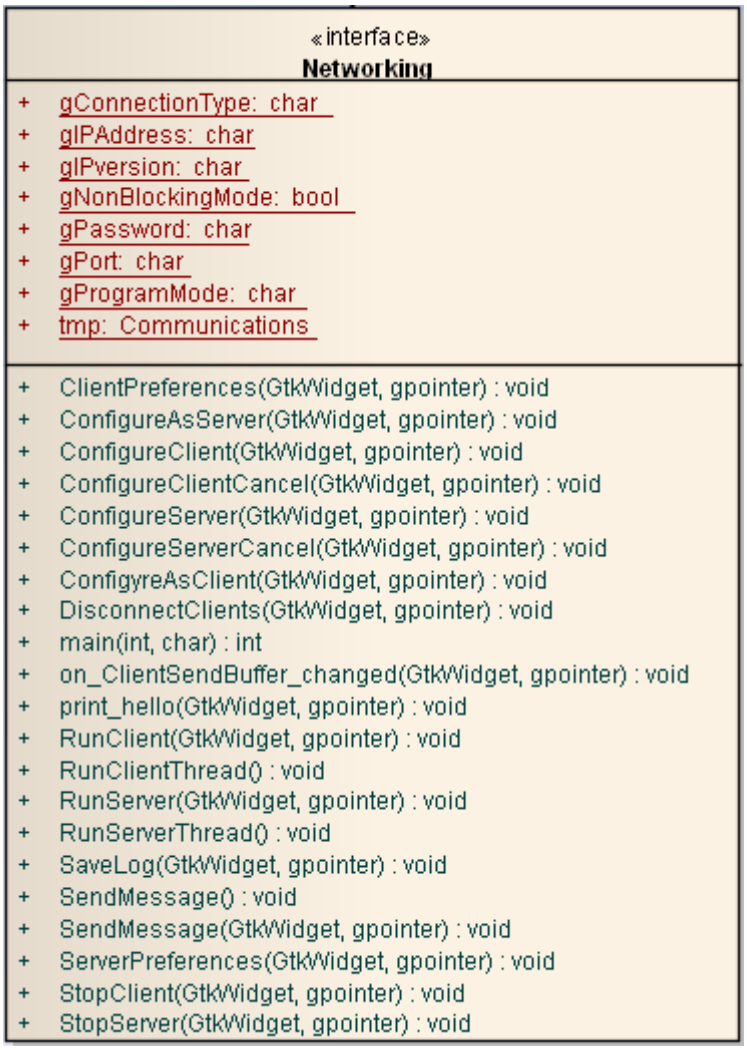
از دو کلاس TCPSocket و UDPSocket به صورت Virtual مشتق می‌شود. بالاترین لایه به حساب می‌آید. تمام اطلاعات برای تعیین حالت برنامه، خصوصیات ارتباط، نحوه برقراری آن و ... از این کلاس به لایه‌های پایینتر ارجاع داده می‌شوند.



شکل ۱۵ خصوصیات و متدهای کلاس Communication

❖ Networking.cpp

فایل اصلی که تابع main در آن قرار دارد و شیء کلاس Communication نیز در آن ساخته می‌شود. سیگنالهای گرافیکی نیز در این فایل تعریف و پیاده‌سازی شده‌اند.



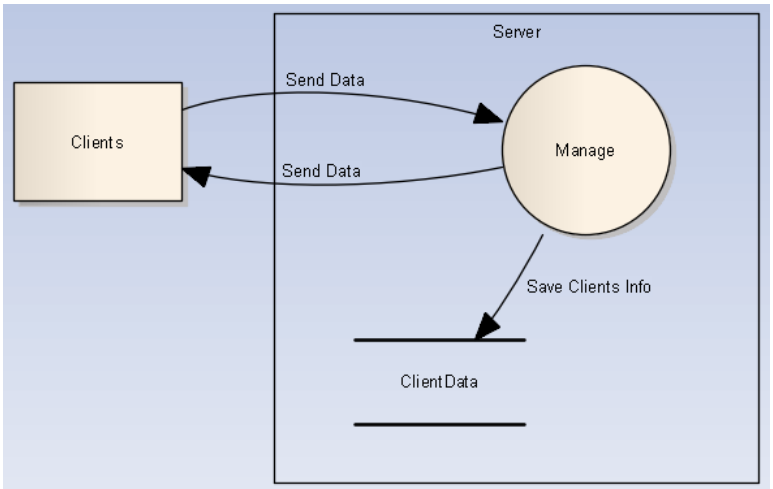
شکل ۱۶ متغیرها و توابع Networking

۳-۴- نمودارهای جریان داده[1]

۳-۴-۱- حالت Server

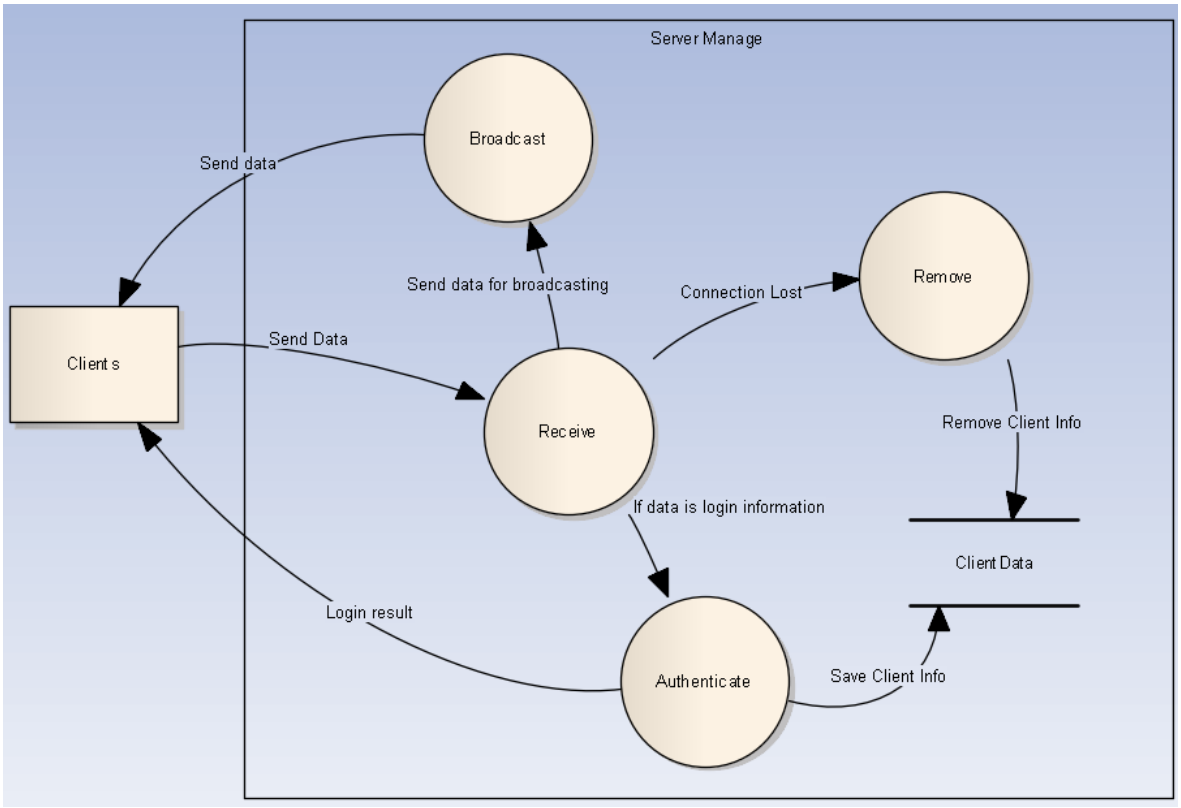
❖ نمودار سطح صفر

[1] Data Flow Diagram (DFD)



شکل ۱۷ نمودار سطح صفر جریان داده حالت Server

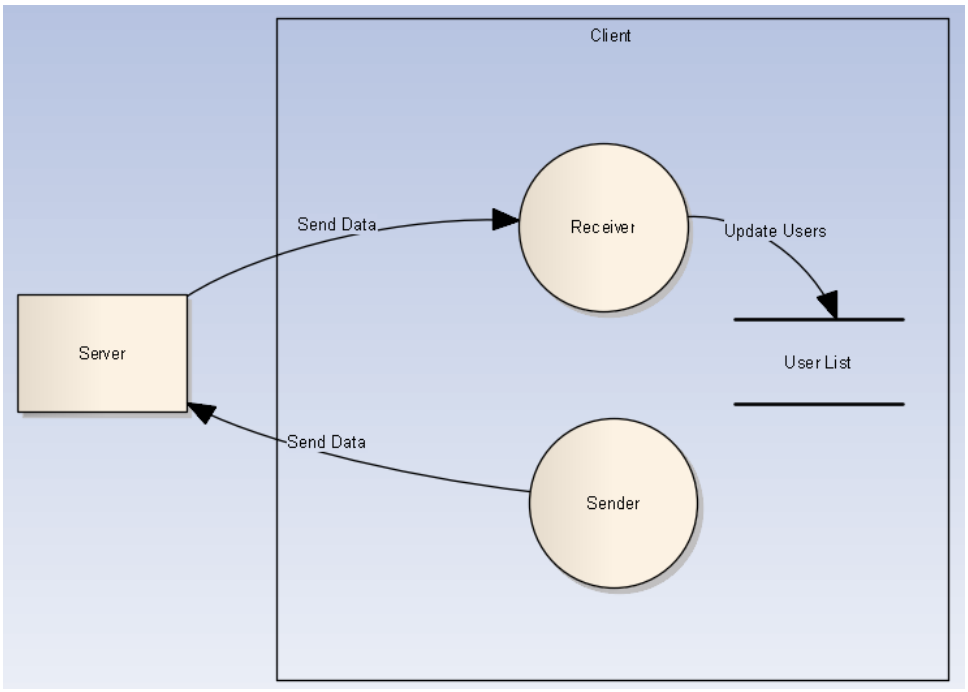
❖ نمودار سطح یک



شکل ۱۸ نمودار سطح یک جریان داده حالت Server

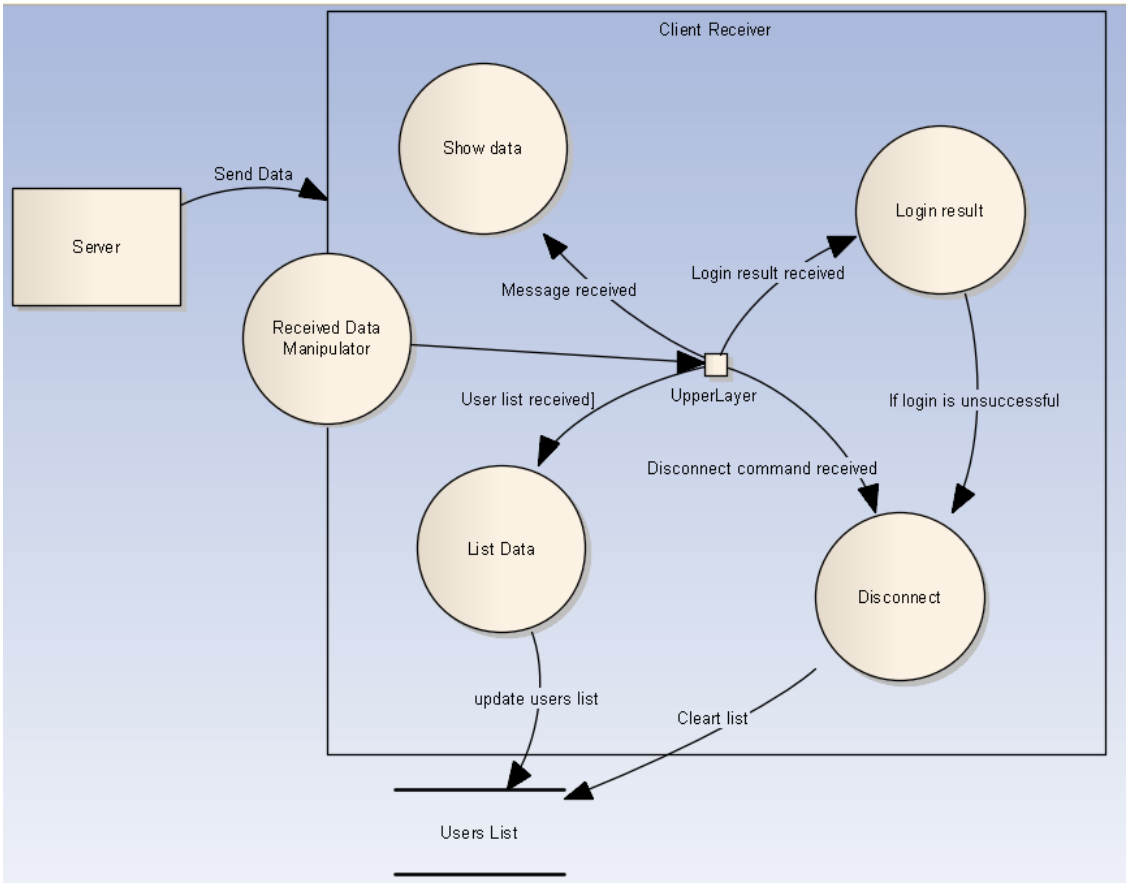
۳-۴-۲- در حالت Client

❖ نمودار سطح صفر



شکل ۱۹ نمودار سطح صفر جریان داده حالت Client

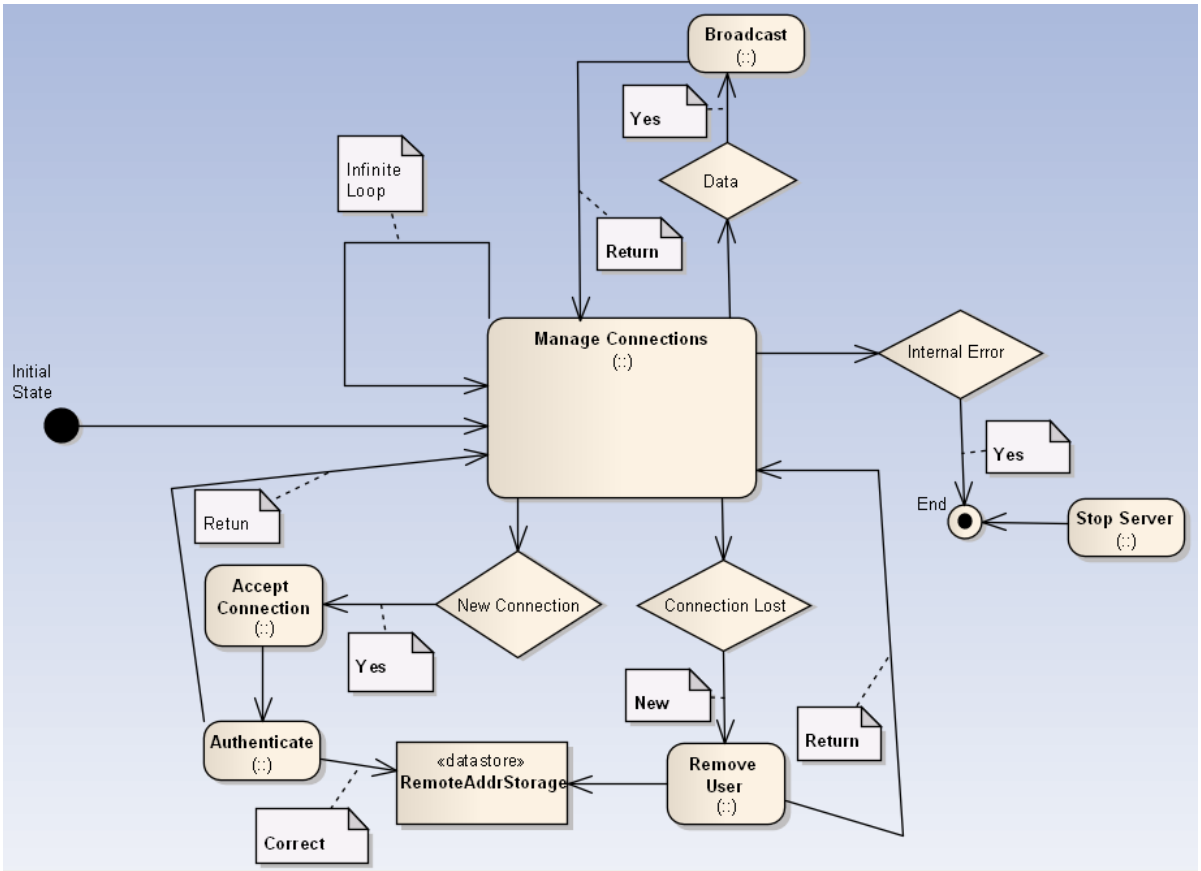
❖ نمودار سطح یک



شکل ۲۰ نمودار سطح صفر جریان داده حالت Client

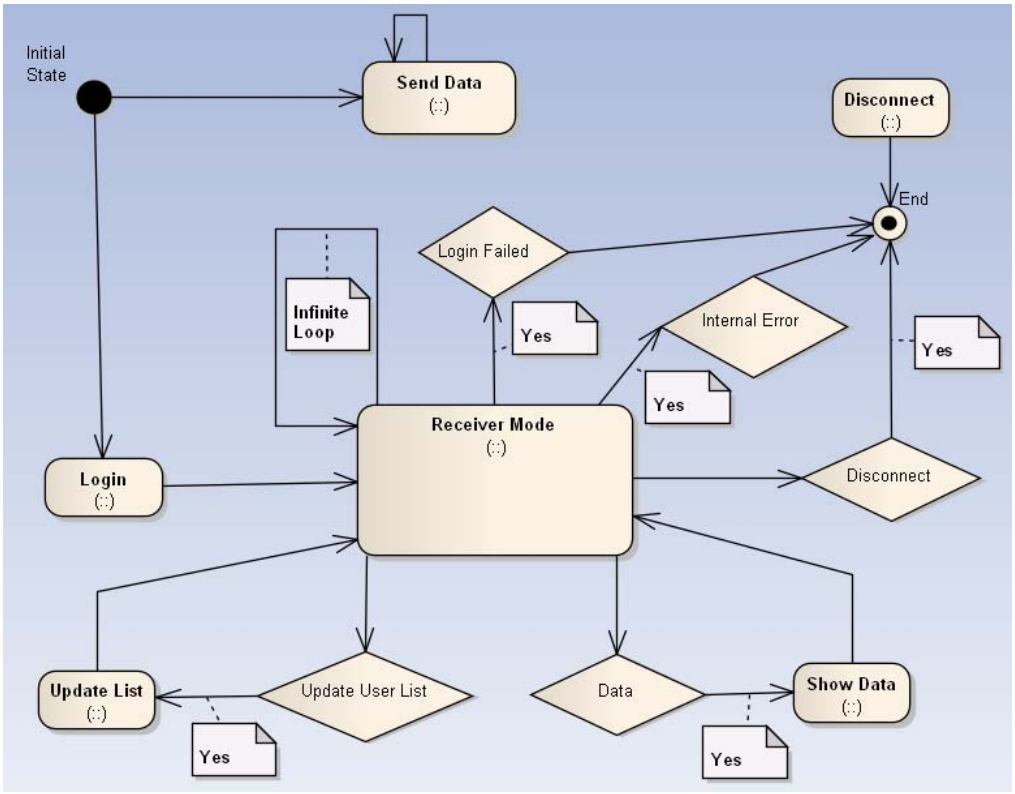
۳-۵- نمودار فعالیت

۳-۵-۱- در حالت Server



شکل ۲۱ نمودار فعالیت حالت Server

۳-۵-۲- در حالت Client



شکل ۲۲ نمودار فعالیت حالت Client

«نمودارها در محیط نرم‌افزار Enterprise Architect 8 رسم شده‌اند.»

فصل ۴:

پیاده‌سازی

۴-۱- مقدمه

در این فصل پیاده‌سازی کد نوشته شده را شرح خواهم داد. قبل از آن نحوه آماده‌سازی و نصب نرم‌افزارهای استفاده شده در سیستم عامل Linux که مورد برای شروع پروژه مورد نیاز بوده‌اند را توضیح خواهم داد.

۴-۲- شروع کار

۴-۲-۱- سیستم عامل

ابتدا سیستم عامل Linux نسخه ubuntu 10.10 را نصب می‌کنیم. برای سهولت کار می‌توان نصب آنرا بر روی ماشین مجازی[1]، مانند VMware Workstation انجام داد (کاری که من کردم).

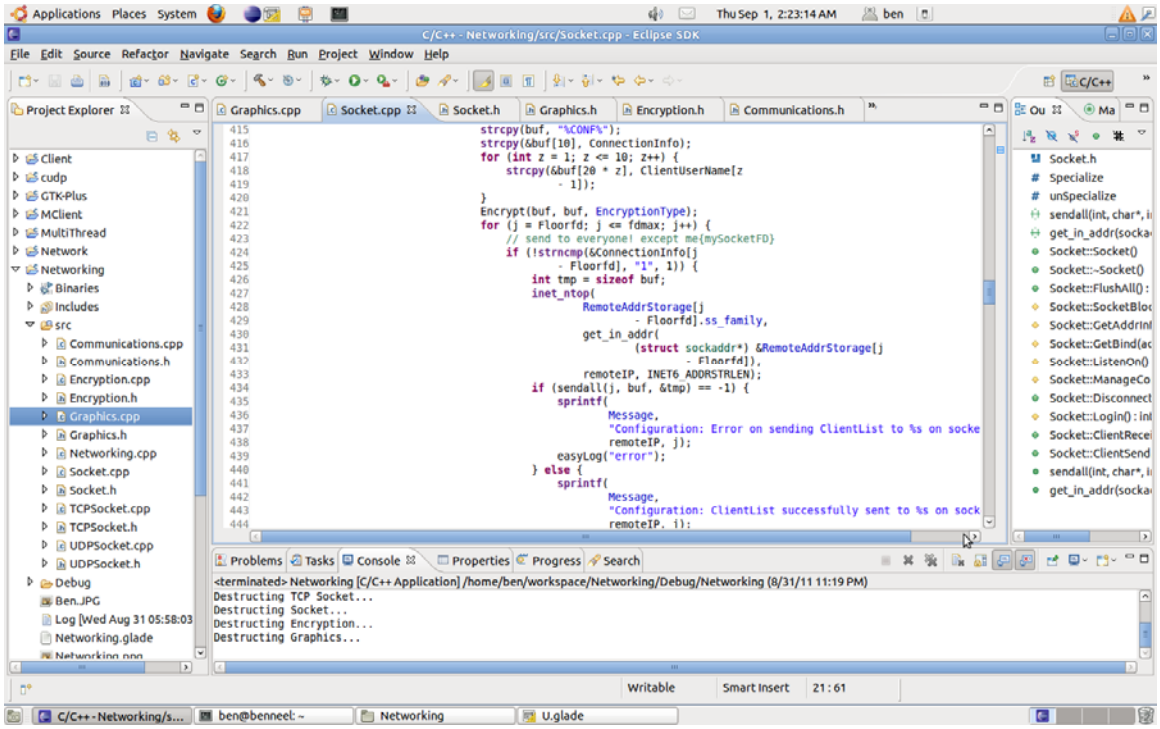
۴-۲-۲- محیط برنامه‌نویسی

کدنویسی باید با زبان برنامه‌نویسی C++ انجام شود، محیط Eclipse پیشنهاد می‌شود. برای نصب آن از منوی اصلی روی Applications و گزینه Ubuntu Software Center را کلیک کنید. وارد IDEs از بخش Developer Tools شوید. Eclipse را انتخاب و آنرا نصب کنید. این محیط در حالت پیش‌فرض فقط از Java پشتیبانی می‌کند. برای اضافه کردن C++ باید مراحل زیر را انجام دهید:

- از منوی Help گزینه Install New Software را کلیک کنید.

[1] Virtual Machine

- در قسمت Work with لینک زیر را کپی کنید و اینتر کنید.
  - http://download.eclipse.org/tools/cdt/releases/galileo
- با ظاهر شدن دو گزینه CDT، هر دو را expand کنید.
- حتما و فقط گزینه‌های زیر را انتخاب کنید:
  - Eclipse C/C++ Development Tools
  - CDT GNU Toolchain Build Support
  - CDT GNU Toolchain Debug Support
  - Eclipse C/C++ Development Platform
- گزینه Next را کلیک کرده و مراحل را تا کامل شدن دانلود و نصب ادامه دهید.



شکل ۲۳ محیط برنامه‌نویسی Eclipse

✓ **برای کامپایل کردن برنامه باید پارامترهای زیر را به کامپایلر و لینکر Eclipce اضافه کنید:**

- GCC C++ Compiler

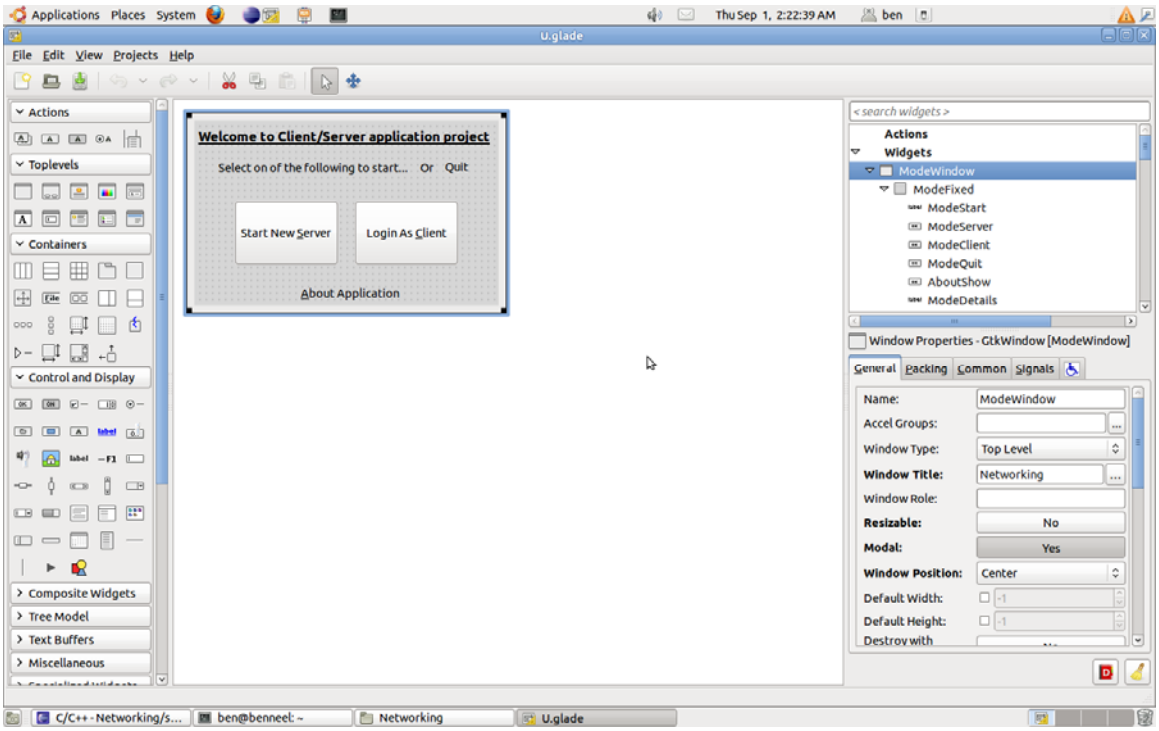
```
g++ -lcrypto `pkg-config --cflags --libs gtk+-2.0 gmodule-2.0`
```

- GCC C++ Linker

```
g++ -lcrypto `pkg-config --cflags --libs gtk+-2.0 gmodule-2.0`
```

۴-۲-۳- گرافیک نرم‌افزار؛ GTK+

در حالت عادی تمام کتابخانه‌های GTK+ بر روی سیستم نصب هستند و نیازی به نصب کتابخانه دیگری نیست. برای طراحی گرافیکی، نرم‌افزار Glade Interface Designer پیشنهاد می‌شود. برای نصب از منوی Applications گزینه Ubuntu Software Center را کلیک کنید. در پنجره باز شده از قسمت Developer Tools وارد Graphical Interface Design شده و مراحل نصب Glade Interface Designer را طی کنید.



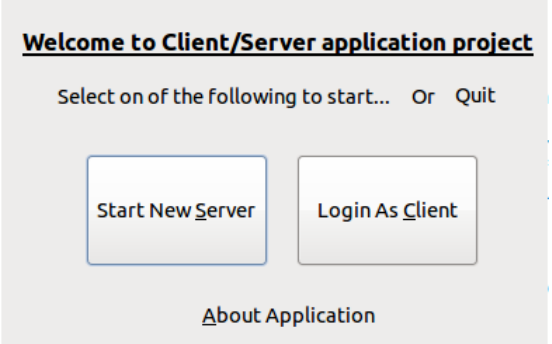
شکل ۲۴ طراحی در محیط Glade

۴-۳- طراحی محیط نرم‌افزار

با استفاده از Glade که از کتابخانه‌های GTK بحره می‌برد، طراحی نرم‌افزار را انجام می‌دهیم. در آخر فایل حاوی دستورات با پسوند glade. و حاوی محتوای xml بیانگر گرافیک نرم‌افزار به شرح زیر می‌باشد:

❖ پنجره ModeWindow

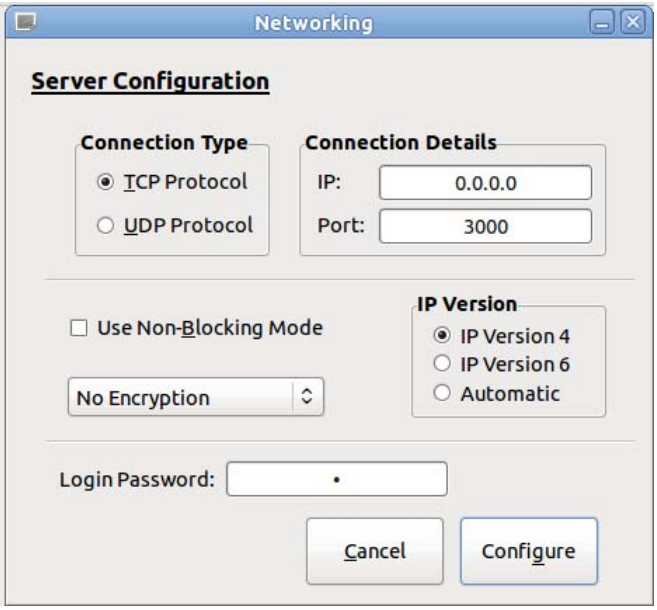
در این قسمت انتخاب می‌کنید که نرم‌افزار به صورت میطبان عمل کند یا به صورت کاربر به میزبان وصل شود. گزینه‌ای نیز برای نمایش اطلاعاتی در مورد نرم‌افزار گنجانده شده است.



شکل ۲۵ پنجره آغاز برنامه

❖ پنجره TypeServerWindow

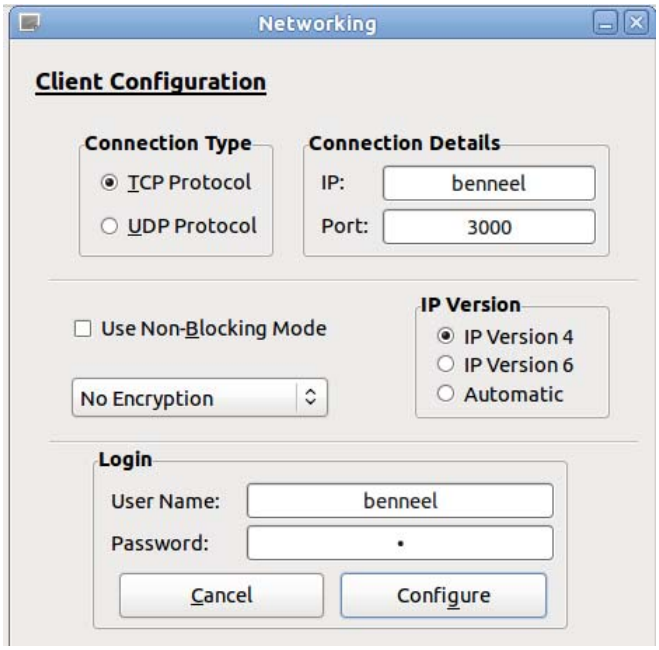
با انتخاب حالت میزبان وارد قسمت تنظیمات آن می‌شوید. نوع ارتباط (اتصال‌گرا یا بدون اتصال)، آدرس و پورتی که میزبان به آن مقید می‌شود، نسخه آدرس، حالت بدون بلوک شدن روی دستورات اصلی شبکه (مانند send() و recv() و ...)، الگوریتم حفاظت داده‌ها و رمز عبور باید تعیین شوند. بعد از تکمیل اطلاعات با زدن دکمه Configure وارد پنجره اصلی میزبان می‌شود.



شکل ۲۶ پنجره تنظیمات Server

❖ پنجره TypeClientWindow

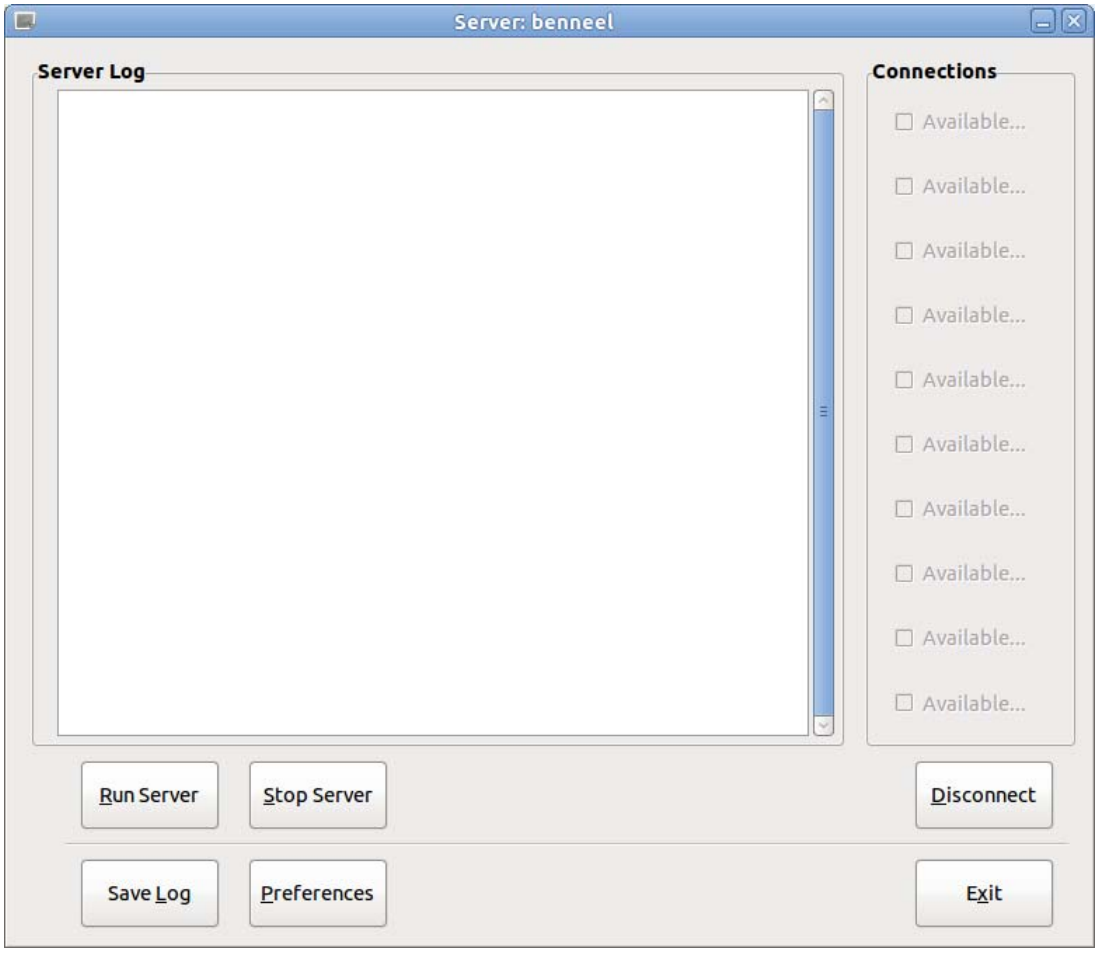
❖ با انتخاب حالت کاربر وارد قسمت تنظیمات آن می‌شوید. همانند تنظیمات میزبان، نوع ارتباط (اتصال‌گرا یا بدون اتصال)، آدرس و پورتی که میزبان به آن مقید می‌شود، نسخه آدرس، حالت بدون بلوک شدن روی دستورات اصلی شبکه (مانند send() و recv() و ...)، الگوریتم حفاظت داده‌ها، نام کاربری و رمز عبور باید تعیین شوند. بعد از تکمیل اطلاعات با زدن دکمه Configure وارد پنجره اصلی کاربر می‌شود.



شکل ۲۷ پنجره تنظیمات Client

❖ پنجره ServerWindow

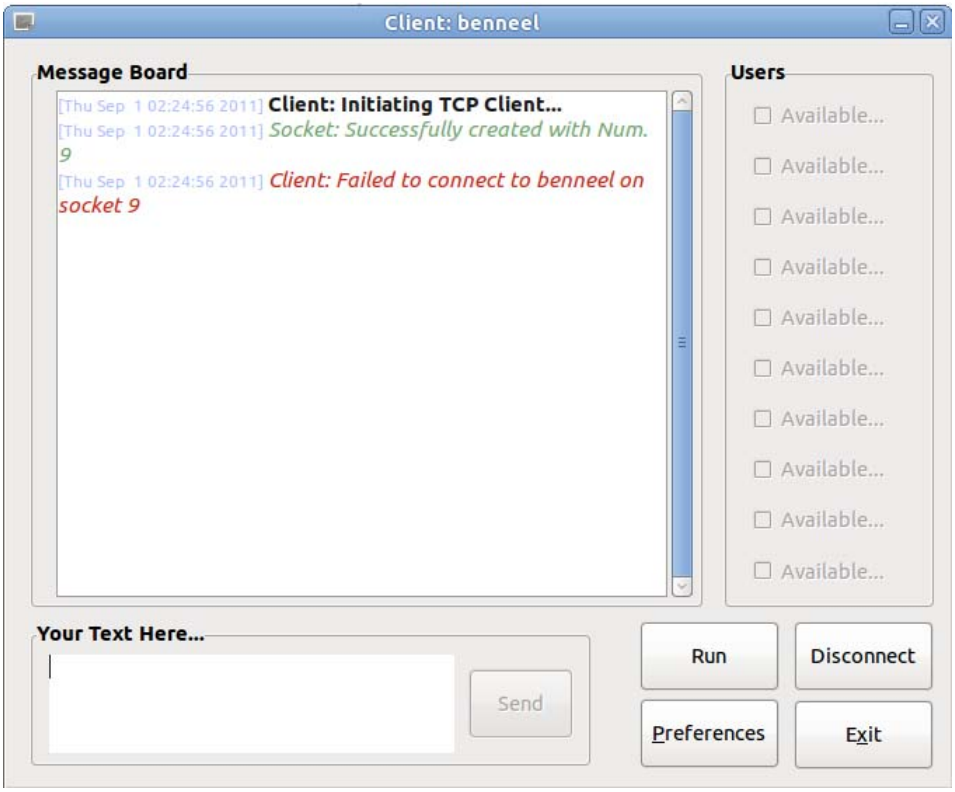
مدیریت کاربران در این قسمت انجام می‌شود. با ورود به این بخش و با زدن دکمه Run، میزبان اجرا شده و ارتباطات را مدیریت می‌کند. گزینه Preferences برای تعیین تنظیمات مورد استفاده قرار می‌گیرد. دکمه Stop، کیزبان را متوقف کرده و تمام ارتباطات را قطع می‌کند. برای قطع ارتباط خاصی، کاربر مورد نظر را انتخاب و Disconnect کنید.گزینه Save Log کاری انجام نمی‌دهد؛ فقط برای نمایش پیغامی برای کاربر مبنی بر ذخیره شدن اطلاعات تراکنش نشان می‌دهد. برای خروج از برنامه نیز می‌توانید از دکمه Exit استفاده کنید.



شکل ۲۸ پنجره Server

❖ پنجره ClientWindow

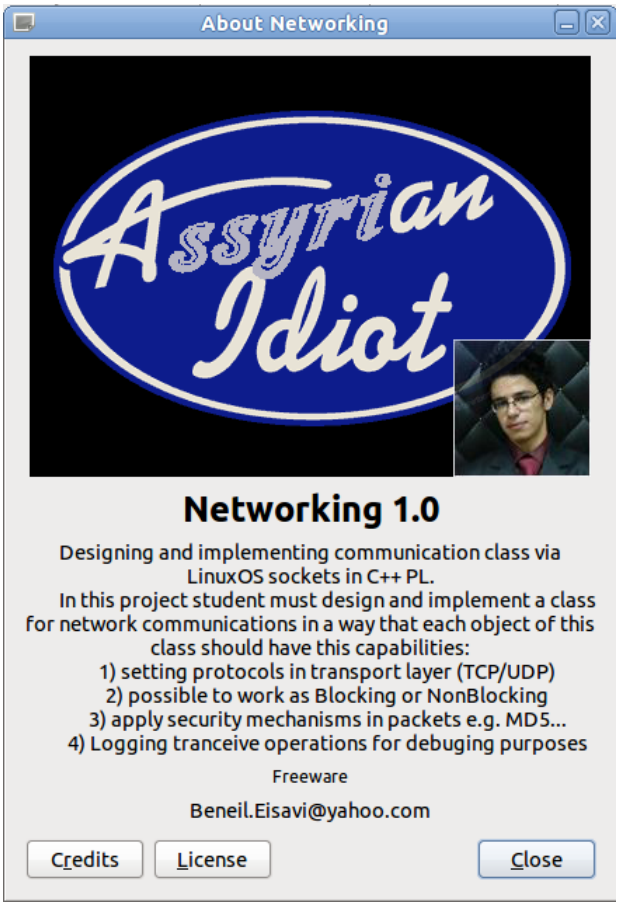
برای برقراری ارتباط از دکمه Run استفاده کنید. اگر موفق به اتصال نشدید، تنظیمات برنامه را با استفاده از دکمه Preferences تغییر دهید. در کادر مشخص شده متن خود را تایپ کرده و دکمه Send را کلیک کنید. برای قطع ارتباط می‌توانید از دکمه Disconnect استفاده کنید. دکمه Exit برنامه را خاتمه می‌دهد.



شکل ۲۹ پنجره Client

❖ پنجره About

اطلاعاتی مختصر در مورد نرم‌افزار و من ;) نمایش می‌دهد.



شکل ۳۰ پنجره درباره نرم‌افزار

۴-۴- کدنویسی

۴-۴-۱- پیامهای کنترلی

پیامهای بیان شده در زیر، در برنامه مقداردهی شده و ارسال می‌شوند. در هنگام دریافت نیز بدین صورت که بیان شده‌اند مورد بررسی قرار گرفته و عملیات مربوط اجرا می‌شود.

۴-۴-۱-۱- به روز رسانی لیست کاربران

%CONF%			ConnectionInfo			UserName1			…	UserName10		
0	…	9	10		19	20		39		200		219

جدول ۵ نمایش حافظه متغیر به هنگام ذخیره داده‌های به روز رسانی کاربران

۴-۴-۱-۲- ارسال پیغام

ConnectionInfo			UserName			SendText		
0		9	10		29	30		…

جدول ۶ نمایش حافظه متغیر به هنگام ذخیره پیام برای ارسال

۴-۴-۱-۳- Login

UserName			Password		
0		19	20		34

جدول ۷ نمایش حافظه متغیر به هنگام ذخیره نام کاربری و رمز عبور برای Login

۴-۴-۱-۴- فرمان Disconnect

MD5(“%DISCONNECT%”)		
0		11

جدول ۸ نمایش حافظه متغیر به هنگام ذخیره فرمان قطع ارتباط

۴-۴-۱-۵- پیام Login ناموفق

MD5(“%LOGIN_FAILED%”)		
0		13

جدول ۹ نمایش حافظه متغیر به هنگام ذخیره اعلام Login ناموفق

Graphics.h –۲–۴–۴

Graphics Class Header

```
1)  /*
2)   * Graphics.h
3)   *
4)   *  Created on: Aug 13, 2011
5)   *      Author: ben
6)   */
7)
8)  #ifndef GRAPHICS_H_
9)  #define GRAPHICS_H_
10)
11) #include <iostream>          فراخوانی کتابخانه‌های جریان ورودی و خروجی
12) using namespace std;          تعیین فضای نامی
13)
14) #include <stdio.h>            فراخوانی کتابخانه دارای توابع مربوط به صفحه کلید و مونیتور
15) #include <stdlib.h>
16)                               فراخوانی کتابخانه استاندارد برای استفاده از توابع ساده مانند atoi() و exit()
17) #include <string.h>           فراخوانی کتابخانه ساختمان داده رشته‌ای و توابع آن
18) #include <sstream>           فراخوانی کتابخانه جریان رشته‌ای
19)
20) #include <time.h>             فراخوانی کتابخانه زمان
21)
22) #include <errno.h>            فراخوانی کتابخانه برای استفاده از توابع خطایابی و خروجی‌های دارای خطا
23)
24) #include <gtk-2.0/gtk/gtk.h>  فراخوانی کتابخانه گرافیکی
25) #include <pthread.h>          فراخوانی کتابخانه پشتیبانی از چند نخی
26)
27)                               این ماکرو برای به دست آوردن اندیس ابتدای buffer مربوط به GtkTextView استفاده می‌شود.
28) #define SIter(buffer,Iter) gtk_text_buffer_get_start_iter(buffer, &Iter);
29)
30)                               این ماکرو برای به دست آوردن اندیس ابتدای buffer مربوط به GtkTextView استفاده می‌شود.
31) #define EIter(buffer,Iter) gtk_text_buffer_get_end_iter(buffer, &Iter);
32)
33)                               از این ماکرو برای نمایش پیغام بر روی کنسول، نمایش بر روی GtkTextView با tag بیان شده و ذخیره
                                  تراکنش در فایل Log استفاده می‌شود.
34) #define easyLog(tag)\
35)     cout<<Message;\
36)     BufferInsert(tag);\
37)     fprintf(LogFD,"%s%s", GetCurrentTime().c_str(),Message);
38)
39) class Graphics {
40) private:
41) void LogInit();
```

Graphics Class Header

```
42)  فایلی برای Log تراکنشها باز کرده و برای نوشتن آماده می‌کند.
43)
44)  protected:
45)  char ProgramMode[10];
46)  حالت برنامه را در خود نگه می‌دارد: Server یا Client
47)  char *Message;
48)  پیغام برای نمایش را در این متغیر نگه می‌داریم که در تابع BufferInsert() مورد استفاده قرار می‌گیرد
49)  GtkTextBuffer *ServerTextBuffer, *ClientTextBuffer, *ClientSendBuffer;
50)  این بافرها به GtkTextViewهای متناظر خود طراحی شده در نرم‌افزار Glade مرتبط می‌شوند. با استفاده از این
     بافرها می‌توان در GtkTextViewها نوشت و یا متنی را گرفت.
51)  public:
52)      FILE *LogFD;          اشاره‌گر به فایل ثبت تراکنشها
53)      GtkBuilder* Builder;
54)  اشاره‌گری که از فایل Networking.glade گرافیک برنامه را می‌سازد و برای استفاده‌های بعدی در دسترس قرار
     می‌دهد.
55)
56)      GObject* GO(const char* objectname);
57)  به دست آوردن GtkObject با نام خاص
58)      GtkWidget* GW(const char* widgetname);
59)  به دست آوردن GtkWidget با نام خاص
60)      void BufferInsert(const char *tag);
61)  افزودن پیغام به GtkTextView با tag تعیین شده
62)      char* BufferGet();
63)  به دست آوردن متن از ClientSendText که برای ارسال مورد استفاده قرار می‌گیرد.
64)
65)      void quick_message(const char *message);
66)  نمایش پیغام با محتوای تعیین شده (در برنامه استفاده نشده است؛ به دلیل مفید بودن قید کردم)
67)      void ClientCheckUpdate(const char buf[256]);
68)  به روز رسانی CheckBoxها که هر کدام بیانگر کاربری هستند.
69)      void ClientCheck(const int num, const char name[20]);
70)  انتخاب کردن یا عدم انتخاب CheckBox خاص با نام مشخص شده
71)      char* getClientCheck();
72)  خروجی رشته به طول ۱۰ کاراکتر با مقادیر '0' یا '1' است که بیانگر انتخاب یا عدم انتخاب CheckBoxهاست.
73)
74)      string GetCurrentTime();      تابعی برای به دست آورن زمان حال به فرم خاص
75)
76)      Graphics();          سازنده کلاس
77)      virtual ~Graphics();      مخرب کلاس
78)  };
79)
80)  #endif /* GRAPHICS_H_ */
```

Graphics.cpp -۳-۴-۴

Graphics Class

```
1)  /*
2)   * Graphics.cpp
3)   *
4)   *  Created on: Aug 13, 2011
5)   *      Author: ben
6)   */
7)
8)  #include "Graphics.h"          شامل شدن محتوای سر فایل کلاس
9)
10) Graphics::Graphics() {
11)     // TODO Auto-generated constructor stub
12)     cout << "Constructing Graphics...\n";      پیغام سازنده
13)     Message = new char();                  انتصاب فضا برای متغیر پیغام
14)     LogFD = new FILE();                    انتصاب فضا برای اشاره‌گر فایل
15)     LogInit();                        فراخوانی متد سازنده فایل ثبت تراکنش
16) }
17)
18) Graphics::~Graphics() {
19)     // TODO Auto-generated destructor stub
20)     cout << "Destructing Graphics...\n";      پیغام مخرب
21) }
22)
23) GObject* Graphics::GO(const char* objectname) {
24)     return (gtk_builder_get_object(Builder, objectname));
25) }
26) با استفاده از این متد، از Builder که تمام ساختارهای گرافیکی در آن ذخیره شده‌اند، بر حسب نام داده شده یک
    کنترل، اشاره‌گری از نوع GObject به آن کنترل cast می‌کند.
27) GtkWidget* Graphics::GW(const char* widgetname) {
28)     return GTK_WIDGET(gtk_builder_get_object(Builder, widgetname));
29) }
30) با استفاده از این متد، از Builder که تمام ساختارهای گرافیکی در آن ذخیره شده‌اند، بر حسب نام داده شده یک
    کنترل، اشاره‌گری از نوع GtkWidget به آن کنترل cast می‌کند.
31)
32)
33)                    این متد متنی را که در متغیر Message ذخیره شده است را با tag مشخص شده اضافه می‌کند.
34) void Graphics::BufferInsert(const char *tag) {
35)     if (!strcmp(ProgramMode, "Server")) {      در صورتی که برنامه در حالت میزبان اجرا می‌شود
36)                                        (به توضیحات بخش Client مراجعه کنید)
37)         GtkTextIter EndIterMain;
38)         GtkTextMark *mark;
39)         ServerTextBuffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(GO(
40)                 "ServerTextView")));
41)         EIter(ServerTextBuffer,EndIterMain);
```

Graphics Class

```
42)         gtk_text_buffer_insert_with_tags_by_name(ServerTextBuffer,
43)                 &EndIterMain, GetCurrentTime().c_str(), -1, "time", NULL);
44)         gtk_text_buffer_insert_with_tags_by_name(ServerTextBuffer,
45)                 &EndIterMain, Message, -1, tag, NULL);
46)         mark = gtk_text_buffer_create_mark(ServerTextBuffer, "end",
47)                 &EndIterMain, FALSE);
48)         gtk_text_view_scroll_to_mark(GTK_TEXT_VIEW(GO("ServerTextView")),
    mark,
49)                 0.05, TRUE, 0.0, 1.0);
50)         gtk_widget_queue_draw(GW("ServerTextView"));
51)     } else if (!strcmp(ProgramMode, "Client")) {      در صورتی که برنامه در حالت کاربر اجرا
    می‌شود
52)                                   تعریف متغیری که به انتهای متن موجود در GtkTextView اشاره خواهد کرد
53)         GtkTextIter EndIterMain;
54)                                   برای قرار دادن علامت در GtkTextView مورد نظر استفاده می‌شود.
55)         GtkTextMark *mark;
56)                                   ابتدا برای شروع کار بافری از ClientTextView می‌گیریم تا در ادامه از آن برای اضافه کردن متن استفاده کنیم،
                                      به cast انجام شده در زیر دقت کنید:
57)         ClientTextBuffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(GO(
58)                 "ClientTextView")));
59)                                   به دست آوردن انتهای متن موجود در بافر ClientTextBuffer که دقیقا به متن موجود در
                                      ClientTextView اشاره می‌کند؛ و قرار دادن آن در متغیر EndIterMain:
60)         EIter(ClientTextBuffer,EndIterMain);
61)                                   اضافه کردن متن به بافر مورد نظر؛ دستور زیر در زمان را به بافر (در حقیقت به جعبه متن) اضافه می‌کند.
62)                                   متن از متد GetCurrentTime().c_str() کرفته می‌شود. tag استفاده شده time است. این tag باعث
                                      می‌شود که متن با ویژگیهای تعریف شده در این tag به جعبه متن اضافه شود. اضافه شدن متن از اندیس تعیین
                                      شده در EndIterMain آغاز می‌شود.
63)         gtk_text_buffer_insert_with_tags_by_name(ClientTextBuffer,
64)                 &EndIterMain, GetCurrentTime().c_str(), -1, "time", NULL);
65)                                   این بخش کد متن مورد نظر از لایه بالاتر را با tag تعیین شده اضافه می‌کند.
66)         gtk_text_buffer_insert_with_tags_by_name(ClientTextBuffer,
67)                 &EndIterMain, Message, -1, tag, NULL);
68)                                   این دستورات موجب scroll شدن جعبه متن به انتهای متن می‌شود.
69)         mark = gtk_text_buffer_create_mark(ClientTextBuffer, "end",
70)                 &EndIterMain, FALSE);
71)         gtk_text_view_scroll_to_mark(GTK_TEXT_VIEW(GO("ClientTextView")),
    mark,
72)                 0.05, TRUE, 0.0, 1.0);
73)                                   جعبه متن را به صف GtkMain اضافه می‌کند، تا تغییرات مورد نظر در آن اعمال شوند.
74)         gtk_widget_queue_draw(GW("ClientTextView"));
75)     }
76) }
77)
78)                                   فقط در مورد برنامه در حالت Client کاربرد دارد. زیرا متن وارد شده از طرف کاربر در جعبه متن
                                      ClientSendText را گرفته و برمی‌گرداند.
```

Graphics Class

```
79) char * Graphics::BufferGet() {
80)     این دو Iter برای به دست آوردن ابتدا و انتهای متن موجود در جعبه متن با واسطه بافر مورد استفاده قرار خواهند گرفت.
81)     GtkTextIter StartIterSend, EndIterSend;
82)     char sendtxt[225] = "";          متن گرفته شده از جعبه متن در این متغیر ذخیره خواهد شد.
83)     مقداردهی بافر ClientSendBuffer به جعبه متن ClientSnedText که حاوی متن وارد شده از طریق کاربر است.
84)     ClientSendBuffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(GO(
85)             "ClientSendText")));
86)     به دست آوردن اندیسهای ابتدا و انتهای متن داخل جعبه به واسطه بافر آن
87)     SIter(ClientSendBuffer,StartIterSend);
88)     EIter(ClientSendBuffer,EndIterSend);
89)     کپی کردن متن به متغیر گفته شده به طول ۱۸۰ کاراکتر اول
90)     strncpy(sendtxt, (char*) gtk_text_iter_get_text(&StartIterSend,
91)             &EndIterSend), 180);
92)     gtk_text_buffer_set_text(ClientSendBuffer, "", 0);      پاک کردن متن جعبه
93)     پردازش تمام به تغییرات انجام شده (بعضی مواقع منجر به رخداد Bug سیستمی می‌شود) [9]
94)     gdk_window_process_all_updates();
95)     usleep(100000);          منتظر ماندن به مدت یک دهم ثانیه
96)     return sendtxt;          برگرداندن متن به عنوان خروجی متد
97)
98) }
99)
100) //Show Message*****************************************************************
101) //Function to open a dialog box displaying the message provided.
102) void Graphics::quick_message(const char *message) {
103)     GtkWidget *dialog, *label, *content_area;
104)     /* Create the widgets */
105)     dialog = gtk_dialog_new_with_buttons("Message", GTK_WINDOW(GO(
106)             "ClientWindow")), GTK_DIALOG_DESTROY_WITH_PARENT,
GTK_STOCK_OK,
107)             GTK_RESPONSE_NONE, NULL);
108)     content_area = gtk_dialog_get_content_area(GTK_DIALOG(dialog));
109)     label = gtk_label_new((gchar*) message);
110)     /* Ensure that the dialog box is destroyed when the user responds. */
111)     g_signal_connect_swapped(dialog, "response",
112)             G_CALLBACK(gtk_widget_destroy), dialog);
113)     /* Add the label, and show everything we've added to the dialog. */
114)     gtk_container_add(GTK_CONTAINER(content_area), label);
115)     gtk_widget_show_all(dialog);
116) }
117)
118) هدف این متد به روز رسانی GtkCheckButtonهاست. آرگومان این متد (buf[256]) تمام اطلاعات مورد نظر برای این کار را باید داشته باشد. هر CheckButton بیانگر یک User است. کاراکترهای ۱۰ تا ۱۹ با مقدار '0' یا '1' بیانگر انتخاب کردن یا نکردن CheckButton و به ازای اندیس این کاراکتر نام User از آرگومان استخراج شده و به عنوان Label برای CheckButton مقداردهی می‌شود. متد فقط در حالت Client برنامه
```

Graphics Class

استفاده شده است.

```
119) void Graphics::ClientCheckUpdate(const char buf[256]) {
120)     int i = 0;
121)     while (i <= 9) {          حلقه، کاراکترهای ۱۰ تا ۱۹ را تست می‌کند
122) برای به دست آوردن نام GtkCheckButton، از جریان رشته‌ای بهره گرفته شده است. به این دلیل که به سادگی می‌توان اعداد را به رشته‌ای از کاراکترها چسباند و خروجی را به نوع مورد نظر در آورد.
123)         if (!strncmp(&buf[i + 10], "1", 1)) {     برقراری این شرط یعنی انتخاب کردن
124)             stringstream x;              تعریف جریان رشته‌ای
125)             char tmp[20];                تعریف متغیر برای نگهداری نام کاربر
126) دستور زیر نام کاربر را بر حسب اندیس کاراکتر به دست می‌آورد، طول نام کاربر ۲۰ کاراکتر است. ۱۰ کاراکتر اول رشته حاوی متن کنترلی، ۱۰ کاراکتر دوم حاوی اندیسهای متناظر با GtkCheckButtonها و از اندیس ۲۰ نام Userها به تعداد ۱۰تا و با طول ۲۰ قارا گرفته شده‌اند.
127)             strncpy(tmp, &buf[20 * (i + 1)], 20);
128)             x << "ClientC";
129)             x << (i + 1);          نامها از ۱ تا ۱۰ هستند
130)             x << "\0";          کاراکتر آخر رشته اضافه می‌شود
131) سه دستور زیر به ترتیب GtkCheckButton با نام x را فعال کرده، در حالت انتخاب قرار داده و برچسب آن را تعیین می‌کند.
132)             gtk_widget_set_sensitive(GW(x.str().c_str()), true);
133)             GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active = true;
134)             gtk_button_set_label(GTK_BUTTON(GW(x.str().c_str())), (gchar*) tmp);
135)         } else if (!strncmp(&buf[i + 10], "0", 1)) {     برقراری این شرط یعنی پاک کردن انتخاب
136)             stringstream x;
137)             x << "ClientC";
138)             x << (i + 1);
139)             x << "\0";
140)             GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active = false;
141)             gtk_button_set_label(GTK_BUTTON(GW(x.str().c_str())),
142)                     "Available...");
143)             gtk_widget_set_sensitive(GW(x.str().c_str()), false);
144)         }
145)         i++;              اندیس، محدوده از ۰ تا ۹
146)         usleep(1000);          انتظار به مدت ۱۰۰۰ میکرو ثانیه
147)     }
148) }
149)
150) این متد دقیقا همانند متد توضیح شده در بالا عمل می‌کند با این تفاوت که فقط برای یک GtkCheckButton خاص تعیین شده در num عمل می‌کند. در صورتی که نام عدد مثبت باشد آنرا فعال و با متغیر name برچسب می‌دهد، در صورت منفی بودن GtkCheckButton تعیین شده را از انتخاب در می‌آورد.
151) void Graphics::ClientCheck(const int num, const char name[20]) {
152)     if (num > 0) {
153)         stringstream x;
154)         x << "ServerC";
155)         x << num;
```

Graphics Class

```
156)            x << "\0";
157)            gtk_widget_set_sensitive(GW(x.str().c_str()), true);
158)            GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active = true;
159)            gtk_button_set_label(GTK_BUTTON(GW(x.str().c_str())), (gchar*) name);
160)        } else if (num < 0) {
161)            stringstream x;
162)            x << "ServerC";
163)            x << abs(num);
164)            x << "\0";
165)            GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active = false;
166)            gtk_button_set_label(GTK_BUTTON(GW(x.str().c_str())), "Available...");
167)            gtk_widget_set_sensitive(GW(x.str().c_str()), false);
168)        }
169)    }
170)
```

171) از این متد به منظور تعیین اینکه آیا GtkCheckButtonها انتخاب شده‌اند یا نه مورد استفاده قرار می‌گیرد. خروجی آن یک رشته ۱۰ کاراکتری با مقادیر ’0‘ یا ’1‘ است. هر یک از مقادیر متناظر با یک GtkCheckButton است.

```
172)    char* Graphics::getClientCheck() {
173)        int i = 0;
174)        char check[10];
175)        if (!strcmp(ProgramMode, "Server")) {
176)            while (i <= 9) {
177)                stringstream x;
178)                x << "ServerC";
179)                x << (i + 1);
180)                x << "\0";
181)                if (gtk_widget_get_sensitive(GW(x.str().c_str()))) {
182)                    if (GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active) {
183)                        strcpy(&check[i], "1");
184)                    }
185)                } else {
186)                    strcpy(&check[i], "0");
187)                }
188)                i++;
189)                usleep(10000);
190)            }
191)        } else {
192)            while (i <= 9) {
193)                stringstream x;
194)                x << "ClientC";
195)                x << (i + 1);
196)                x << "\0";
197)                if (gtk_widget_get_sensitive(GW(x.str().c_str()))) {
198)                    if (GTK_TOGGLE_BUTTON(GW(x.str().c_str()))->active) {
199)                        strcpy(&check[i], "1");
200)                    }
201)                } else {
202)                    strcpy(&check[i], "0");
```

Graphics Class

```
203)                }
204)                i++;
205)                usleep(10000);
206)            }
207)        }
208)        return check;
209)    }
210)
211)                خروجی این متد رشته‌ای کاراکتری حاوی زمان به فرم ' [Thu Sep 1 02:24:56 2011]' است
212)    string Graphics::GetCurrentTime() {
213)        time_t CurTime = time(0);          زمان فعلی را برمی‌گرداند
214)        tm *localtm = localtime(&CurTime);     زمان را بر اساس وقت محلی سیستم تبدیل می‌کند
215)        char t[50] = "", tmp[50] = "";
216)        strcpy(tmp, asctime(localtm));
217)        t[0] = '[';
218)        int z = 0;
219)        while (tmp[z] != '\n') {
220)            t[z + 1] = tmp[z];
221)            z++;
222)        }
223)        t[z + 1] = ']';
224)        t[z + 2] = ' ';
225)        t[z + 3] = '\0';
226)        return t;
227)    }
228)
229)    از این متد برای باز کردن فایل Log ثبت تراکنشها استفاده می‌شود. نام فایل با استفاده از جریان رشته‌ای ساخته
        می‌شود که شامل رشته Log و زمان کنونی سیستم است.
230)    void Graphics::LogInit() {
231)        string t;
232)        system("mkdir –p Log");          ساخت پوشه ثبت تراکنشها
233)        t.assign("./Log/Log ");
234)        t.append(GetCurrentTime());
235)        t[strlen(t.c_str()) - 1] = 0;          کاراکتر آخر رشته را بر روی کاراکتر فاصله می‌نویسد
236)        LogFD = fopen(t.c_str(), "w");          فایل را برای فقط نوشتن باز می‌کند
237)    }
```

Encryption.h –۴–۴–۴

Encryption Class Header

```
1)  /*
2)   * Encryption.h
```

Encryption Class Header

```
3)  *
4)  * Created on: Aug 19, 2011
5)  *     Author: ben
6)  */
7)
8)  #ifndef ENCRYPTION_H_
9)  #define ENCRYPTION_H_
10)
11) #include <iostream>
12) using namespace std;
13)
14) #include <stdio.h>
15) #include <stdlib.h>
16) #include <string.h>
17)
18) #include "openssl/evp.h"          کتابخانه مورد نیاز برای اعمال الگوریتمهای امنیتی
19)
20) class Encryption {
21) private:
22)     این دو متغیر تنظیمات نحوی اعمال الگوریتمهای امنیتی را در خود نگه می‌داردن تا در توابع init مربوطه فراخوانی
        شوند. متغیر ectx برای encryption و متغیر dctx برای decryption مورد استفاده قرار می‌گیرند.
23)     EVP_CIPHER_CTX ectx, dctx;
24)
25)     این تابع رشته حاوی کاراکترهای هگزادسیمال حاصل از encryption را به صورت رشته‌های کاراکتری تبدیل و
        برمی‌گرداند
26)     string digest_to_hex_string(const unsigned char *digest, int len);
27)
28) protected:
29)     int EncryptionType;     عدد صحیح بیانگر نوع حفاظت
30)     char Password[15];      رمز عبور از ابتدای برنامه در این متغیر مقداردهی می‌شود
31)
32)     void EncryptionInit();
33)     void Encrypt(const char plaintext[256], char ciphertext[256], int Type);
34)     void Decrypt(const char ciphertext[256], char plaintext[256], int Type);
35)
36) public:
37)     string md5(string text);
38)
39)     Encryption();
40)     virtual ~Encryption();
41) };
42)
43) #endif /* ENCRYPTION_H_ */
```

Encryption.cpp –۵–۴–۴

Encryption Class

```
1)  /*
2)   * Encryption.cpp
3)   *
4)   *  Created on: Aug 19, 2011
5)   *      Author: ben
6)   */
7)
8)  #include "Encryption.h"
9)
10) Encryption::Encryption() {
11)     // TODO Auto-generated constructor stub
12)     cout << "Constructing Encryption...\n";
13)
14) }
15)
16) Encryption::~Encryption() {
17)     // TODO Auto-generated destructor stub
18)     cout << "Destructing Encryption...\n";
19) }
20) /*****************************************************************************
21)  public utils
22)  *****************************************************************************/
23) string Encryption::digest_to_hex_string(const unsigned char *digest, int len) {
24)     char* hexstring = (char*) malloc(len * 2 + 1);
25)     int i;
26)     len = len > 510 ? 510 : len;
27)     for (i = 0; i < len; i++) {
28)         sprintf(&hexstring[2 * i], "%02x", digest[i]);
29)     }
30)     string hex(hexstring);
31)     free(hexstring);
32)     return hex;
33) }
34) /*****************************************************************************
35)  md5
36)  *****************************************************************************/
37) الگوریتم md5 یک Hash یک طرفه است؛ یعنی با encrypt رشته مورد نظر، متن حاصل را به رشته اولیه
    نمی‌توان تبدیل کرد. بنابراین از این الگوریتم برای امنیت رمز عبور استفاده می‌شود که در زمان Login کاربران
    ابتدا رمز عبورmd5 شده و سپس ارسال می‌شود. هر رشته خاص فقط یک حالت md5 دارد. پس به راحتی در
    طرف دیگر می‌توان روی رشته حاصل که رمزگزاری شده مقایسه انجام داد. حاصل این متد رشته digest شده با
    الگوریتم md5 است.
38) string Encryption::md5(string text) {
39)     EVP_MD_CTX mdctx;
40)     unsigned char md_value[EVP_MAX_MD_SIZE];
41)     unsigned int md_len;
```

Encryption Class

```
42)     EVP_DigestInit(&mdctx, EVP_md5());
43)     EVP_DigestUpdate(&mdctx, text.c_str(), text.size());
44)     EVP_DigestFinal_ex(&mdctx, md_value, &md_len);
45)     EVP_MD_CTX_cleanup(&mdctx);
46)     return digest_to_hex_string(md_value, md_len);
47) }
48) /*****************************************************************************
49)  Two way hashing: AES, DES, BlowFish, RC4
50)  *****************************************************************************/
```

51) برای استفاده از الگوریتمهای با Hashing دوطرفه باید دو کلید داشت. کلید عمومی را iv در نظر بگیرید و کلید خصوصی را با استفاده از رمز عبور که در ابتدای برنامه وارد می‌کنید، با الگوریتم یکطرفه sha1 تولید می‌کنیم. نکته جالب اینجاست که در صورت ورود رمز عبور اشتباه در هر طرف ارتباط (اگر Login را در نظر نگیریم) باز نمی‌توان پیامهای سالم ارسال و دریافت کرد.

52) با هر بار اجرا شدن تابع با توجه به نوع رمزنگاری انتخاب شده که در متغیر EncryptionType ذخیره شده، متغیرهای ectx و dctx مقداردهی می‌شوند.

```
53) void Encryption::EncryptionInit() {
54)     unsigned char key[EVP_MAX_KEY_LENGTH];
55)     unsigned char iv[EVP_MAX_IV_LENGTH] = "Ben";
56)     EVP_CIPHER_CTX_init(&ectx);
57)     EVP_CIPHER_CTX_init(&dctx);
58)     /* 8 bytes to salt the key_data during key generation. This is an example of
59)      compiled in salt. We just read the bit pattern created by these two 4 byte
60)      integers on the stack as 64 bits of contigous salt material -
61)      ofcourse this only works if sizeof(int) >= 4 */
62)     unsigned int salt[] = { 12345, 54321 };
63)     int i, nrounds = 5;
```

64) در هر case عملیات بدین صورت انجام می‌شود؛ ابتدا key یا همان کلید خصوصی با استفاده از تابع EVP_BytesToKey تولید می‌شود، این تابع رمزعبور را به تعداد nrounds با الگوریتم یکطرفه sha1، digest می‌شود، دقت داشته باشید که حاصل این digestها در دو طرف یکسان خواهد بود اما برگشت پذیر نخواهد بود. با تولید key بدین روش، دو متغیر ectx و dctx برای الگوریتم مورد نظر بر حسب case مقداردهی اولیه می‌شوند تا در توابع مربوط به خود، یعنی encrypt یا decrypt کردن مورد استفاده قرار گیرند.

```
65)     switch (EncryptionType) {
66)     case 0:
67)         //No Encryption Mode
68)         break;
69)     case 1:     //AES Encryption
70)         i = EVP_BytesToKey(EVP_aes_128_cfb(), EVP_sha1(),
71)                 (unsigned char *) &salt, (unsigned char *) Password, strlen(
72)                         Password), nrounds, key, iv);
73)         EVP_EncryptInit_ex(&ectx, EVP_aes_128_cfb(), NULL,
74)                 (unsigned char*) key, (unsigned char*) &iv);
75)         EVP_DecryptInit_ex(&dctx, EVP_aes_128_cfb(), NULL,
76)                 (unsigned char*) key, (unsigned char*) &iv);
77)         break;
78)     case 2: //DES Encryption
79)         i = EVP_BytesToKey(EVP_des_ede_cfb(), EVP_sha1(),
80)                 (unsigned char *) &salt, (unsigned char *) Password, strlen(
```

Encryption Class

```
81)                         Password), nrounds, key, iv);
82)         EVP_EncryptInit_ex(&ectx, EVP_des_ede_cfb(), NULL,
83)                 (unsigned char*) key, (unsigned char*) &iv);
84)         EVP_DecryptInit_ex(&dctx, EVP_des_ede_cfb(), NULL,
85)                 (unsigned char*) key, (unsigned char*) &iv);
86)         break;
87)     case 3: //rc4 Encryption
88)         i = EVP_BytesToKey(EVP_rc4(), EVP_sha1(), (unsigned char *) &salt,
89)                 (unsigned char *) Password, strlen(Password), nrounds, key, iv);
90)         EVP_EncryptInit_ex(&ectx, EVP_rc4(), NULL, (unsigned char*) key,
91)                 (unsigned char*) &iv);
92)         EVP_DecryptInit_ex(&dctx, EVP_rc4(), NULL, (unsigned char*) key,
93)                 (unsigned char*) &iv);
94)         break;
95)     case 4: //Blowfish Encryption
96)         i = EVP_BytesToKey(EVP_bf_cfb(), EVP_sha1(), (unsigned char *) &salt,
97)                 (unsigned char *) Password, strlen(Password), nrounds, key, iv);
98)         EVP_EncryptInit_ex(&ectx, EVP_bf_cfb(), NULL, (unsigned char*) key,
99)                 (unsigned char*) &iv);
100)        EVP_DecryptInit_ex(&dctx, EVP_bf_cfb(), NULL, (unsigned char*) key,
101)                (unsigned char*) &iv);
102)        break;
103)    }
104)    return;
105) }
```

106) متد encrypt بدین صورت عمل می‌کند که ابتدا با توجه به متغیر ectx رشته موجود در plaintext را رمزگزاری کرده و خروجی را در ciphertext قرار می‌دهد، سپس با استفاده از تابع EncryptFinal کاراکتر آخر رشته نیز رمزگزاری شده و به متغیر ciphertext الصاق می‌شود. در انتها حتما دو متغیر ctx باید به مقدار خالی انتصاب شوند تا در استفاده‌های بعدی دچار عدم همخوانی نشویم.

```
107) void Encryption::Encrypt(const char plaintext[256], char ciphertext[256],
108)         int Type) {
109)     if (Type) {
110)         EncryptionInit();
111)         int in_len = 0, out_len = 0;
112)         in_len = strlen(plaintext);
113)         EVP_EncryptUpdate(&ectx, (unsigned char*) ciphertext, &out_len,
114)                 (unsigned char*) plaintext, in_len);
115)         EVP_EncryptFinal_ex(&ectx, (unsigned char*) ciphertext[out_len],
116)                 &out_len);
117)         EVP_CIPHER_CTX_cleanup(&ectx);
118)         EVP_CIPHER_CTX_cleanup(&dctx);
119)     }
120) }
```

121) متد decrypt نیز همانند encrypt البته (عکس آن) بدین صورت عمل می‌کند که ابتدا با توجه به متغیر dctx رشته موجود در ciphertext را رمزگشایی کرده و خروجی را در plaintext قرار می‌دهد، سپس با استفاده از تابع DeccryptFinal کاراکتر آخر رشته نیز رمزگشایی شده و به متغیر plaintext الصاق می‌شود. در انتها حتما دو متغیر ctx باید به مقدار خالی انتصاب شوند تا در استفاده‌های بعدی دچار عدم همخوانی نشویم.

```
122) void Encryption::Decrypt(const char ciphertext[256], char plaintext[256],
```

Encryption Class

```
123)            int Type) {
124)        if (Type) {
125)            EncryptionInit();
126)            int in_len = 0, out_len = 0;
127)            out_len = strlen(ciphertext);
128)            EVP_DecryptUpdate(&dctx, (unsigned char*) plaintext, &in_len,
129)                    (unsigned char*) ciphertext, out_len);
130)            EVP_DecryptFinal_ex(&dctx, (unsigned char*) plaintext[in_len], &in_len);
131)            EVP_CIPHER_CTX_cleanup(&ectx);
132)            EVP_CIPHER_CTX_cleanup(&dctx);
133)        }
134)    }
```

Socket.h –۶–۴–۴

Socket Class Header

```
1)  /*
2)   * Socket.h
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)
8)  #ifndef SOCKET_H_
9)  #define SOCKET_H_
10)
11) #include "Graphics.h"
12) #include "Encryption.h"
13)
14) #include <sys/types.h>
15) #include <sys/socket.h>
16) #include <netdb.h>
17) #include <fcntl.h>       //NonBlocking Sockets [8]
18) #include <sys/time.h>
19)
20) #include <netinet/in.h>
21) #include <arpa/inet.h>
22) #include <unistd.h>
23)
24) این ساختار در حالیت مورد استفاده قرار می‌گیر که نرم‌افزار از اتصال نوع UDP استفاده می‌کند. هدف این
    ساختمان داده، ذخیره اطلاعات مربوط به کاربرانی هست که به Server وصل شده و با Login اعلام شروع ارتباط
    می‌کنند. RemoteAddr اطلاعات آدرس، host نام سیستم متصل شده، service نام یا شماره سرویسی که به
```

Socket Class Header

ارتباط اختصاص داده شده و UserName نام کاربری User است که هنگام اجرای برنامه وارد کرده است.

```
25) struct UDPServiceInfo {
26)     sockaddr_storage RemoteAddr;
27)     char host[1024];
28)     char service[20];
29)     char UserName[20];
30) };
31)
```

32) کلاس Socket از دو کلاس Graphics و Encryption مشتق می‌شود. دقت کنید که ارثبری به صورت public است.

```
33) class Socket: public Graphics, public Encryption {
34) protected:
```

35) این دو متغیر حاوی یک عدد صحیح که بیانگر مشخصه سوکت است می‌باشند؛ mySocketFD در حالت Server و ConnectionFD در حالت Client از نرم‌افزار مورد استفاده قرار می‌گیرند. می‌توان از یک متغیر استفاده کرد، فقط به دلیل خوانایی بیشتر برنامه به صورت مجزا تعریف شده‌اند.

```
36)     int mySocketFD; //used in TCP/UDP server, listening socket descripter
37)     int ConnectionFD; //used in TCP/UDP client, connecting socket descriptor
```

38) چهار متغیر زیر تعیین کننده نحوه تعریف سوکت می‌باشند. در لایه بالاتر مقداردهی می‌شوند.

```
39)     int Domain, Type, Protocol, Flag;
40)     char IPversion[10];
41)     char ListenAddress[INET6_ADDRSTRLEN], ListenPort[10];
42)     bool NonBlockingMode;
```

43) در حالت NonBlocking نیاز به زمان انتظار داریم تا از اتلاف CPU هنگام استفاده جلوگیری شود.

```
44)     timeval TWait; //time wait in nonblocking mode
```

45) دو متغیر زیر مجموعه‌ای از مشخصه‌های سوکت را در خود نگه می‌دارند، هدف استفاده از آنها در تابع select است که راهی بسیار مناسب برای پیدا کردن سوکت فعال را فراهم می‌کند. (توضیح بیشتر داده خواهد شد)

```
46)     fd_set Master_FDs; //master file descriptor list
47)     fd_set Read_FDs; //temp file descriptor list for select()
```

48) این متغیر اطلاعات ماشین متصل شونده را در خود نگه می‌دارد؛ اگر نرم‌افزار در حالت TCP اجرا شده باشد به هنگام accept مقداردهی شده و در صورتیکه در حالت UDP باشد به هنگام recv مقدادهی می‌شود.

```
49)     sockaddr_storage RemoteAddr; // client address
```

50) این دو متغیر برای نگهداری اطلاعات ده کاربر (به ترتیب اولی برای TCP و دومی برای UDP) مورد استفاده قرار می‌گیرند

```
51)     sockaddr_storage RemoteAddrStorage[10]; // client addresses in TCP mode
52)     UDPServiceInfo UDPClientInfo[10];// client addresses in UDP mode
```

53) ربای به دست آوردن IP آدرس device شبکه مانند مورد استفاده قرار می‌گیرد

```
54)     addrinfo* GetAddrInfo();
55)     int GetBind(addrinfo*);          مقید کردن سوکت بر اساس آرگومان
56)     int ListenOn();                  گوش دادن برای دریافت ارتباط جدید
57)     void ManageConnections();        مدیریت ارتباطات موجود
```

58) این متد برای ارسال اطلاعات کاربر از جمله UserName و Password مورد استفاده قرار می‌گیرد

```
59)     int Login();
60)
61)     int SocketBlockingMode();        تبدیل سوکت به حالت غیر بلوکه شدنی
```

Socket Class Header

```
62)
63) public:
64) اگر متغیر true باشد یعنی نرم‌افزار در حال Manage کردن یا ReceiveMode است. اگر false باشد یعنی نرم‌افزار از کار افتاده است و می‌توان دوباره آنرا Run کرد.
65)     bool AppStat;
66)
67)     char HostName[128];          نام ماشین که نرم‌افزار در آن اجرا می‌شود
68)     char UserName[20];           نام کاربری
69)
70)     void DisconnectClients();    برای قطع اتصال کاربران مورد استفاده قرار می‌گیرد
71)
72)     void ClientSendMode();       ارسال پیام کاربر
73)     void ClientReceiverMode();   دریافت پیام از سرور
74)
75)     void FlushAll();             خالی کردن متغیرهای اصلی
76)
77)     Socket();
78)     virtual ~Socket();
79) };
80)
81) #endif /* SOCKET_H_ */
```

Socket.cpp –۷–۴–۴

Socket Class

```
1)  /*
2)  * Socket.cpp
3)  *
4)  *  Created on: Jul 5, 2011
5)  *      Author: ben
6)  */
7)
8)  #include "Socket.h"
9)
10) هدف تعریف این ماکرو این است که به دلیل کپی کردن چند رشته در یک متغیر چند کاراکتر آخر رشته به وجود می‌آیند، به همین خاطر کار این ماکرو این است که رشته را از اول تا آخر (طول ۲۵۶) حرکت کرده و تمام کاراکترهای کد اسکی 0 را با کاراکتر کد اسکی 1 تعویض کند. اساس تعریف این ماکرو به این خاطر بود که در هنگام رمزنگاری یا رمزگشایی در رسیدن به اولین کاراکتر آخر رشته، عملیات متوقف می‌شد، بنابراین تمام رشته‌های آخر رشته به غیر از آخری تبدیل می‌شوند تا عملیات به صورت صحیح انجام شوند.
11) #define Specialize(str)          \
```

Socket Class

```
12)     for (int z = 0; z <= 255; z++) \
13)         if (str[z] == 0)                \
14)             str[z] = 0x01;
15)                                                  عکس ماکرو بالا عمل می‌کند
16) #define unSpecialize(str)        \
17)     for (int z = 0; z <= 255; z++)\
18)         if (str[z] == 0x01)          \
19)             str[z] = 0;
20)
21) int sendall(int, char*, int*);
22) void *get_in_addr(sockaddr*); // get sockaddr: IPv4 or IPv6:
23)
24) Socket::Socket() {
25)     // TODO Auto-generated constructor stub
26)     cout << "Constructing Socket...\n";
27)
28)     AppStat = false;                                  وظعیت نرم‌افزار غیرفعال تعیین
        می‌شود
29)     gethostname(HostName, sizeof HostName);          به دست آوردن نام ماشین
30)                                                  NULL کردن مجموعه‌های مشخصه سوکت
31)     FD_ZERO(&Master_FDs);
32)     FD_ZERO(&Read_FDs);
33) }
34)
35) Socket::~Socket() {
36)     // TODO Auto-generated destructor stub
37)     cout << "Destructing Socket...\n";
38) }
39) نحوه کار این تابع به این شرح است که ابتدا بر حسب حال برنامه (Server یا Client) و سپس بر حسب حالت
    اتصال (TCP یا UDP) مشخص می‌کند که چه اعمالی برای reset کردن برنامه لازم است که انجام شوند.
40) void Socket::FlushAll() {
41) تابع sprintf، عینا printf عمل می‌کند با این تفاوت که حاصل را به جای چاپ در کنسول در متغیر مشخص
    شده (در اینجا Message) چاپ می‌کند. در ادامه با استفاده از ماکروی easyLog چاپ پیام انجام می‌شود.
    عبارت "italic" به عنوان tag استفاده می‌شود.
42)     sprintf(Message, "Program: Reset mode activated\n");
43)     easyLog("italic");
44)     AppStat = false;
45)     if (!strcmp(ProgramMode, "Server")) {
46)         if (Type == SOCK_STREAM) {
47)             DisconnectClients();
48)             sleep(1);
49)             if (mySocketFD > 0) {
50)                 shutdown(mySocketFD, SHUT_RDWR);    بستن سوکت با حالت مشخص شده
51)                 close(mySocketFD);                  دستور بستن دوطرفه ارتباط سوکت
52)                 mySocketFD = -1;
53)             }
54)         } else if (Type == SOCK_DGRAM) {
```

Socket Class

```
55)                 DisconnectClients();
56)                 sleep(1);
57)                 if (mySocketFD > 0) {
58)                     close(mySocketFD);
59)                     mySocketFD = -1;
60)                 }
61)             }
62)             sprintf(Message, "Program: Server Stopped\n");
63)             easyLog("error");
64)         } else if (!strcmp(ProgramMode, "Client")) {
65)             if (Type == SOCK_STREAM) {
66)                 if (ConnectionFD > 0) {
67) با بستن یکطرفه ارتباط در حالت دریافت در سمت Client، سرور متوجه قطع ارتباط به دلیل اتصال‌گرا بودن
    می‌شود. برای اجرای فرایندهای بعدی کمی زمان نیاز است به همین دلیل ۱ ثانیه sleep داده‌ایم. به دلیل زیاد
    بودن این فرایندها از ذکر آنها خوداری می‌کنم، با مطالعه بهتر کد به لزوم آن پی می‌برید.
68)                     shutdown(ConnectionFD, SHUT_WR);
69)                     sleep(1);
70)                     shutdown(ConnectionFD, SHUT_RD);
71)                     close(ConnectionFD);
72)                     ConnectionFD = -1;
73)                 }
74)             } else if (Type == SOCK_DGRAM) {
75)                 if (ConnectionFD > 0) {
76) به دلیل بدون اتصال بودن پروتکل UDP، هنگام بسته شدن سوکت در یک سمت، سمت دیگر از قطع شدن ارتباط
    آگاه نمی‌شود، بنابراین با ارسال پیغامی مبنی بر قطع ارتباط به سرور فرایند را برای هدفهای بعدی تکمیل می‌کنیم.
77)                     char *ConnInfo=new char();
78)                     strcpy(ConnInfo, “00000000000000000000”);
79)                     ClientCheckUpdate(ConnInfo);
80)                     char sendBuffer[256] = "";
81)                     strcpy(sendBuffer, "%DISCONNECTED%");
82)                     strcpy(sendBuffer, md5(sendBuffer).c_str());
83)                     sendto(ConnectionFD, sendBuffer, sizeof sendBuffer, 0,  sockaddr *)
                                        &RemoteAddr, sizeof RemoteAddr);
84)                     sleep(1);
85)                     close(ConnectionFD);
86)                     ConnectionFD = -1;
87)                 }
88)             }
89)             sprintf(Message, "Program: Client Stopped\n");
90)             easyLog("error");
91)         }
92)         FD_ZERO(&Master_FDs); // clear the master and temp sets
93)         FD_ZERO(&Read_FDs);
94)         AppStat = false;
95)     }
96) این متد برای تبدیل سوکتها از حالت بلوکه‌پذیر به حالت بلوکه‌ناپذیر مورد استفاده قرار می‌گیرد. به دلیل عدم بلوکه
    شدن برخی توابع مربوط به سوکت نیاز به زمانی برای انتظار است تا در استفاده از CPU صرفه‌جویی شود. زیرا توابع
    دریافت و ارسال در حلقه نامحدود قرار گرفته‌اند و در صورت عدم بوکه شدن روی آنها باعث پایین آوردن عملکرد
```

Socket Class

CPU خواهند شد. این زمان انتظار در دستور select گنجانده شده و با اتمام ضرب‌العجل، تابع مقدار timeout را برمی‌گرداند.

```
97)  int Socket::SocketBlockingMode() {
98)      if (NonBlockingMode) {
99)          TWait.tv_sec = 0;
100)         TWait.tv_usec = 750000;
101)         if (!strcmp(ProgramMode, "Server")) {
```

102) برای اعمال NonBlocking ابتدا flagهای سوکت را گرفته و flag حالت NonBlocking یعنی O_NONBLOCK را با آن Or بیتی می‌کنیم و با تابع fcntl به آن اعمال می‌کنیم.

```
103)             int fdflags = fcntl(mySocketFD, F_GETFL);
104)             if (fcntl(mySocketFD, F_SETFL, fdflags | O_NONBLOCK) < 0) {
105)                 perror("fcntl");
106)                 sprintf(Message, "NonBlocking: Error on setting socket %d to
                                                 NonBlock.\n",mySocketFD);
107)                 easyLog("error");
108)                 return 1;
109)             } else {
110)                 sprintf(Message, "NonBlocking: Socket %d successfully set to
NonBlock.\n",                    mySocketFD);
111)                 easyLog("success");
112)             }
113)         } else if (!strcmp(ProgramMode, "Client")) {
114)             int fdflags = fcntl(ConnectionFD, F_GETFL);
115)             if (fcntl(ConnectionFD, F_SETFL, fdflags | O_NONBLOCK) < 0) {
116)                 perror("fcntl");
117)                 sprintf(Message, NonBlocking: Error on setting socket %d to
                                                 NonBlock.\n",ConnectionFD);
118)                 easyLog("error");
119)                 return 1;
120)             } else {
121)                 sprintf(Message, NonBlocking: Socket %d successfully set to
                                                 NonBlock.\n",ConnectionFD);
122)                 easyLog("success");
123)             }
124)         }
125)     } else {
126)         memset(&TWait, 0, sizeof TWait);
127)     }
128)     return 0;
129) }
```

130) ساختمان داده addrinfo همانطور که در فصلهای قبلی توضیح داده شد، به عنوان خروجی این تابع استفاده شده است. حاصل این تابع، IP آدرسهای ماشینی است که نرم‌افزار بر روی آن اجرا شده است و این به دلیل آن است که flag با ثابت AI_PASSIVE مقداردهی شده است، این ثابت بیانگر این است که از IPهای همین ماشین استفاده کن. خروجی تابع getaddrinfo می‌تواند چند مقدار باشد که بسته به تعداد deviceهای شبکه موجود در ماشین است.

```
131) addrinfo* Socket::GetAddrInfo(void) {
132)     struct addrinfo hints, *ai;
```

Socket Class

```
133)     int rv;
134)
135)     memset(&hints, 0, sizeof hints);
136)     hints.ai_family = Domain;
137)     hints.ai_socktype = Type;
138)     hints.ai_protocol = Protocol;
139)     hints.ai_flags = Flag;
140)
141)     if ((rv = getaddrinfo(ListenAddress, ListenPort, &hints, &ai)) != 0) {
142)         sprintf(Message, Error On Getting Network Address Information!!
(%s)\n",gai_strerror(rv));
143)         easyLog("error");
144)         return NULL;
145)     }
146)     return ai;
147) }
```

148) این متد از آدرسهای موجود در خروجی حاصل از متد GetAddrInfo() استفاده کرده و بر اولین آدرسی که بتواند مقید شود، خروجی مفوقیت آمیز را بازگشت می‌دهد.

```
149) int Socket::GetBind(addrinfo* ai) {
150)     addrinfo *p;
151)     int Listener = -1;
152)     int yes = 1; // for setsockopt() SO_REUSEADDR, below
153)     for (p = ai; p != NULL; p = p->ai_next) {
154)         Listener = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
155)         if (Listener < 0) {
156)             continue;
157)         }
```

158) در بعضی مواقع پورت مورد نظر در حالت time_wait است، با این کد می‌توان پورت مورد نظر را مجددا استفاده کرد که با استفاده از دستور زیر میسر است.

```
159)         setsockopt(Listener, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));
160)         if (bind(Listener, p->ai_addr, p->ai_addrlen) < 0) {
161)             close(Listener);
162)             continue;
163)         }
164)         break;
165)     }
166)     if (Listener < 0) {
167)         sprintf(Message, "Socket: Error on creating socket\n");
168)         easyLog("error");
169)     } else {
170)         sprintf(Message, "Socket: Successfully created with Num. %d\n",Listener);
171)         easyLog("success");
172)     }
173)     // if we got here, it means we didn't get bound
174)     if (p == NULL) {
175)         sprintf(Message, "Bind Failed: Server binding failed on %s:%s
                                    (%s).\n",ListenAddress, ListenPort, IPversion);
176)         easyLog("error");
177)         return -1;
```

Socket Class

```
178)     }
179)     sprintf(Message, Bind Successful: Server binded on %s:%s (%s) socket
%d.\n",                                  ListenAddress, ListenPort, IPversion,
Listener);
180)     easyLog("success");
181)     freeaddrinfo(ai); // all done with this
182)     return Listener;
183) }
184)
185) int Socket::ListenOn() {
186)     if (listen(mySocketFD, 10) == -1) {
187)         sprintf(Message, Listen Error: Server listening on socket %d failed.\n",
mySocketFD);
188)         easyLog("error");
189)         return -1;
190)     }
191)     FD_SET(mySocketFD, &Master_FDs); // add the listener to the master set
192)     sprintf(Message, "Listen Successful: Server started listening on socket %d for
incoming                                  connections.\n", mySocketFD);
193)     easyLog("success");
194)     return 1;
195) }
```

196) قلب سرور همین متد است. منطق این متد بدین صورت است که در یک حلقه بینهایت منتظر ارتباط جدید می‌ماند، با برقراری ارتباط منتظر Login شدن Client می‌ماند. با موفقیت آمیز بودن مرحله قبل، اطلاعات کاربر در ساختمان داده‌های مورد نظر ذخیره می‌شود. سرور منتظر دریافت داده‌ها می‌ماند، دستور select مشخص می‌کند که آیا داده‌ای دریافت شده یا خیر. با دریافت داده، محتوای آن بررسی شده و به سایر Clientها ارسال می‌شود.

```
197) void Socket::ManageConnections() {
```

198) مشخصه سوکت یک عدد صحیح مثبت است؛ از آنجاییکه در برنامه مشخص نمی‌شود که آخرین مشخصه کدام است نیاز به نگهداری شماره آن داریم. متغیر زیر همین هدف را دنبال می‌کند. استفاده از آن دستور select و برای بهینه‌سازی حلقه دریافت و ارسال داده نقش اساسی را ایفا می‌کند، زیرا سقفی برای محدود کردن حلقه تعیین می‌کند.

```
199)     int fdmax; // maximum file descriptor number
```

200) این متغیر مشخصه سوکت ارتباط جدید را در خود نگه می‌دارد و لزوما بزرگترین عدد در میان سایر مشخصه‌ها را می‌تواند داشته باشد. همانطورکه واضح بود در تابع accept مورد استفاده قرار می‌گیرد و کاربردی در ارتباط بدون اتصال ندارد.

```
201)     int newfd; // newly accept()ed socket descriptor
```

202) این متغیر فقط به منظور داشتن تعداد ارتباطات مورد استفاده قرار می‌گیرد؛ با برقراری ارتباط جدید به آن یک واحد اضافه شده و با از دست دادن ارتباط از آن کم می‌شود، هدف آن جلوگیری از ارتباط بیش از ۱۰ تا است.

```
203)     int cntConn = 0;//get track of the number of connections
```

204) متغیری است که در حلقه‌ها مورد استفاده قرار می‌گیرد که همیشه حد پایین آن مقدار مشخصه سوکت سرور یعنی mySocketFD و حد بالای آن همیشه مقدار fdmax را خواهد داشت.

```
205)     int SocketFD = 0; //loop through the socketfds
```

206) مقدار این ثابت به این دلیل یکی بیشتر از mySocketFD است چون بد از آن مقدار مشخصه‌ها به ارتباطات جدید تعلق می‌گیرد و هدف این ثابت به دست آوردن مقدار مشخصه سوکت اولین اتصال است.

Socket Class

```
207)    const int Floorfd = mySocketFD + 1; // connection FDs start at 0
208)    socklen_t addr_len;     متغیری برای نگه داشتن طول آدرس
209)    char buf[256];          داده‌های دریافتی و ارسالی در این متغیر نگه‌داری می‌شوند
210)    هر کاراکتر این این متغیر متناظر با یک GtkCheckButton است. بدین صورت که اگر مقدار کاراکتر '0' باشد یعنی GtkCheckButton باید غیر فعال شود و اگر مقدار کاراکتر آن '1' باشد GtkCheckButton متناظر با اندیس آن باید فعال باشد. مقادیر کاراکتری این متغیر با برقراری ارتباط جدید یا از دست دادن ارتباطی تغییر می‌کنند و بالطبع تاثیر آن بر روی GtkCheckButton مشهود خواهد بود. با برقراری ارتباط جدید و بعد از Login، اندیس متناظر که با استفاده از متغیر cntConn مشخص می‌شود مقدار '1' گرفته و تابع ClientCheck با آرگومان GetUserName فراخوانی می‌شود و با از دست دادن ارتباط اندیس مربوط به کاربر این متغیر به مقدار '0' تخصیص و تابع ClientCheck فراخوانی می‌شود.
211)    char ConnectionInfo[10] = "0";//used mask to set check boxes
212)    طرح جالب دیگری که پیاده‌سازی کردم این بود که به کاربران این امکان داده شد تا انتخاب کنند به کدان کاربران تمایل به ارسال داده دارند. این طرح در قالب یک رشته بیت ۱۰ کاراکتری بیانگر کاربران انتخاب شده درون پیام جاسازی شده که در متغیر زیر کپی شده و مورد استفاده قرار می‌گیرد.
213)    char sendConnectionInfo[10] = "0";//mask used to send data
214)    متغیری با ۱۰ رشته ۲۰ کاراکتری به منظور ذخیره نام کاربری Clientها
215)    char ClientUserName[10][20] = { "" };
216)    گرفتن نام کاربری Clientی که جدیدا ارتباط برقرار کرده
217)    char getClientUserName[20] = "";
218)    گرفتن رمز عبور Clientی که جدیدا ارتباط برقرار کرده، این رمز که به صورت md5 رمزنگاری شده دریافت می‌شود
219)    char getClientPass[15] = "";
220)    int nbytes, nbytesall;          خروجی برای توابع ارسال و دریافت
221)    char remoteIP[INET6_ADDRSTRLEN];        متغیری برای ذخیره آدرس
222)    int j;
223)    نتیجه Login یک کاربر را در خود نگه می‌دارد، اندیس متناظر از ۰ تا ۱۰ می‌باشد (همانند متغیرهای بالایی)
224)    bool Authenticate[10] = { 0 };
225)    تاکنون بزرگترین مقدار مشخصه مربوط به مشخصه سوکت خودمان است، مقداردهی را به صورت زیر انجام می‌دهیم
226)    fdmax = mySocketFD;
227)    char host[1024];    متغیر برای نگه داشت نام ماشین کاربر در حالت ارتباط بدون اتصال
228)    char service[20];   متغیر برای نگه داشت نام سرویس یا شماره پورت کاربر در حالت ارتباط بدون اتصال
229)    مقداردهی‌ای اولیه برای متغیرها به صورت مقادیر NULL برای جلوگیری از خطا صورت می‌گیرد.
230)    for (int i = 0; i <= 9; i++) {
231)        strcpy(UDPClientInfo[i].host, "");
232)        strcpy(UDPClientInfo[i].service, "");
233)        strcpy(UDPClientInfo[i].UserName, "");
234)        bzero(&UDPClientInfo[i].RemoteAddr, sizeof UDPClientInfo[i].RemoteAddr);
235)        ClientCheck(-(i + 1), "");
236)    }
237)    addr_len = sizeof RemoteAddr;
238)    حلقه اصلی که تا بروز خطا یا Stop کردن به کار خود ادامه می‌دهد
```

Socket Class

```
239)    if (Type == SOCK_STREAM) {
240)        for (;;) {
241)            strcpy(buf, "");
242)            Read_FDs = Master_FDs; // copy it
243)  شرط زیر بررسی می‌کند که آیا نرم‌افزار در حالت بدون بوکه شدن است یا خیر؟ اگر بدون بلوکه شدن است پس
      دستور select را با پارامتر TWait که زمان انتظار را بیان می‌کند اجرا می‌شود، در غیر این صورت دستور slect
      بدون زمان انتظار اجرا می‌شود. خروجی دستور select در صورتی که برابر با 1- باشد یعنی برنامه با خطا مواجه
      شده و دستور FlushAll به منظور reset کردن اجرا می‌شود.
244)            if (NonBlockingMode) {
245)                TWait.tv_sec = 0;
246)                TWait.tv_usec = 750000;
247)                if (select(fdmax + 1, &Read_FDs, NULL, NULL, &TWait) == -1) {
248)                    sprintf(Message,  "Internal Server Error!: Error on socket.\n");
249)                    easyLog("error");
250)                    if (AppStat)
251)                        FlushAll();
252)                    return;
253)                }
254)            } else {
255)                if (select(fdmax + 1, &Read_FDs, NULL, NULL, NULL) == -1) {
256)                    sprintf(Message, Internal Server Error!: Error on socket.\n");
257)                    easyLog("error");
258)                    if (AppStat)
259)                        FlushAll();
260)                    return;
261)                }
262)            }
263)  با اجرای موفقیت آمیز دستور select حاصل دستور آرگومان Read_FDs است. این دستور بدین صورت عمل
      می‌کند که هر سوکتی اگر داده‌ای را ارسال کرده باشد پس بای خوانده شود بنابراین مشخصه متناظر با آن سوکت
      به عنوان خواند علامت زده می‌شود. حال بایت تمام سوکتها را بررسی کنیم تا از میان آنها سوکتهای علامت زده را با
      دستور FD_ISSET بیابیم:
264)            for (SocketFD = mySocketFD; SocketFD <= fdmax; SocketFD++) {
265)                if (FD_ISSET(SocketFD, &Read_FDs)) { // we got one!!
266)  اگر سوکت علامت زده شده برابر با مشخصه سوکت خودمان باشد بدین معنیست که ارتباط جدید وجود دارد پس
      باید accept شود.
267)                    if (SocketFD == mySocketFD) {
268)  در صورتیکه تعداد ارتباطات موجود کمتر از ۱۰ تا باشد ادامه می‌دهیم در غیر این صورت بعد از پذیرفتن ارتباط، آنرا
      قطع می‌کنیم تا کاربر از عدم پذیرش ارتباط آگاه شود
269)                        if (cntConn == 10) {
270)  اگر وارد این قسمت شدیم یعنی به حد نصاب ارتباطات رسیده‌ایم پس ابتدا ارتباط را وصل می‌کنیم
271)                            newfd = accept(mySocketFD, struct sockaddr *)
      &RemoteAddr,                                    &addr_len);
272)  سپس آنرا می‌بندیم
273)                            close(newfd);
274)  پیغام مناسب را چاپ می‌کنیم
275)                            sprintf(Message, "Connection: Maximum connections
```

Socket Class

```
reached\n\tNew connection from %s refused!\n", remoteIP);
276)                    easyLog("italic");
277)                    continue;
278)                }
279)                    اگر به این بخش رسیدیم یعنی می‌توان ارتباط جدید را قبول کنیم
280)                addr_len = sizeof RemoteAddr;
281)                newfd = accept(mySocketFD, (struct sockaddr *) &RemoteAddr, &addr_len);
282)                if (newfd == -1) {    بررسی می‌کنیم که پذیرفتن ارتباط با خطا مواجه نشده باشد
283)                    sprintf(Message, "Server: Error on accepting connection\n");
284)                    easyLog("error");
285)                } else {
286)                    در این قسمت با بررسی حالت بلوکه بودن، مشخصه سوکت را تنظیم می‌کنیم
287)                    if (NonBlockingMode) {
288)                        int fdflags = fcntl(newfd, F_GETFL);
289)                        if (fcntl(newfd, F_SETFL, fdflags | O_NONBLOCK) < 0) {
290)                            perror("fcntl");
291)                            sprintf(Message, NonBlocking: Error on setting socket %d to NonBlock.\n", newfd);
292)                            easyLog("error");
293)                        } else {
294)                            sprintf(Message, "NonBlocking: Socket %d successfully set to NonBlock.\n", newfd);
295)                            easyLog("success");
296)                        }
297)                    }
298)                تعداد ارتباطات را تعیین و مشخصه سوکت جدید را به متغیر مجموعه مشخصه‌های سوکت اعمال می‌کنیم تا در آینده استفاده شود.
299)                cntConn++;
300)                FD_SET(newfd, &Master_FDs);
301)                همیشه مقدار بزرگترین مشخصه سوکت را باید نگه داریم.
302)                if (newfd > fdmax) { // keep track of the max
303)                    fdmax = newfd;
304)                }
305)                مشخصات آدرس کاربر جدید را طبق اندیس متناظر که حتما در محدوده ۰ تا ۹ است را ذخیره می‌کنیم
306)                RemoteAddrStorage[newfd - Floorfd] = RemoteAddr;
307)                مشخصات ارتباط جدید را به صورت پیغامی چاپ می‌کنیم.
308)                inet_ntop(RemoteAddr.ss_family, get_in_addr( (struct sockaddr*) &RemoteAddr), remoteIP, INET6_ADDRSTRLEN);
309)                sprintf(Message,
310)                    "Connection: new connection from %s on socket %d\n", remoteIP, newfd);
311)                easyLog("bold");
312)            }
```

Socket Class

```
313)                    } else {
314)  در صورتیکه مشخصه سوکت علامت خورده حاصل از دستور select برابر با مشخصه سوکت ما نباشد بدین
      معنیست که از سوکت دیگری داده دریافت کرده‌ایم، پس باید عمل rec به معنی دریافت داده را انجام دهیم.
315)                    inet_ntop(RemoteAddrStorage[SocketFD - Floorfd].ss_family,
                                get_in_addr((struct sockaddr*)
                        &RemoteAddrStorage[SocketFD - Floorfd]), remoteIP,
                                INET6_ADDRSTRLEN);
316)                    if ((nbytes = recv(SocketFD, buf, sizeof buf, 0)) <= 0) {
317)  در صورتیکه طول داده دریافتی برابر با 0 باشد بدین معنیست که ارتباط سوکت متناظر قطع شده است. دقت کنید
      که این حالت فقط برای ارتباط نوع انصال‌گرا قابل تشخیص است. اما اگر مقدار بازگشتی برابر 1- باشد یعنی با خطا
      مواجه شده‌ایم. پیغامهای مناسب را چاپ می‌کنیم. فرایندهای بعد از از دست دادن ارتباط بدین قرار است که ابتدا
      تعداد کاربران یکی کم می‌شود، چک‌باکس متناظر غیرفعال می‌شود، از مجموعه مشخصه سوکتها حذف می‌شود و در
      نهایت به دیگر کاربران اعلام می‌شود تا لیست خود را به روز رسانی کنند.
318)                        if (nbytes == 0) {
319)                            // connection closed
320)                            sprintf(Message, "Connection Lost: connection %s on
      socket %d                                 hung up\n",remoteIP,
      SocketFD);
321)                            easyLog("lost");
322)                        } else {
323)                            sprintf(Message, "Receive Error: cannot get data from
      %s on socket                              %d\n",remoteIP, SocketFD);
324)                            easyLog("error");
325)                        }
326)  فرایند حذف به هنگام از دست دادن ارتباط از اینجا آغاز می‌شود.
327)  کاهش تعداد ارتباطات
328)                        cntConn--;
329)  کاراکتر متناظر با GtkCheckButton بر حسب اندیس متناظر با کاربر به مقدار '0' انتساب می‌شود
330)                        strncpy(&ConnectionInfo[SocketFD - Floorfd], "0", 1);
331)  GTKCheckButton مورد نظر از انتخاب در می‌آید، علامت منفی پشت اندیس متناظر با کاربر بیانگر این عمل
      است.
332)                        ClientCheck(-(SocketFD - Floorfd + 1), "");
333)  تصدیق Login کاربر لغو می‌شود
334)                        Authenticate[SocketFD - Floorfd] = false;
335)  نام کاربری پاک می‌شود
336)                        strcpy(ClientUserName[SocketFD - Floorfd], "");
337)  مشخصه سوکت بسته شده و پاک می‌شود
338)                        shutdown(SocketFD, SHUT_RDWR);
339)                        close(SocketFD); // bye!
340)                        FD_CLR(SocketFD, &Master_FDs);
341)  از این قسمت به دیگر کاربران اعلام می‌شود که کدام کاربر ارتباط خود را از دست داده است و لیست را به آنها
      ارسال می‌کند تا به روز رسانی را انجام دهند.
342)                        strcpy(buf, "%CONF%");
343)                        strcpy(&buf[10], ConnectionInfo);
```

Socket Class

```
344)                                    نام کاربران به تعداد ۲۰ کاراکتر از اندیس ۲۰ام به بافر اضافه می‌شود
345)                            for (int z = 1; z <= 10; z++) {
346)                                strcpy(&buf[20 * z], ClientUserName[z - 1]);
347)                            }
348)                                    به دلیل وجود چندید کاراکتر انتهای رشته، تبدیل زیر صورت می‌گیرد
349)                            Specialize(buf);
350)                                    عملیات رمزنگاری صورت می‌گیرد
351)                            Encrypt(buf, buf, EncryptionType);
352)                                    اطلاعات کاربران ارسال می‌شود
353)                            for (j = Floorfd; j <= fdmax; j++) {
354)                                if (!strncmp(&ConnectionInfo[j - Floorfd], "1", 1)) {
355)                                    int tmp = sizeof buf;
356)                                    inet_ntop( RemoteAddrStorage[SocketFD -
Floorfd].ss_family,
            get_in_addr((struct sockaddr*)
                        &RemoteAddrStorage[SocketFD - Floorfd]),
                                                remoteIP, INET6_ADDRSTRLEN);
357)                                    if (sendall(j, buf, &tmp) == -1) {
358)                                        sprintf(Message,  "Configuration: Error on
sending ClientList                                                  to %s on
socket %d!\n",remoteIP, j);
359)                                        easyLog("error");
360)                                    } else {
361)                                        sprintf(Message, "Configuration: ClientList
successfully sent                                                   to %s on
socket %d!\n",remoteIP, j);
362)                                        easyLog("success");
363)                                    }
364)                                }
365)                            }
366)                        } else {
367)     زمانی به این قسمت می‌رسیم که داده‌ها بدون خطا دریافت شده‌اند، پس هدف پخش آنها به کاربران است، اما ابتدا
        باید بررسی کنیم که کاربری که داده‌ها را ارسال کرده است Login کرده یا خیر.
368)                            nbytesall = nbytes;
369)                                    ابتدا داده را رمزگشایی می‌کنیم
370)                            Decrypt(buf, buf, EncryptionType);
371)                                    کاراکترهای انتهای رشته را به حالت اولیه برمی‌گردانیم
372)                            unSpecialize(buf);
373)     این شرط به ما نشان می‌دهد که کاربر Login کرده یا خیر؛ در صورتیکه Login نکرده باشد، داده‌ها برای تصدیق
        اطلاعات نام کاربری و رمز عبور بررسی می‌شوند، اما در صورت Login کاربر داده‌ها به عنوان پیغام حساب شده و
        به دیگر کاربران ارسال می‌شوند.
374)                            if (!Authenticate[SocketFD - Floorfd]) {
375)     در صورت عدم Login وارد این قسمت می‌شویم، نام کاربری و رمز عبور را استخراج می‌کنیم.
376)                                strncpy(ClientUserName[SocketFD - Floorfd], buf, 20);
377)                                strncpy(getClientPass, &buf[20], 15);
378)     صحت رمز عبور را بررسی می‌کنیم؛ در صورت درستی آن، پیغام مناسب را چاپ کرده، اطلاعات کاربر را ذخیره کرده
```

Socket Class

و GtkCheckButton منتاظر را فعال می‌کنیم، در ادامه به دیگر کاربران، افزوده شدن کاربر جدید را اعلام می‌کنیم.

```
379)                    if (!strcmp(getClientPass, Password)) {
380)  در صورتیکه رمز عبور صحیح باشد وارد این قسمت می‌شویم، ابتدا تصدیق Login را اعمال می‌کنیم و سپس پیغامی مبنی بر موفقیت بودن را چاپ می‌کنیم.
381)                        Authenticate[SocketFD - Floorfd] = true;
382)                        sprintf(Message, Login Successful: connection from
    %s on                                   socket %d\n",remoteIP,
    SocketFD);
383)                        easyLog("success");
384)  اندیس متناظر را برای کاربر فعال می‌کنیم و GtkCheckButton را نیز بر حسب نام کاربری مقداردهی می‌کنیم
385)                        strncpy(&ConnectionInfo[SocketFD - Floorfd], "1",
    1);
386)                        ClientCheck(SocketFD - Floorfd + 1,
    ClientUserName[SocketFD -                               Floorfd]);
387)  به بقیه کاربران اعلام می‌کنیم که کاربر جدید با چه نام کاربری اضافه شده است، البته تمام اطلاعات کاربران موجود به صورت یکجا ارسال می‌شود.
388)                        strcpy(buf, "%CONF%");
389)                        strcpy(&buf[10], ConnectionInfo);
390)                                                  اضافه کردن نام کاربران
391)                        for (int z = 1; z <= 10; z++) {
392)                            strcpy(&buf[20 * z], ClientUserName[z - 1]);
393)                        }
394)                        Encrypt(buf, buf, EncryptionType);      رمزگزاری بر روی
    داده‌ها
395)                        for (j = Floorfd; j <= fdmax; j++) {
396)                            if (!strncmp(&ConnectionInfo[j  - Floorfd], "1", 1))
    {
397)                                int tmp = sizeof buf;
398)                                inet_ntop(RemoteAddrStorage[j -
    Floorfd].ss_family,
            _in_addr((struct sockaddr*)
                        &RemoteAddrStorage[j - Floorfd]), remoteIP,
                            INET6_ADDRSTRLEN);
399)                                if (sendall(j, buf, &tmp) == -1) {
400)                                    sprintf(Message, Configuration: Error on
    sending                                 ClientList to %s on socket
    %d!\n", remoteIP, j);
401)                                    easyLog("error");
402)                                } else {
403)                                    sprintf(Message, "Configuration: ClientList
    successfully                            sent to %s on socket
    %d!\n",remoteIP, j);
404)                                    easyLog("success");
405)                                }
406)                            }
407)
```

Socket Class

```
408)                                    }
409)                                    continue;
410)                                } else {
411)  هنگامیکه رمز عبور صحیح نباشد وارد این قسمت می‌شویم، ابتدا پیغامی مبنی بر عدم صحت اطلاعات چاپ می‌کنیم
412)                                    sprintf(Message, "Login Failed: connection from %s on socket %d\n",remoteIP, SocketFD);
413)                                    easyLog("error");
414)  سپس پیغام کنترلی مبنی بر عدم Login موفق را به کاربر ارسال می‌کنیم، به دلایل امنیتی آنرا md5 کرده و سپس ارسال می‌کنیم
415)                                    strcpy(buf, "%LOGIN_FAILED%");
416)                                    strcpy(buf, md5(buf).c_str());
417)                                    send(SocketFD, buf, sizeof buf, 0);
418)                                    continue;
419)                                }
420)                            }
421)  در صورتیکه پیغام به درستی دریافت شود و صحت Login کاربر موجود باشد به این بخش می‌رسیم، یعنی از کاربر پیغام دریافت کرده‌ایم
422)                            sprintf(Message, "Receive Successful: %d bytes received from %s on socket %d\n", nbytes, remoteIP, SocketFD);
423)                            easyLog("success");
424)  اگر یادمان باشد داده‌ها را کمی بالاتر رمزگشایی کردیم، در صورتی که دوباره رمزگشایی می‌کردیم، داده‌های اصلی را از دست می‌دادیم. حال ۱۰ کاراکتر اول را که بیانگر کاربران منتخب برای ارسال پیغام هستند را کپی می‌کنیم.
425)                            strncpy(sendConnectionInfo, buf, 10);
426)  پیغام را مرتب کرده و رمزنگاری می‌کنیم
427)                            Specialize(buf);
428)                            strcpy(buf, &buf[10]);
429)                            Encrypt(buf, buf, EncryptionType);
430)  در این بخش بر اساس کاربران منتخب که در متغیر SendConnectionInfo کپی شدند، پیغام را ارسال می‌کنیم؛ هر اندیس متناظر با یک کاربر است، کاراکتر '1' به معنای انتخاب '0' به معنای عدم انتخاب کاربر مورد نظر است.
431)                            for (j = Floorfd; j <= fdmax; j++) {
432)                                // send to everyone!
433)                                if (FD_ISSET(j, &Master_FDs)) {
434)                                    // except me{mySocketFD} and sender{SocketFD}
435)                                    if (j != mySocketFD && j != SocketFD && Authenticate[j - Floorfd] && !strncmp(&sendConnectionInfo[j - Floorfd], "1", 1)) {
436)                                        inet_ntop(RemoteAddrStorage[j - Floorfd].ss_family, get_in_addr( (struct sockaddr*) &RemoteAddrStorage[j - Floorfd]), remoteIP, INET6_ADDRSTRLEN);
437)                                        if (sendall(j, buf, &nbytesall) == -1) {
438)                                            sprintf(Message, "Send Error: %d bytes of data
```

Socket Class

```
sent to %s                                                    on socket %d!\n",
nbytesall, remoteIP, j);
                                        easyLog("error");
                                    } else {
                                        sprintf(Message, Send Successful: %d bytes
sent to %s on                                                 socket %d!\n",
nbytesall, remoteIP, j);
                                        easyLog("success");
                                    }
                                }
                            }
                        }
                    } // END handle data from client
                } else { // END got new incoming connection
                    //cout << "********time out*******\n";
                }
            } // END looping through file descriptors
        } // END for(;;)--and you thought it would never end!
    } else if (Type == SOCK_DGRAM) {ارتباط از نوع بدون اتصال
        for (;;) {
            strcpy(buf, "");
در صورتیکه حالت بدون بلوکه شدن فعال باشد مجبور به اضافه کرد دسور select برای اعمال زمان انتظار
می‌شویم.
            if (NonBlockingMode) {
                TWait.tv_sec = 0;
                TWait.tv_usec = 750000;
                select(NULL, NULL, NULL, NULL, &TWait);
            }
داده‌ها را دریافت و اطلاعات کاربر ارسال کننده را در متغیرهای مربوطه ذخیره می‌کنیم، این اطلاعات در ادامه مورد
استفاده قرار می‌گیرند، به دلیل ارتباط بدون اتصال این اطلاعات از حساسیت بیشتری برخوردارند.
            if ((nbytes = recvfrom(mySocketFD, buf, sizeof buf, 0, (sockaddr *)
&RemoteAddr,                                                  &addr_len)) <= 0) {
                inet_ntop(RemoteAddr.ss_family, get_in_addr((sockaddr*)
&RemoteAddr),                                                 remoteIP, INET6_ADDRSTRLEN);
با استفاده از دستور زیر، اطلاعات کاربر که در زمان ارسال داده‌ها ذخیره شد را به نام ماشین و نام سرویس تبدیل
می‌کنیم.
                getnameinfo((sockaddr *) &RemoteAddr, sizeof RemoteAddr, host,
sizeof host,                                                  service, sizeof service, 0);
                if (NonBlockingMode) {
اگر در حالت بلوکه نشدن باشیم در صورت رخداد خطا در دریافت داده‌هه نمی‌تان گفت که به خاطر عدم دریافت
صحیح داده‌ها این اتفاق رخ داده یا به خاطر time-out خوردن
                } else {
                    perror("recvfrom");
                    sprintf(Message, "Receive Error: cannot get data from %s with
%s,%s\n",                                                     remoteIP, host, service);
                    easyLog("error");
```

Socket Class

```
474)                }
475) در این بخش بررسی شد که آیا پیغام دریافتی گزارش می‌دهد که کاربر متناظر با ارسال کننده داده‌ها، ارتباط خود را قطع کرده است یا خیر؟
476)            } else if (!strcmp(buf, md5("%DISCONNECTED%").c_str())) {
477) در صورت ورود به این بخش یعنی کاربر ارتباط خودرا قطع کرده پس باید فرایند حذف اطلاعات او را آغاز کرد. ابتدا آدرس، نام ماشین و نام سرویس کاربر متناظر را به دست آورده و پیغامی مبنی بر قطع ارتباط او چاپ می‌کنیم.
478)                inet_ntop(RemoteAddr.ss_family, get_in_addr((sockaddr*)
&RemoteAddr),                                    remoteIP, INET6_ADDRSTRLEN);
479)                getnameinfo((sockaddr *) &RemoteAddr, sizeof RemoteAddr, host,
sizeof host,                                    service, sizeof service, 0);
480)                sprintf(Message, "Connection Removed: connection from %s with
%s,%s\n",                                    remoteIP, host, service);
481)                easyLog("lost");
482) به دنبال کاربری با مشخصات وی در ساختمان داده‌ای که اطلاعات کاربران را ذخیره می‌کنیم می‌گردیم، در صورت پیدا کردن عملیات حذف را انجام می‌دهیم.
483)                for (int z = 0; z <= 9; z++) {
484)                    if (!strcmp(UDPClientInfo[z].host, host) &&
                                    !strcmp(UDPClientInfo[z].service,
service)) {
485)                        strncpy(&ConnectionInfo[z], "0", 1);
486)                        ClientCheck(-(z + 1), "");
487)                        Authenticate[z] = false;
488)                        strcpy(UDPClientInfo[z].host, "");
489)                        strcpy(UDPClientInfo[z].service, "");
490)                        strcpy(UDPClientInfo[z].UserName, "");
491)                        bzero(&UDPClientInfo[z].RemoteAddr, sizeof
                                        UDPClientInfo[z].RemoteAddr);
492)                        break;
493)                    }
494)                }
495) بعد از عملیات حذف، دیگر کاربران را از این موضوع آگاه می‌کنیم؛ بافر را با اطلاعات کاربران تنظیم کرده و ارسال می‌کنیم.
496)                //Send Update List*********************************************
497)                strcpy(buf, "%CONF%");
498)                strcpy(&buf[10], ConnectionInfo);
499)                for (int z = 1; z <= 10; z++) {
500)                    strcpy(&buf[20 * z], UDPClientInfo[z - 1].UserName);
501)                }
502)                Specialize(buf);
503)                Encrypt(buf, buf, EncryptionType);
504)                for (int i = 0; i <= 9; i++) {
505) آدرس متناظر با اندیس را به دست می‌آوریم تا در بتوانیم در پیغام مورد استفاده قرار دهیم.
506)                    inet_ntop(UDPClientInfo[i].RemoteAddr.ss_family, get_in_addr(
                                    (sockaddr*)
&UDPClientInfo[i].RemoteAddr),
            remoteIP, INET6_ADDRSTRLEN);
507) بررسی می‌کنیم که آیا در محدوده ۰ تا ۹ کاربری وجود دارد؟ در صورت پاسخ مثبت اطلاعات کاربران موجود را
```

Socket Class

```
508)                    if (strcmp(UDPClientInfo[i].host, "")) {
509)                        if ((nbytes = sendto(mySocketFD, buf, strlen(buf), 0,
(sockaddr *)
        &UDPClientInfo[i].RemoteAddr, sizeof
                            UDPClientInfo[i].RemoteAddr)) <= 0) {
510)                            sprintf(Message, Error on sending ClientList to %s
with
        %s,%s\n",remoteIP, UDPClientInfo[i].host,
                            UDPClientInfo[i].service);
511)                            easyLog("error");
512)                        } else {
513)                            sprintf(Message, "ClientList successfully sent to %s
with                                                %s,%s\n",remoteIP,
UDPClientInfo[i].host,
        UDPClientInfo[i].service);
514)                            easyLog("success");
515)                    }
516)                }
517)            }
518)            continue;بازگشت به حلقه اصلی
519)        } else {
520)  اگر به این قسمت رسیدیم یعنی داده‌ای دریافت کرده‌ایم، حال باید بررسی کنیم که ارسال کننده، معتبر است یا
     خیر. در صورت معتبر بودن، پیغام دریافتی را پخش می‌کنیم در غیر این صورت باید صحت اعتبارش بررسی شود.
521)            Decrypt(buf, buf, EncryptionType);
522)            unSpecialize(buf);
523)  اطلاعات کاربر ارسال کننده داده از جمله آدرس، نام ماشین و نام سرویس و نام کاربری را به دست می‌آوریم.
524)            inet_ntop(RemoteAddr.ss_family, get_in_addr( (sockaddr*)
&RemoteAddr),                                       remoteIP,
INET6_ADDRSTRLEN);
525)            getnameinfo((sockaddr *) &RemoteAddr, sizeof RemoteAddr, host,
sizeof host,                                        service, sizeof service, 0);
526)            strncpy(getClientUserName, &buf[10], 20);
527)            bool found = false;این متغیر بیانگر یافتن یا عدم آن خواهد بود
528)            for (int z = 0; z <= 9; z++) {
529)  برای یافتن کاربر کافیست که نام ماشین و نام سرویس را بررسی کنیم.
530)                if (!strcmp(UDPClientInfo[z].host, host) &&
                                                    !strcmp(UDPClientInfo[z].service,
service)) {
531)  یافته شد! پیغام مناسب را چاپ کردا و داده را پخش می‌کنیم.
532)                    found = true;
533)                    sprintf(Message, Receive Successful: %d bytes received from
%s with                                             %s,%s\n", nbytes,
remoteIP, host, service);
534)                    easyLog("success");
535)  کاربران منتخب را به دست آورده تا داده را برای آنها ارسال کنیم.
536)                    strncpy(sendConnectionInfo, buf, 10);
```

Socket Class

```
537)                    Specialize(buf);
538)                    strcpy(buf, &buf[10]);
539)                    Encrypt(buf, buf, EncryptionType);
540)                    for (int i = 0; i <= 9; i++) {
541)  آدرس کاربری که می‌خواهیم برای او ارسال کنیم را به دست می‌آوریم (مورد نیاز برای چاپ پیغام)
542)                        inet_ntop(UDPClientInfo[i].RemoteAddr.ss_family,
get_in_addr(                                    (sockaddr*)
&UDPClientInfo[i].RemoteAddr),
        remoteIP, INET6_ADDRSTRLEN);
543)  تست می‌کنیم که آیا کاربر با مشخصات مورد نظر طبق لیست منتخب موجود است یا خیر در صورت وجود اقدام به
      ارسال می‌کنیم.
544)                        if((strcmp(UDPClientInfo[i].host, "") &&
                                        !strncmp(&sendConnectionInfo[i],
"1", 1))){
545)                          if (strcmp(UDPClientInfo[i].host, host) ||
                                        strcmp(UDPClientInfo[i].service,
service)) {
546)                              if ((nbytes= sendto(mySocketFD, buf, strlen(buf), 0,
(sockaddr *)
        &UDPClientInfo[i].RemoteAddr, sizeof
                            UDPClientInfo[i].RemoteAddr)) <= 0) {
547)                                  sprintf(Message, "Send Failed: Sending data to %s
with                                        %s,%s failed\n",remoteIP,

        UDPClientInfo[i].host,
                            UDPClientInfo[i].service);
548)                                  easyLog("error");
549)                              } else {
550)                                  sprintf(Message, "Send Successful: %d bytes send
to %s with                                  %s,%s\n",nbytes,
remoteIP,
        UDPClientInfo[i].host,
                            UDPClientInfo[i].service);
551)                                  easyLog("success");
552)                              }
553)                          }
554)                      }
555)                  }
556)                  break;
557)              }
558)          }
559)  حالا اگر از کاربری داده‌ای دریافت کرده باشیم ولی در لیستمان موجود نباشد به این بخش می‌رسیم؛ کاری که باید
      انجام دهیم بررسی صحت اطلاعات او می‌باشد.
560)          if (!found) {
561)  نام کاربری و رمز عبور را به دست می‌آوریم.
562)              strncpy(getClientUserName, buf, 20);
563)              strncpy(getClientPass, &buf[20], 15);
564)              sprintf(Message, "Connection: New connection as %s from %s
```

Socket Class

```
                with                                    %s,%s\n", getClientUserName,
remoteIP, host, service);
565)                    easyLog("bold");
566)                                                    رمز عبور را بررسی می‌کنیم.
567)                    if (!strcmp(getClientPass, Password)) {
568)                                                    اگر وارد این بخش شدیم به این معنیست که رمز عبور معتبر است. فرایند اضافه کردن کاربر به لیست را انجام
                                                        می‌دهیم. سپس دیگر کاربران را مطلع می‌کنیم.
569)                        Authenticate[cntConn] = true;
570)                        sprintf(Message, "Login Successful: %s from %s with
%s,%s\n",                                               getClientUserName,
remoteIP, host, service);
571)                        easyLog("success");
572)                                                    نام ماشین، نام سرویسی که ماشین بر روی آن مرتبط شده است را اضافه می‌کنیم
573)                        strcpy(UDPClientInfo[cntConn].host, host);
574)                        strcpy(UDPClientInfo[cntConn].service, service);
575)                                                    نام کاربری را نیز اضافه می‌کنیم
576)                        strcpy(UDPClientInfo[cntConn].UserName,
getClientUserName);
577)                                                    اطلاعات مربوط به آدرس را ذخیره می‌کنیم
578)                        UDPClientInfo[cntConn].RemoteAddr = RemoteAddr;
579)                                                    اندیس مربوطه را فعال کرده و GtkCheckButton متناظر را با نام کاربری در حالت انتخاب قرار می‌دهیم.
580)                        strncpy(&ConnectionInfo[cntConn], "1", 1);
581)                        ClientCheck(cntConn + 1, getClientUserName);
582)                                                    نقش این متغیر علاوه بر نگه داشتن تعداد کاربران تا سقف ۱۰ تا، با مقداردهی انجام شده بدین صورت عمل می‌کند
                                                        که همیشه کاربری که از همه دیرتر وارد شده است را در صورت پر شده حد نساب به کاربر جدید مقداردهی
                                                        می‌کند، بازه تغییرات متغیر زیر از 0 تا9 می‌باشد.
583)                        cntConn = (cntConn + 1) % 10;
584)                                                    در ادامه دیگر کاربران را از ورود کاربر جدید مطلع می‌کنیم؛ لیست اطلاعات کاربران را با یک پیغام کنترلی تنظیم
                                                        کرده و ارسال می‌کنیم.
585)                        strcpy(buf, "%CONF%");                  متن کنترلی
586)                        strcpy(&buf[10], ConnectionInfo);  کپی اندیس کاربران موجود
587)                        for (int z = 1; z <= 10; z++) {         کپی نام کاربری کاربران
588)                            strcpy(&buf[20 * z], UDPClientInfo[z - 1].UserName);
589)                        }
590)                        Specialize(buf);                        تنظیم کاراکترهای آخر
                        رشته
591)                        Encrypt(buf, buf, EncryptionType);رمزگزاری داده‌ها
592)                        for (int i = 0; i <= 9; i++) {
593)                                                    ابتدا آدرس ماشینی را که می‌خواهیم داده‌ای برای آن ارسال کنیم را به دست می‌آوریم.
594)                            inet_ntop(UDPClientInfo[i].RemoteAddr.ss_family,
get_in_addr(                                            (sockaddr*)
&UDPClientInfo[i].RemoteAddr),
                    remoteIP, INET6_ADDRSTRLEN);
595)                                                    بررسی می‌کنیم که در اندیس ساختمان داده اطلاعات کاربران، کاربری موجود می‌باشد یا خیر؟
```

Socket Class

```
596)                            if (strcmp(UDPClientInfo[i].host, "")) {
597)                                                  در صورت وجود ارسال می‌کنیم
598)                                if ((nbytes= sendto(mySocketFD, buf, strlen(buf), 0,
(sockaddr *)
        &UDPClientInfo[i].RemoteAddr, sizeof
                                UDPClientInfo[i].RemoteAddr)) <= 0) {
599)                                    sprintf(Message, "Error on sending ClientList to %s
with                                                    %s,%s\n",remoteIP,
UDPClientInfo[i].host,
        UDPClientInfo[i].service);
600)                                    easyLog("error");
601)                                } else {
602)                                    sprintf(Message, "ClientList successfully sent to %s
with                                                    %s,%s\n",remoteIP,
UDPClientInfo[i].host,
        UDPClientInfo[i].service);
603)                                    easyLog("success");
604)                                }
605)                            }
606)                        }
607)                        found = false;
608)                    } else {
609)                                        اگر رمز عبور نامعتبر بود کاربر را با یک پیغام کنترلی آگاه می‌سازیم.
610)                        sprintf(Message, "Login Failed: %s from %s with
                                                %s,%s\n",getClientUserName,
remoteIP, host, service);
611)                        easyLog("error");
612)                        strcpy(buf,"%LOGIN_FAILED%");
613)                        strcpy(buf,md5(buf).c_str());
614)                        sendto(mySocketFD, buf, sizeof buf, 0, (sockaddr *)
&RemoteAddr, sizeof                                     RemoteAddr);
615)                    }
616)                }// Not Found
617)            } //else receive
618)        }//for
619)    }
620) هنگاهی به این بخش می‌رسیم که Manager دچار مشکل شده باشد، بنابراین کاربران را از GtkCheckButtonها پاک می‌کنیم.
621)    ClientCheckUpdate("00000000000000000000");
622) }// Server Manager
623)
624) برای قطع کردن ارتباط خاصی از این تابع استفاده می‌شود؛ کاربرد آن فقط در Server بوده و عملکرد آن بدینگونه
است که ابتدا کاربرانی که انتخاب شده‌اند را تحت رشته کاراکتری به طول ۱۰ برمی‌گرداند سپس بر اساس
اندیسهایی که دارا مقدار '1' هستند، کاربر متناظرشان را قطع می‌کند.
625) void Socket::DisconnectClients() {
626)    int i = 0;
627)    char users[10];        متغیر برای نگهداری اندیسها
```

Socket Class

```
628)     strcpy(users, getClientCheck());      به دست آوردن اندیسها
629)     while (i <= 9) {
630)         if (!strncmp(&users[i], "1", 1)) {
631)                                   اگر وارد این قستم شدیم یعنی کاربر برای قطع ارتباط انتخاب شده است.
632)             int nbytes;
633)             char sendBuffer[256] = "";
634)                                   متن کنترلی برای قطع ارتباط را تنظیم کرده و md5 می‌کنیم.
635)             strcpy(sendBuffer, "%DISCONNECT%");
636)             strcpy(sendBuffer, md5(sendBuffer).c_str());
637)                                   حال بر اساس نوع ارتباط (TCP یا UDP) اقدام به ارسال می‌کنیم
638)             if (Type == SOCK_STREAM) {
639)                 if ((nbytes = send(i + mySocketFD + 1, sendBuffer,
640)                         sizeof sendBuffer, 0)) == -1) {
641)                                   ارسال متن کنترلی با مشکل مواجه شد
642)                     sprintf(Message, "Connection: Error on sending DISCONNECT
command to                           %s on socket
%d!\n",ListenAddress, i + mySocketFD + 1);
643)                     easyLog("error");
644)                 } else {
645)                                   ارسال متن کنترلی با موفقیت آمیز بود
646)                     sprintf(Message, "Connection: Connection on socket %d
closed\n", i +                       mySocketFD + 1);
647)                     easyLog("success");
648)                 }
649)             } else if (Type == SOCK_DGRAM) {
650)                 char remoteIP[INET6_ADDRSTRLEN];
651)                 inet_ntop(UDPClientInfo[i].RemoteAddr.ss_family, get_in_addr(
(sockaddr*)                          &UDPClientInfo[i].RemoteAddr),
remoteIP,
        INET6_ADDRSTRLEN);
652)                 if ((nbytes = sendto(mySocketFD, sendBuffer, sizeof sendBuffer, 0,
(sockaddr *)                         &UDPClientInfo[i].RemoteAddr,
sizeof
        UDPClientInfo[i].RemoteAddr)) <= 0) {
653)                                   ارسال متن کنترلی با مشکل مواجه شد
654)                     sprintf(Message, "Connection: Error on sending DISCONNECT
command to                           %s with %s,%s\n",remoteIP,
UDPClientInfo[i].host,
        UDPClientInfo[i].service);
655)                     easyLog("error");
656)                 } else {
657)                                   ارسال متن کنترلی با موفقیت آمیز بود
658)                     sprintf(Message, "Connection: Connection %s with %s,%s
                                     closed\n",remoteIP,
UDPClientInfo[i].host,
        UDPClientInfo[i].service);
659)                     easyLog("success");
660)                 }
```

Socket Class

```
661)                }
662)            }
663)            i++;
664)        }
665)    }
666)
667)                    این متد در Client مورد استفاده قرار گرفته و هنگام متصل شدن به Login Server می‌کند.
668)    int Socket::Login() {
669)        char sendBuffer[256];
670)        int nbytes;
671)                    نام کاربری و رمز عبور برای ارسال تنظیم می‌شوند.
672)        strcpy(sendBuffer, UserName);
673)        strcpy(&sendBuffer[20], Password);
674)        Specialize(sendBuffer);        تنظیم کاراکتر آخر رشته‌ها
675)        Encrypt(sendBuffer, sendBuffer, EncryptionType);            رمزنگاری داده
676)        if (Type == SOCK_STREAM) {ارسال در حالت ارتباط اتصال‌گرا
677)            if ((nbytes = send(ConnectionFD, sendBuffer, sizeof sendBuffer, 0))
678)                    == -1) {    ارسال ناموفق
679)                sprintf(Message, "Login: Login failed due to sending data.\n");
680)                easyLog("error");
681)            } else {    ارسال موفق
682)                sprintf(Message, "Login: Login data successfully sent\n");
683)                easyLog("success");
684)            }
685)        } else if (Type == SOCK_DGRAM) {ارسال در حالت ارتباط بدون‌اتصال
686)            if ((nbytes = sendto(ConnectionFD, sendBuffer, sizeof sendBuffer, 0,
    (sockaddr *)                &RemoteAddr, sizeof RemoteAddr)) <= 0) { ارسال
    ناموفق
687)                perror("sendto");
688)                sprintf(Message, "Login: Login failed due to sending data.\n");
689)                easyLog("error");
690)            } else {ارسال موفق
691)                sprintf(Message, "Login: Login data successfully sent\n");
692)                easyLog("success");
693)            }
694)        }
695)        return 0;
696)    }
697)
698)    این متد نیز به عنوال قلب Client محسوب می‌شود، این متد در پشت پرده اجرا می‌شود و هنگام دریافت اطلاعات
    آنها را بررسی کرده و عملیات مناسب را انجام می‌دهد.
699)    void Socket::ClientReceiverMode() {
700)        char recBuffer[256];
701)        int nbytes;
702)        char remoteIP[INET6_ADDRSTRLEN];
703)        char host[1024];
```

Socket Class

```
704)    char service[20];
705)    socklen_t addr_len = sizeof RemoteAddr;
706)    if (Type == SOCK_STREAM) {دریافت کردن در حالت ارتباط اتصال‌گرا
707)        برای بررسی اینکه داده‌ای دریافت شده است فقط مشخصه سوکت خودمان مورد استفاده قرار خواهد گرفت.
708)        FD_ZERO(&Master_FDs);
709)        FD_SET(ConnectionFD,&Master_FDs);
710)        for (;;) {
711)            strcpy(recBuffer, "");
712)            Read_FDs = Master_FDs; // copy it
713)  در اینجا نیز بررسی می‌کنیم که آیا نرم‌افزار در حالت بلوکه شدن است یا بدون بلوکه شدن؟ عملیات مناسب را انجام
      می‌دهیم. در صورتیکه نرم‌افزار در حالت بلوکه شدن نباشد یعنی دستور recv یا recvfrom در هنگام دریافت
      داده منتظر آن نمی‌مانند، فقط بررسی می‌کنند که آیا داده‌ای موجود است یا خیر؛ اگر نبود مقدار بازگشتی time-
      out را برمی‌گرداند. اما اگر در حالت بلوکه شدن باشد تا دریافت داده‌ای منتظر می‌مانند.
714)            if (NonBlockingMode) {
715)  وارد حالت بلوکه نشدن می‌شویم، پس دستور select را با زمان انتظار برای جلوگیری از اتلاف وقت CPU اجرا
      می‌کنیم.
716)                TWait.tv_sec = 0;
717)                TWait.tv_usec = 750000;
718)                if (select(ConnectionFD + 1, &Read_FDs, NULL, NULL, &TWait)
719)                        == -1) {
720)                    sprintf(Message,
721)                            "Internal Client Error!: Error on socket.\n");
722)                    easyLog("error");
723)                    if (AppStat)
724)                        FlushAll();
725)                    return;
726)                }
727)            } else {
728)  این قسمت مربوط به نرم‌افزار در حالت بوکه شدن است، پس دستور select بدون زمان انتظار اجرا می‌شود.
729)                if (select(ConnectionFD + 1, &Read_FDs, NULL, NULL, NULL) == -1) {
730)                    sprintf(Message,
731)                            "Internal Client Error!: Error on socket.\n");
732)                    easyLog("error");
733)                    if (AppStat)
734)                        FlushAll();
735)                    return;
736)                }
737)            }
738)  در اینجا بررسی می‌کنیم که مشخصه سوکت ما داده‌ای برای دریافت دارد یا خیر؛ زیرا ممکن است time-out نیز
      دخیل باشد.
739)            if (FD_ISSET(ConnectionFD, &Read_FDs)) {
740)                if ((nbytes
741)                        = recv(ConnectionFD, recBuffer, sizeof recBuffer, 0))
742)                        <= 0) {
743)  طول داده‌های دریافتی را بررسی می‌کنیم، اگر برار 0 بود، یعنی Server ارتباط را بر روی ما بسته است، عملیات
      لازم را انجام می‌دهیم.
```

Socket Class

```
744)                    if (nbytes == 0) {
745)                        sprintf(Message, "Connection Lost: server hung up\n");
746)                        easyLog("lost");
747)                        shutdown(ConnectionFD, SHUT_RD);
748)                        shutdown(ConnectionFD, SHUT_WR);
749)                        close(ConnectionFD);
750)                        FD_CLR(ConnectionFD, &Master_FDs);
751)                        FD_ZERO(&Master_FDs);
752)                        ClientCheckUpdate("00000000000000000000");
753)                        if (AppStat)
754)                            FlushAll();
755)                        return;
756)                    } else {زمانیکه داده‌ها با خطا مواجه شده باشند
757)                        sprintf(Message,
758)                                "Receive Error: cannot get data from server\n");
759)                        easyLog("error");
760)                    }
761)                } else {
762) اگر دریاف داده‌ها موفقیت آمیز باشد ابتدا بررسی می‌کنیم که آیا پیغام کنترلی قطع ارتباط از طرف Server است؟
     عملیات مناسب را در صورت صحیح بودن انجام می‌دهیم.
763)                    if (!strcmp(recBuffer, md5("%DISCONNECT%").c_str())) {
764)                        sprintf(Message, "\t\t***Disconnected***\n");
765)                        easyLog("bold");
766)                        shutdown(ConnectionFD, SHUT_WR);
767)                        shutdown(ConnectionFD, SHUT_RD);
768)                        close(ConnectionFD);
769)                        FD_CLR(ConnectionFD, &Master_FDs);
770)                        FD_ZERO(&Master_FDs);
771)                        ClientCheckUpdate("00000000000000000000");
772)                        if (AppStat)
773)                            FlushAll();
774)                        return;
775) درصورتیکه به این بخش رسیدیم بررسی می‌کنیم که آیا پیغام کنترلی Login ناموفق از طرف Server ارسال
     شده است؟ عملیات مناسب را در صورت صحیح بودن انجام می‌دهیم.
776)                    } else if (!strcmp(recBuffer, md5("%LOGIN_FAILED%").c_str())) {
777)                        sprintf(Message, "\t\t***Login Failed***\n");
778)                        easyLog("bold");
779)                        shutdown(ConnectionFD, SHUT_WR);
780)                        shutdown(ConnectionFD, SHUT_RD);
781)                        close(ConnectionFD);
782)                        FD_CLR(ConnectionFD, &Master_FDs);
783)                        FD_ZERO(&Master_FDs);
784)                        ClientCheckUpdate("00000000000000000000");
785)                        if (AppStat)
786)                            FlushAll();
787)                        return;
788)                    }
789) در این بخش بررسی می‌کنیم که آیا داده‌های دریافتی حاوی اطلاعات برای به روز رسانی کاربران است یا پیغام
```

Socket Class

```
                دریافت شده است؟ متن را رمزگشایی کرده آنرا تنظیم کرده و عملیات مناسب را انجام می‌دهیم.
790)                Decrypt(recBuffer, recBuffer, EncryptionType);
791)                unSpecialize(recBuffer);
792)                if (!strcmp(recBuffer, "%CONF%")) {بررسی در صورت داده‌های به روز رسانی
793)                    sprintf(Message, "\t\t***Users updated***\n");
794)                    easyLog("bold");
795)                    ClientCheckUpdate(recBuffer);
796)                    continue;
797)                } else {آخرین بخش است، پیغام دریافتی را چاپ می‌کنیم}
798)                    sprintf(Message, "%s: %s\n", recBuffer, &recBuffer[20]);
799)                    easyLog("");
800)                }
801)            }
802)        }
803)    }
804)} else if (Type == SOCK_DGRAM) {
805)    for (;;) {
806)        strcpy(recBuffer, "");
807)        if (NonBlockingMode) {
808)در حالت بلوکه نشدن نیاز به دستور select مشهود است و فقط برای اعمال زمان انتظار مورد استفاده قرار می‌گید.
809)            TWait.tv_sec = 0;
810)            TWait.tv_usec = 750000;
811)            select(NULL, NULL, NULL, NULL, &TWait);
812)        }
813)داده‌ها را دریافت می‌کنیم، بررسی خطا را انجام می‌دهیم.
814)        if ((nbytes = recvfrom(ConnectionFD, recBuffer, sizeof recBuffer,
815)                0, (sockaddr *) &RemoteAddr, &addr_len)) <= 0) {
816)            inet_ntop(RemoteAddr.ss_family, get_in_addr(
817)                    (sockaddr*) &RemoteAddr), remoteIP, INET6_ADDRSTRLEN);
818)            getnameinfo((sockaddr *) &RemoteAddr, sizeof RemoteAddr, host,
819)                    sizeof host, service, sizeof service, 0);
820)            if (NonBlockingMode) {
821)در صورتیکه در حالت بلوکه نشدن باشیم نمی‌توان گفت که خطا در هنگام عدم دریافت داده‌ها رخ داده یا به دلیل
                time-out به همین خاطر این بخش را بدون دستور رها می‌کنیم.
822)            } else {
823)                perror("recvfrom");
824)                sprintf(
825)                        Message,
826)                        "Receive Error: cannot get data from %s with %s,%s\n",
827)                        remoteIP, host, service);
828)                easyLog("error");
829)            }
830)        } else {
831)با رسیدن به این بخش یعنی داده‌ها با دریافت موفق روبه‌رو شدند
832)            inet_ntop(RemoteAddr.ss_family, get_in_addr(
833)                    (sockaddr*) &RemoteAddr), remoteIP, INET6_ADDRSTRLEN);
834)            getnameinfo((sockaddr *) &RemoteAddr, sizeof RemoteAddr, host,
835)                    sizeof host, service, sizeof service, 0);
```

Socket Class

```
836)  ابتدا بررسی می‌کنیم که آیا متن کنترلی مبنی بر قطع ارتباط از سوی Server دریافت شده است یا متن کنترلی
      مربوط به عملیات Login ناموفق بوده است؟ اگر در هر دو حالت جواب خیر بود بنابراین متن را تنظیم و رمز
      گشایی کرده و بررسی می‌کنیم که آیا اطلاعات مربوط به کاربران دریافت شده و یا خود پیغام است؟
837)                if (!strcmp(recBuffer, md5("%DISCONNECT%").c_str())) {
838)  این بخش مربوط به متن کنترلی قطع ارتباط است.
839)                    sprintf(Message, "\t\t***Disconnected***\n");
840)                    easyLog("bold");
841)                    char sendBuffer[256] = "";
842)  به دلیل ارتباط بدون اتصال با متن کنترلی به Server اعلام می‌کنیم که قطع رابطه کردیم.
843)                    strcpy(sendBuffer, "%DISCONNECTED%");
844)                    strcpy(sendBuffer, md5(sendBuffer).c_str());
845)                    sendto(ConnectionFD, sendBuffer, sizeof sendBuffer, 0,
846)                           (sockaddr *) &RemoteAddr, sizeof RemoteAddr);
847)                    shutdown(ConnectionFD, SHUT_RDWR);
848)                    close(ConnectionFD);
849)                    ClientCheckUpdate("00000000000000000000");
850)                    if (AppStat)
851)                        FlushAll();
852)                    return;
853)                } else if (!strcmp(recBuffer, md5("%LOGIN_FAILED%").c_str())) {
854)  این بخش مربوط به عملیات ناموفق Login است، عملیات لازم جهت reset کردن را انجام می‌دهیم.
855)                    sprintf(Message, "\t\t***Login Failed***\n");
856)                    easyLog("bold");
857)                    shutdown(ConnectionFD, SHUT_RDWR);
858)                    close(ConnectionFD);
859)                    ClientCheckUpdate("00000000000000000000");
860)                    if (AppStat)
861)                        FlushAll();
862)                    return;
863)                }
864)  این بخش مربوط به لیست کاربران یا پیغام است.
865)                Decrypt(recBuffer, recBuffer, EncryptionType);
866)                unSpecialize(recBuffer);
867)                if (!strcmp(recBuffer, "%CONF%")) {
868)  اگر وارد این بخش شدیم یعنی لست کاربران دریافت شده است. عملیات به روز رسانی را انجام می‌دهیم.
869)                    sprintf(Message, "\t\t***Users updated***\n");
870)                    easyLog("bold");
871)                    ClientCheckUpdate(recBuffer);
872)                    continue;
873)                } else {
874)  بخش مربوط به چاپ پیغام دریافتی از Server است.
875)                    sprintf(Message, "%s: %s\n", recBuffer, &recBuffer[20]);
876)                    easyLog("");
877)                }
878)            }
879)        }
880)    }
```

Socket Class

```
881)  }
882)  متدیست که فقط با فشردن کلید Send در Client اجرا می‌شود، هدف آن ارسال متن درون جعبه ClientSnedText به سرور است.
883)  void Socket::ClientSendMode() {
884)      int nbytes;
885)      char sendBuffer[256] = "";
886)  دستور زیر لیست کاربرانی را بر می‌گرداند که مایل به ارسال داده به آنها هستیم، با انتخاب کردن آنها از میان ۱۰ GtkCheckButton این تابع مقدار بازگشتی یک رشته کاراکتری به طول ۱۰ را که بیانگر این انتخابها می‌باشد را انجام می‌دهد. این رشته را در بافر ارسالی قرار می‌دهیم.
887)      strcpy(sendBuffer, getClientCheck());
888)  نام کاربری خود را اضافه می‌کنیم. (درست بعد از رشته انتخابها)
889)      strcpy(&sendBuffer[10], UserName);
890)  متن موجود در جعبه متن ClientSendText را در بافر ارسالی قرار می‌دهیم. (درست بعد از نام کاربری)
891)      strcpy(&sendBuffer[30], BufferGet());
892)  کاراکتر آخر رشته را تنظیم می‌کنیم تا در عملیات رمز نگاری دچار مشکل نشویم.
893)      Specialize(sendBuffer);
894)  رمزنگاری را انجام می‌دهیم.
895)      Encrypt(sendBuffer, sendBuffer, EncryptionType);
896)  حال بسته به نوع ارتباط (TCP یا UDP) اقدام به ارسال پیغام می‌نماییم.
897)      if (Type == SOCK_STREAM) {
898)          if ((nbytes = send(ConnectionFD, sendBuffer, sizeof sendBuffer, 0))
899)                  == -1) {ارسال ناموفق
900)              sprintf(Message, "Send Error: %d bytes of data sent to %s!\n",
901)                      nbytes, ListenAddress);
902)              easyLog("error");
903)          } else {ارسال موفق
904)  عملیات زیر برای چاپ پیغام ارسالیام در جعبه متن خودمان می‌باشد.
905)              Decrypt(sendBuffer, sendBuffer, EncryptionType);
906)              unSpecialize(sendBuffer);
907)              sprintf(Message, "Me: %s\n", &sendBuffer[30]);
908)              easyLog("");
909)          }
910)          gtk_widget_grab_focus(GW("ClientSendText"));
911)      } else if (Type == SOCK_DGRAM) {
912)          if ((nbytes = sendto(ConnectionFD, sendBuffer, strlen(sendBuffer), 0,
913)                  (sockaddr *) &RemoteAddr, sizeof RemoteAddr)) <= 0) {ارسال ناموفق
914)              perror("sendto");
915)              sprintf(Message, "Send Error: %d bytes of data sent to %s!\n",
916)                      nbytes, ListenAddress);
917)              easyLog("error");
918)          } else {ارسال موفق
919)              Decrypt(sendBuffer, sendBuffer, EncryptionType);
920)              unSpecialize(sendBuffer);
921)              sprintf(Message, "Me: %s\n", &sendBuffer[30]);
922)              easyLog("");
923)          }
```

Socket Class

```
924)     }
925) }
926) کاربرد این تابع بدین صورت است که، درصورتیکه به هنگام ارسال داده‌ها، طول داده‌های ارسالی نسبت به طول داده
     مورد نظر برای ارسال کمتر باشد، اقدام به ارسال ادامه داده‌ها می‌نماییم. نکته اینکه مقدار بازگشتی تابع send
     تعداد کاراکترهای ارسالی است.
927) int sendall(int s, char *buf, int *len) {
928)     int total = 0; // how many bytes we've sent
929)     int bytesleft = *len; // how many we have left to send
930)     int n;
931)     while (total < *len) {
932)         n = send(s, buf + total, bytesleft, 0);
933)         if (n == -1) {
934)             break;
935)         }
936)         total += n;
937)         bytesleft -= n;
938)     }
939)     *len = total; // return number actually sent here
940)     return n == -1 ? -1 : 0; // return -1 on failure, 0 on success
941) }
942)         این تابع طوری عمل می‌کند که بر اساس نوع نسخه IP (IPv4 یا IPv6)، اطلاعات آدرسی را برمی‌گرداند.
943) void *get_in_addr(sockaddr *sa) {
944)     return &(((sockaddr_in*) sa)->sin_addr);
945) }
```

TCPSocket.h –۸–۴–۴

TCPSocket Class Header

```
1)  /*
2)  * TCPSocket.h
3)  *
4)  *  Created on: Jul 5, 2011
5)  *      Author: ben
6)  */
7)
8)  #ifndef TCPSOCKET_H_
9)  #define TCPSOCKET_H_
10)
11) #include "Socket.h"
12)
13)     این کلاس برای تعریف ارتباط اتصال‌گرا به وجود آمده است، از کلاس Socket به صورت Public و Virtual
        مشتق می‌شود.
```

TCPSocket Class Header

```
14) class TCPSocket: public virtual Socket {
15) private:
16)     int GetConnect(addrinfo*);
17) 
18) protected:
19)     int CreateTCPServer();
20)     int CreateTCPClientReceiverMode();
21) 
22) public:
23)     TCPSocket();
24)     virtual ~TCPSocket();
25) };
26) 
27) #endif /* TCPSOCKET_H_ */
```

TCPSocket.cpp –۹–۴–۴

TCPSocket Class

```
1)  /*
2)   * TCPSocket.cpp
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)  
8)  #include "TCPSocket.h"
9)  
10) TCPSocket::TCPSocket() {
11)     // TODO Auto-generated constructor stub
12)     cout << "Constructing TCP Socket...\n";
13) }
14) 
15) TCPSocket::~TCPSocket() {
16)     // TODO Auto-generated destructor stub
17)     cout << "Destructing TCP Socket...\n";
18) }
19) 
20)                                 عملیات راه‌اندازی server به صورت زیر انجام می‌گیرد:
21) int TCPSocket::CreateTCPServer() {
22)     sprintf(Message, "Server: Initiating TCP Server...\n");
23)     easyLog("bold");
24)     addrinfo *ai;
25)                                 در صورت فعال بودن برنامه Reset کردن سرور
26)     if (AppStat)
```

TCPSocket Class

```
27)         FlushAll();
28)                                   به دست آوردن اطلاعات آدرسی deviceها
29)     if ((ai = GetAddrInfo()) == NULL)
30)         return 1; //Error On Getting Address Information
31)                                   مقید کردن سوکت بر اساس آدرسهای به دست آمده به پورت تعیین شده
32)     if ((mySocketFD = GetBind(ai)) == -1)
33)         return 2; //Error On Binding Socket
34)                                   تعیین حالت سوکت (بلوکه بودن یا بلوکه نبودن)
35)     if (SocketBlockingMode())
36)         return 3;
37)                                   گوش دادن سوکت بر روی آدرس و پورت مورد نظر برای دریافت درخواستهای جدید
38)     if (ListenOn() == -1)
39)         return 4; //Error On Listener
40)     sprintf(Message, "Server Ready: TCP Connection Management started...\n");
41)     easyLog("bold");
42)     AppStat = true;
43)                                   اجرای مدیر ارتباطات
44)     ManageConnections();
45)     return 0;
46) }
47)
48)                                   این متد در حالت Client مورد استفاده قرار می‌گیرد و هدف آن برقراری ارتباط با Server است.
49) int TCPSocket::GetConnect(addrinfo* ai) {
50)     // loop through all the results and connect to the first we can
51)                                   ابتدا سوکتی را ساخته در صورت برقراری ارتباط خارج می‌شویم. این سوکت طبق آدرسهای به دست آمده از
                                      GetAddrInfo اقدام به اتصال می‌کند. در صورت موفقیت آمیز بودن، حالت سوکت را (بلوکه شدن) تعیین
                                      می‌کنیم.
52)     addrinfo *p;
53)     for (p = ai; p != NULL; p = p->ai_next) {
54)         if ((ConnectionFD
55)                 = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
56)             perror("Client: socket");
57)             continue;
58)         }
59)                                   اقدام به برقراری اتصال
60)         if (connect(ConnectionFD, p->ai_addr, p->ai_addrlen) == -1) {
61)             close(ConnectionFD);
62)             perror("Client: connect");
63)             continue;
64)         }
65)         break;
66)     }
67)     if (ConnectionFD < 0) {
68)         sprintf(Message, "Socket: Error on creating socket\n");
69)         easyLog("error");
70)     } else {
71)         sprintf(Message, "Socket: Successfully created with Num. %d\n",
```

TCPSocket Class

```
72)                 ConnectionFD);
73)             easyLog("success");
74)         }
75)         if (p == NULL) {
76)             sprintf(Message, "Client: Failed to connect to %s on socket %d\n",
77)                     ListenAddress, ConnectionFD);
78)             easyLog("error");
79)             return 1;
80)         }
81)                                                   تعیین حالت بلوکه شدن
82)         if (SocketBlockingMode())
83)             return 2;
84)         sprintf(Message, "Client: Successfully connected to %s on socket %d\n",
85)                 ListenAddress, ConnectionFD);
86)         easyLog("success");
87)         freeaddrinfo(p); // all done with this structure
88)         return 0;
89)     }
90)                                                   عملیات راه‌اندازی Client نیز به صورت زیر است:
91)     int TCPSocket::CreateTCPClientReceiverMode() {
92)         sprintf(Message, "Client: Initiating TCP Client...\n");
93)         easyLog("bold");
94)         addrinfo *ai;
95)                                                   اگر برنامه در حالت اجرا باشد آنرا reset می‌کنیم.
96)         if (AppStat)
97)             FlushAll();
98)                                                   اطلاعات آدرسی server را به دست می‌آوریم
99)         if ((ai = GetAddrInfo()) == NULL)
100)            return 1; //Error On Getting Address Information
101)                                                  بر اساس اطلاعات به دست آمده در مرحله قبل اقدام به برقراری ارتباط می‌کنیم
102)        if (GetConnect(ai))
103)            return 2; //Error On Connectiong To Server
104)                                                  اگر برقراری ارتباط موفقیت آمیز بود اطلاعات Login را ارسال می‌کنیم
105)        if (Login())
106)            return 3;
107)        sprintf(Message, "Client Ready: TCP Receiver Mode started...\n");
108)        easyLog("bold");
109)        AppStat = true;
110)                                                  مدیر دریافت پیغامها را اجرا می‌کنیم.
111)        ClientReceiverMode();
112)        return 0;
113)    }
```

UDPSocket.h –۱۰–۴–۴

UDPSocket Class Header

```
1)  /*
2)   * UDPSocket.h
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)
8)  #ifndef UDPSOCKET_H_
9)  #define UDPSOCKET_H_
10)
11) #include "Socket.h"
12)
13)   این کلاس برای تعریف ارتباط بدون اتصال به وجود آمده است، از کلاس Socket به صورت Public و Virtual
      مشتق می‌شود.
14) class UDPSocket: public virtual Socket {
15) protected:
16)     int CreateUDPServer();
17)     int CreateUDPClientReceiverMode();
18)
19) public:
20)     UDPSocket();
21)     virtual ~UDPSocket();
22) };
23)
24) #endif /* UDPSOCKET_H_ */
```

UDPSocket.cpp –۱۱–۴–۴

UDPSocket CLass

```
1)  /*
2)   * UDPSocket.cpp
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)
8)  #include "UDPSocket.h"
9)  void *get_in_addr(sockaddr*); // get sockaddr: IPv4 or IPv6:
10) UDPSocket::UDPSocket() {
11)     // TODO Auto-generated constructor stub
12)     cout << "Constructing UDP Socket...\n";
13) }
```

UDPSocket CLass

```
14)
15) UDPSocket::~UDPSocket() {
16)     // TODO Auto-generated destructor stub
17)     cout << "Destructing UDP Socket...\n";
18) }
19)                                  عملیات راه‌اندازی server به صورت زیر انجام می‌گیرد:
20) int UDPSocket::CreateUDPServer() {
21)     sprintf(Message, "Server: Initiating UDP Server...\n");
22)     easyLog("bold");
23)     struct addrinfo *ai;
24)                                  در صورت فعال بودن برنامه Reset کردن سرور
25)     if (AppStat)
26)         FlushAll();
27)                                  به دست آوردن اطلاعات آدرسی deviceها
28)     if ((ai = GetAddrInfo()) == NULL)
29)         return 1; //Error On Getting Address Information
30)                                  مقید کردن سوکت بر اساس آدرسهای به دست آمده به پورت تعیین شده
31)     if ((mySocketFD = GetBind(ai)) == -1)
32)         return 2; //Error On Binding Socket
33)                                  تعیین حالت سوکت (بلوکه بودن یا بلوکه نبودن)
34)     if (SocketBlockingMode())
35)         return 3;
36) به دلیل بدون اتصال بودن ارتباط دیگر خبری از دستور ListenOn نیست، زیرا ارسال و دریافت بر روی اتصال
    انجام نیم‌شود در هر سری اقدام، اطلاعات آدرسی مورد استفاده قرار می‌گیرند.
37)     sprintf(Message, "Server Ready: UDP Connection Management started...\n");
38)     easyLog("bold");
39)     AppStat = true;
40)                                  اجرای مدیر ارتباطات
41)     ManageConnections();
42)     return 0;
43) }
44)
45)                                  عملیات راه‌اندازی Client نیز به صورت زیر است:
46) int UDPSocket::CreateUDPClientReceiverMode() {
47)     sprintf(Message, "Client: Initiating UDP Client...\n");
48)     easyLog("bold");
49)     struct addrinfo *ai;
50)                                  اگر برنامه در حالت اجرا باشد آنرا reset می‌کنیم.
51)     if (AppStat)
52)         FlushAll();
53)                                  اطلاعات آدرسی server را به دست می‌آوریم
54)     if ((ai = GetAddrInfo()) == NULL)
55)         return 1; //Error On Getting Address Information
56) بر اساس اطلاعات به دست آمده در مرحله قبل اقدام به ساخت سوکت می‌کنیم؛ دقت داشته باشید که نیازی به
    برقراری اتصال نیست.
57)     for (; ai != NULL; ai = ai->ai_next) {
```

UDPSocket CLass

```
58)             ConnectionFD = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
59)             if (ConnectionFD < 0) {
60)                 continue;
61)             } else {
62) حالت سوکت را تعیین کرده و در صورت موفقیت آمیز بودن ساخت سوکت طبق اظلاعات Server آنها را به
    ساختمان داده مناسب خود کپی می‌کنیم تا در بخشهای بعدی از جمله دریافت و ارسال داده‌ها مورد استفاده قرار
    دهیم.
63)                 if (SocketBlockingMode())
64)                     continue;
65)                 memcpy(&RemoteAddr, ai->ai_addr, sizeof RemoteAddr);
66)                 break;
67)             }
68)         }
69)         if (ConnectionFD < 0) {
70)             sprintf(Message, "Socket: Error on creating socket\n");
71)             easyLog("error");
72)             return 2;
73)         }
74) در این قسمت اقدام به Login کرده و ادامه می‌دهیم.
75)         if (Login())
76)             return 3;
77)         sprintf(Message, "Client Ready: UDP Receiver Mode started...\n");
78)         easyLog("bold");
79)         AppStat = true;
80) مدیر دریافت پیغامها را اجرا می‌کنیم
81)         ClientReceiverMode();
82)         return 0;
83)     }
```

Communications.h –۱۲–۴–۴

Communication Class Header

```
1)  /*
2)   * Communications.h
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)
8)  #ifndef COMMUNICATIONS_H_
9)  #define COMMUNICATIONS_H_
10)
11) #include "TCPSocket.h"
```

Communication Class Header

```
12) #include "UDPSocket.h"
13) کلاس جالبی است، آخرین لایه محسوب می‌شود. با این کلاس می‌توانید نوع برنامه، حالت اتصال، حالت سوکت و
    بسیاری دیگر از خصوصیات را تعیین کنید و روند اجرای برنامه را کاملا عوض کنید. متغیرهایی که از لایه‌های
    پایین‌تر در این کلاس به ارث برده شده‌اند، در این کلاس مقداردهی می‌شوند. در حقیق این کلاس رابط بین کاربر و
    پیاده‌سازی نحوه اجرای توابع شبکه است. (طرح جالبی را انداختم؛ از این کلاس خوشم میاد)
14) class Communications: public virtual TCPSocket, public virtual UDPSocket {
15) private:
16)     char CommunicationType[5];    متغیری برای نگهداری نوع اتصال
17)
18)     void ConfigureFlags(int, int, int, int);
19)
20) public:
21)     Communications();
22)     virtual ~Communications();
23)
24)     void init(const char*, const char*, const char*, const char*, const char*,
25)             const char*, const bool, const int);
26)     void Run();
27)     void GetMachineIPs();
28) };
29)
30) #endif /* COMMUNICATIONS_H_ */
```

Communications.cpp –۱۳–۴–۴

Communication Class

```
1)  /*
2)   * Communications.cpp
3)   *
4)   *  Created on: Jul 5, 2011
5)   *      Author: ben
6)   */
7)
8)  #include "Communications.h"
9)
10) Communications::Communications() {
11)     // TODO Auto-generated constructor stub
12)     cout << "Constructing Communication...\n";
13) }
14)         این متد تمام متغیرهای مورد نیاز را مقداردهی می‌کند، این مقدارها از واسط گرافیکی کاربر تهیه می‌شوند.
15) void Communications::init(const char* prgMode, const char* comType,
16)         const char* IPver, const char* comIP, const char* comPort,
17)         const char* comPassword, const bool comNonBlock,
18)         const int comEncryption) {
```

Communication Class

```
19)     if (AppStat)
20)         FlushAll();
21)     //Program Mode**********************************
22)     strcpy(ProgramMode, prgMode);
23)     //Socket Type***********************************
24)     if (!strcmp(IPver, "IPv4")) {
25)         Domain = AF_INET;
26)         ConfigureFlags(AF_INET, -1, 0, AI_PASSIVE);
27)     } else if (!strcmp(IPver, "IPv6")) {
28)         Domain = AF_INET6;
29)         ConfigureFlags(AF_INET6, -1, 0, AI_PASSIVE);
30)     } else {
31)         Domain = AF_UNSPEC;
32)         ConfigureFlags(AF_UNSPEC, -1, 0, AI_PASSIVE);
33)     }
34)     strcpy(IPversion, IPver);
35)     //Connection Type*******************************
36)     strcpy(CommunicationType, comType);
37)     if (!strcmp(comType, "TCP")) {
38)         ConfigureFlags(-1, SOCK_STREAM, 0, AI_PASSIVE);
39)     } else if (!strcmp(comType, "UDP")) {
40)         ConfigureFlags(-1, SOCK_DGRAM, 0, AI_PASSIVE);
41)     }
42)     strcpy(ListenAddress, comIP);
43)     strcpy(ListenPort, comPort);
44)     NonBlockingMode = comNonBlock;
45)     //Security**************************************
46)     strcpy(Password, comPassword);
47)     EncryptionType = comEncryption;
48) }
49)
50) Communications::~Communications() {
51)     // TODO Auto-generated destructor stub
52)     cout << "Destructing Communication...\n";
53) }
```

54) متدی جالب که می‌تواند هر چهار حالت اجرا را پشتیبانی کند، به شرطهای داده شده و متغیرهای استفاده شده دقت کنید. هر حالت از اجرا وارد یک حلقه بینهایت می‌شود.

```
55) void Communications::Run() {
56)     if (!strcmp(ProgramMode, "Server")) {
57)         if (!strcmp(CommunicationType, "TCP")) {
58)             CreateTCPServer();
59)         } else if (!strcmp(CommunicationType, "UDP")) {
60)             CreateUDPServer();
61)         }
62)     } else if (!strcmp(ProgramMode, "Client")) {
63)         if (!strcmp(CommunicationType, "TCP")) {
64)             CreateTCPClientReceiverMode();
65)         } else if (!strcmp(CommunicationType, "UDP")) {
66)             CreateUDPClientReceiverMode();
```

Communication Class

```
67)             }
68)         }
69)         AppStat = false;
70)     }
71)     این متد فقط برای به دست آوردن آدرس deviceهای موجود بر سیستم مورد استفاده قرار می‌گیرد.
72)     void Communications::GetMachineIPs(void) {
73)         struct addrinfo *p;
74)         struct addrinfo hints;
75)         char remoteIP[INET6_ADDRSTRLEN];
76)         int cnt = 0, rv;
77)
78)         memset(&hints, 0, sizeof hints);
79)         hints.ai_family = AF_UNSPEC;    عدم توجه به نسخه آدرس
80)         hints.ai_socktype = SOCK_STREAM;    تعیین حالت اتصال اهمیتی ندارد
81)         hints.ai_flags = AI_PASSIVE;    آدرسهای این ماشین را به دست بیاور
82)     HostName نام ماشینی است که نرم‌افزار در آن اجرا شده است. این متغیر در سازنه کلاس Socket مقداردهی
        شده است.
83)         if ((rv = getaddrinfo(HostName, NULL, &hints, &p)) != 0) {
84)             fprintf(stderr, "Error on getting network device information!! (%s)\n",
85)                     gai_strerror(rv));
86)             fprintf(LogFD,
87)                     "%sError on getting network device information!! (%s)\n",
88)                     GetCurrentTime().c_str(), gai_strerror(rv));
89)             return;
90)         }
91)
92)         printf("IP addresses for '%s':\n", HostName);
93)         fprintf(LogFD, "%sIP addresses for '%s':\n", GetCurrentTime().c_str(),
94)                 HostName);
95)     در صورت وجود آدرسی برای چاپ آن ابتدا بررسی می‌کنیم که نسخه آدرس چیست (نوع IPv4 یا IPv6) سپس
        اقدام به چاپ می‌کنیم.
96)         for (; p != NULL; p = p->ai_next) {
97)             void *addr;
98)             char *ipver = new char();
99)
100)            cnt++;
101)            // get the pointer to the address itself,
102)            // different fields in IPv4 and IPv6:
103)            if (p->ai_family == AF_INET) { // IPv4
104)                struct sockaddr_in *ipv4 = (struct sockaddr_in *) p->ai_addr;
105)                addr = &(ipv4->sin_addr);
106)                strcpy(ipver, "IPv4");
107)            } else { // IPv6
108)                struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *) p->ai_addr;
109)                addr = &(ipv6->sin6_addr);
110)                strcpy(ipver, "IPv6");
111)            }
112)
```

Communication Class

```
113)            // convert the IP to a string and print it:
114)            inet_ntop(p->ai_family, addr, remoteIP, sizeof remoteIP);
115)            printf("  --device %i: %s, %s\n", cnt, ipver, remoteIP);
116)            fprintf(LogFD, "%s  --device %i: %s, %s\n", GetCurrentTime().c_str(),
117)                    cnt, ipver, remoteIP);
118)        }
119)        freeaddrinfo(p);
120)        cout << endl;
121)    }
122)    این دستور هر چند ساده به نظر می‌رسد و لی اساس ساخت سوکت را تعیین می‌کند، با استفاده از این دستور تعیین
        می‌شود که متغیرهای مورد نیاز برای ساخت سوکت دارای آرگومانهایی با چه مقادیری باشند. چهار متغیر زیر از
        کلاس Socket به ارث برده شده‌اند.
123)    void Communications::ConfigureFlags(int D, int T, int P, int F) {
124)        if (D != -1)
125)            Domain = D;
126)        if (T != -1)
127)            Type = T;
128)        if (P != -1)
129)            Protocol = P;
130)        if (F != -1)
131)            Flag = F;
132)    }
```

Networking.cpp –۱۴-۴-۴

Networking Code

```
1)  //============================================================
2)  // Name        : Networking.cpp
3)  // Author      : Ben
4)  // Version     :
5)  // Copyright   : Freeware
6)  // Description : Hello World in C++, Ansi-style
7)  //============================================================
8)  به قسمت اصلی برنامه رسیدیم. جاییکه از کلاس طراحی شده شیءی می‌سازیم؛ هویت آنرا مشخص می‌کنیم و
    سیستم شروع به کار می‌کند...
9)  #include "Communications.h"
10)
11)     متغیرهای زیر برای نگهداری موقت اطلاعات وارد شده در بخش رابط گرافیکی کاربر مورد استفاده قرار می‌گیرند.
12) char gProgramMode[10];
13) char gConnectionType[10];
14) char gIPversion[10];
15) char gIPAddress[15];
```

Networking Code

```
16) char gPort[10];
17) char gPassword[15];
18) bool gNonBlockingMode;
19) int gEncryptionType;
20)
21) Communications tmp;    ساخت شیء از کلاس طراحی شده
22)
23) دو تابع پایین برای اجرای در thread مورد استفاده قرار می‌گیرند، می‌توینستیم یکی را برای هر دو حالت استفاده
    کنیم، فقط به منظور خوانایی برنامه دو تابع متفاوت تعریف شده است.
24) void* RunServerThread(void*);
25) void* RunClientThread(void*);
26) تابع زیر برای اجرا در thread ارسال پیغام مورد استفاده قرار می‌گیرد.
27) void* SendMessage(void*);
28) ماکرو برای چاپ پیغام در هنگام رویداد خطا در ساخت thread مورد استفاده قرار می‌گیرد.
29) #define errexit(code,str)                \
30)  fprintf(stderr,"%s: %s\n",(str),strerror(code));
31) در توابع زیر دقت کنید که بخش extern "C" اضافه شده به ابتدای آنها به دلیل اتصال سیگنالها به کنترلهای
    ریبط گرافیکی طراحی شده به واسطه gtk_builder_connect_signals مورد نیاط هستند. در واقع بیان
    می‌کند که این توابع اتصالی به بیرون از فایل برنامه هستند.
32) //Program Mode*************************************************************
33) این بخش مربوط به توابع مورد نیاز به پنجره ModeWindow می‌باشند.
34) extern "C" void ConfigureAsServer(GtkWidget *widget, gpointer data) {
35) در صورتیکه دکمه ModeServer کلیک شود این تابع اجرا می‌شود.
36) حالت برنامه به متغیر مربوطه کپی می‌شود.
37)     strcpy(gProgramMode, "Server");
38) تعیین حالت TCP به عنوان پیش‌فرض در پنجره TypeServerWindow
39)
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(tmp.GW("TypeServerTCP")
    ), true);
40) نمایش پنجره TypeServerWindow و مخفی کردن پنجره کنونی
41)     gtk_widget_show(tmp.GW("TypeServerWindow"));
42)     gtk_widget_hide(tmp.GW("ModeWindow"));
43) ثبت تراکنش
44)     fprintf(tmp.LogFD, "%sStarting program in 'SERVER MODE'\n",
45)             tmp.GetCurrentTime().c_str());
46) }
47) extern "C" void ConfigureAsClient(GtkWidget *widget, gpointer data) {
48) در صورتیکه دکمه ModeServer کلیک شود این تابع اجرا می‌شود.
49) حالت برنامه به متغیر مربوطه کپی می‌شود.
50)     strcpy(gProgramMode, "Client");
51) تعیین حالت TCP به عنوان پیش‌فرض در پنجره TypeClientWindow
52)
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(tmp.GW("TypeClientTCP"))
    , true);
53) متن موجود در جعبه متن TypeClientUserName به صورت پیش‌فرض به نام ماشینی که نرم‌افزار در آن اجرا
```

Networking Code

```
                                                    شده است مقداردهی می‌شود.
54)     gtk_entry_set_text(GTK_ENTRY(tmp.GW("TypeClientUserName")),
55)             (gchar*) tmp.HostName);
56)                                   نمایش پنجره TypeClientWindow و مخفی کردن پنجره کنونی
57)     gtk_widget_show(tmp.GW("TypeClientWindow"));
58)     gtk_widget_hide(tmp.GW("ModeWindow"));
59)                                                    ثبت تراکنش
60)     fprintf(tmp.LogFD, "%sStarting program in 'CLIENT MODE'\n",
61)             tmp.GetCurrentTime().c_str());
62) }
63) //---------------------------------------------------------------------------
64) //Configure Server**********************************************************
65)                                   توابع زیر در پنجره TypeServerWindow مورد استفاده قرار می‌گیرند.
66) extern "C" void ConfigureServer(GtkWidget *widget, gpointer data) {
67)                                   با کلیک بر روی دکمه TypeServerConfigure عملیات زیر انجام می‌شود.
68)     bool InfoCheck=true;
69)     char *markup;
70)                                   دستورات زیر مشخص می‌کند که حالت اتصال چگونه است و متغیر مربوطه مقداردهی می‌شود.
71)     if (GTK_TOGGLE_BUTTON(tmp.GW("TypeServerTCP"))->active) {
72)         strcpy(gConnectionType, "TCP");
73)     } else if (GTK_TOGGLE_BUTTON(tmp.GW("TypeServerUDP"))->active) {
74)         strcpy(gConnectionType, "UDP");
75)     }
76)     fprintf(tmp.LogFD, "%sServer Connection Type: '%s'\n",
77)             tmp.GetCurrentTime().c_str(), gConnectionType);
78)                                   دستورات زیر بیان می‌کند که نسخه IP چه انتخابی بوده و متغیر مورد نظر دهی می‌شود.
79)     if (GTK_TOGGLE_BUTTON(tmp.GW("TypeServerIPv4"))->active) {
80)         strcpy(gIPversion, "IPv4");
81)     } else if (GTK_TOGGLE_BUTTON(tmp.GW("TypeServerIPv6"))->active) {
82)         strcpy(gIPversion, "IPv6");
83)     } else {
84)         strcpy(gIPversion, "UNSPEC");
85)     }
86)     fprintf(tmp.LogFD, "%sServer IP Version: '%s'\n",
87)             tmp.GetCurrentTime().c_str(), gIPversion);
88)                                   آدرس را به متغیر مربوطه کپی می‌کند.
89)     If (!strcmp(strcpy(gIPAddress, (char*)
            (GTK_ENTRY(tmp.GW("TypeServerIP")))->text),"")){
90)         markup = g_markup_printf_escaped("<span
                    foreground=\"red\">%s</span>","IP:          ");
91)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL4")), markup);
92)         g_free(markup);
93)         InfoCheck = false;
94)     } else {
95)         markup = g_markup_printf_escaped("<span
                    foreground=\"black\">%s</span>","IP:                ");
96)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL4")), markup);
97)         g_free(markup);
```

Networking Code

```
98)     }
99)                                                  پورت مورد نظر را به متغیر ربوطه کپی می‌کند.
100)    If (!strcmp(strcpy(gPort, (char*) (GTK_ENTRY(tmp.GW("TypeServerPort")))-
>text),"")){
101)            markup = g_markup_printf_escaped("<span
                        foreground=\"red\">%s</span>"," Port:         ");
102)            gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL5")), markup);
103)            g_free(markup);
104)            InfoCheck = false;
105)    } else {
106)            markup = g_markup_printf_escaped("<span
                        foreground=\"black\">%s</span>"," Port:      ");
107)            gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL5")), markup);
108)            g_free(markup);
109)    }
110)    fprintf(tmp.LogFD, "%sServer IP Port: '%s'\n",
111)              tmp.GetCurrentTime().c_str(), gPort);
112)                                                  حالت اتصال را تعیین و متغیر را مقدار دهی می‌کند.
113)    if (GTK_TOGGLE_BUTTON(tmp.GW("TypeServerNonBlock"))->active) {
114)        gNonBlockingMode = true;
115)        fprintf(tmp.LogFD, "%sServer with 'NonBlocking' sockets\n",
116)                tmp.GetCurrentTime().c_str());
117)    } else {
118)        gNonBlockingMode = false;
119)        fprintf(tmp.LogFD, "%sServer with 'Blocking' sockets\n",
120)                tmp.GetCurrentTime().c_str());
121)    }
122)                                                  نوع رمزنگاری را تعیین و متغیر را مقداردهی می‌کند.
123)    gEncryptionType = gtk_combo_box_get_active(GTK_COMBO_BOX(tmp.GW(
124)              "TypeServerEncryption")));
125)    fprintf(tmp.LogFD, "%sServer with '%s' Encryption Type\n",
126)              tmp.GetCurrentTime().c_str(),
127)              (char*) gtk_combo_box_get_active_text(GTK_COMBO_BOX(tmp.GW(
128)                        "TypeServerEncryption"))));
129)                                                  رمز عبور را به متغیر مربوطه کپی می‌کند.
130)        If (!strcmp(strcpy(gPassword, (char*)
(GTK_ENTRY(tmp.GW("TypeServerPassword")))->text);
131)            markup = g_markup_printf_escaped("<span
                        foreground=\"red\">%s</span>"," Login Password:");
132)            gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL7")), markup);
133)            g_free(markup);
134)            InfoCheck = false;
135)    } else {
136)            markup = g_markup_printf_escaped("<span
                        foreground=\"black\">%s</span>"," Login Password:");
137)            gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeServerL7")), markup);
138)            g_free(markup);
139)    }
140)    fprintf(tmp.LogFD, "%sServer Password: %s\n",
```

Networking Code

```
    tmp.GetCurrentTime().c_str(),
141)            tmp.md5(gPassword).c_str());
142)
143)        متغیرهای مقداردهی شده در دستور Init کلاس Communication مورد استفاده قرار می‌گیرند.
144)    if (InfoCheck)
145)        tmp.init(gProgramMode, gConnectionType, gIPversion, gIPAddress, gPort,
146)            gPassword, gNonBlockingMode, gEncryptionType);
147)    else
148)        Return;
149)
150)    fprintf(tmp.LogFD, "%sServer Initialized...\n",
151)            tmp.GetCurrentTime().c_str());
152)        دستورات زیر برای پنجره ServerWindow عنوان را انتساب می‌کنند و آنرا نمایش می‌دهند.
153)    char* title = new char();
154)    strcpy(title, "Server: ");
155)    strcat(title, tmp.HostName);
156)    gtk_window_set_title(GTK_WINDOW(tmp.GO("ServerWindow")), title);
157)    gtk_widget_hide(tmp.GW("TypeServerWindow"));
158)    gtk_widget_show(tmp.GW("ServerWindow"));
159)}
160)//---------------------------------------------------------------------------
161)extern "C" void ConfigureServerCancel(GtkWidget *widget, gpointer data) {
162)    از دو قسمت می‌توان وارد پنجره تنظیمات سرور (TypeServerWindow) شد، یکی از پنجره
       ModeWindow و دیگری از پنجره ServerWindow است. تشخیص اینکه از کدام پنجره وارد شده‌ایم از
       طریق تعیین Modal بودن است. بدین صورت که اگر پنجره TypeServerWindow دارای خصوصیت
       modal باشد پس حتما از ServerWindow وارد آن شده‌ایم و با زدن دکمه Cancel باید به بازگردیم و در
       غیر این صورت یعنی modal نبودن، باید به پنجره ModeWindow برگردیم. پیاده‌سازی به صورت زیر انجام
       می‌گیرد.
163)    if (GTK_WINDOW(tmp.GW("TypeServerWindow"))->modal) {
164)        gtk_widget_hide(tmp.GW("TypeServerWindow"));
165)        GTK_WINDOW(tmp.GW("TypeServerWindow"))->modal = false;
166)    } else {
167)        gtk_widget_hide(tmp.GW("TypeServerWindow"));
168)        gtk_widget_show(tmp.GW("ModeWindow"));
169)    }
170)}
171)//Configure Server**************************************************************
172)//---------------------------------------------------------------------------
173)//Configure Client**************************************************************
174)extern "C" void ConfigureClient(GtkWidget *widget, gpointer data) {
175)        با کلیک بر روی دکمه TypeClientConfigure عملیات زیر انجام می‌شود.
176)        دستورات زیر مشخص می‌کند که حالت اتصال چگونه است و متغیر مربوطه مقداردهی می‌شود.
177)    if (GTK_TOGGLE_BUTTON(tmp.GW("TypeClientTCP"))->active) {
178)        strcpy(gConnectionType, "TCP");
179)    } else if (GTK_TOGGLE_BUTTON(tmp.GW("TypeClientUDP"))->active) {
180)        strcpy(gConnectionType, "UDP");
181)    }
```

Networking Code

```
182)     fprintf(tmp.LogFD, "%sClient Connection Type: '%s'\n",
183)             tmp.GetCurrentTime().c_str(), gConnectionType);
184)                                   دستورات زیر بیان می‌کند که نسخه IP چه انتخابی بوده و متغیر مورد نظر مقدار دهی می‌شود.
185)     if (GTK_TOGGLE_BUTTON(tmp.GW("TypeClientIPv4"))->active) {
186)         strcpy(gIPversion, "IPv4");
187)     } else if (GTK_TOGGLE_BUTTON(tmp.GW("TypeClientIPv6"))->active) {
188)         strcpy(gIPversion, "IPv6");
189)     } else {
190)         strcpy(gIPversion, "UNSPEC");
191)     }
192)     fprintf(tmp.LogFD, "%sClient IP Version: '%s'\n",
193)             tmp.GetCurrentTime().c_str(), gIPversion);
194)                                   آدرس را به متغیر مربوطه کپی می‌کند.
195)     If (!strcmp(strcpy(gIPAddress, (char*)
(GTK_ENTRY(tmp.GW("TypeClientIP")))->text);
196)             markup = g_markup_printf_escaped("<span
                        foreground=\"red\">%s</span>","IP:              ");
197)             gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL4")), markup);
198)             g_free(markup);
199)             InfoCheck = false;
200)     } else {
201)             markup = g_markup_printf_escaped("<span
                        foreground=\"black\">%s</span>"," IP:                    ");
202)             gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL4")), markup);
203)             g_free(markup);
204)     }
205)     fprintf(tmp.LogFD, "%sClient IP Address: '%s'\n",
206)             tmp.GetCurrentTime().c_str(), gIPAddress);
207)                                   پورت مورد نظر را به متغیر ربوطه کپی می‌کند.
208)     If (!strcmp(strcpy(gPort, (char*) (GTK_ENTRY(tmp.GW("TypeClientPort")))-
>text);
209)             markup = g_markup_printf_escaped("<span
                        foreground=\"red\">%s</span>"," Port:        ");
210)             gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL5")), markup);
211)             g_free(markup);
212)             InfoCheck = false;
213)     } else {
214)             markup = g_markup_printf_escaped("<span
                        foreground=\"black\">%s</span>"," Port:      ");
215)             gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL5")), markup);
216)             g_free(markup);
217)     }
218)     fprintf(tmp.LogFD, "%sClient IP Port: '%s'\n",
219)             tmp.GetCurrentTime().c_str(), gPort);
220)                                   حالت اتصال را تعیین و متغیر را مقدار دهی می‌کند.
221)     if (GTK_TOGGLE_BUTTON(tmp.GW("TypeClientNonBlock"))->active) {
222)         gNonBlockingMode = true;
223)         fprintf(tmp.LogFD, "%sClient with 'NonBlocking' sockets\n",
224)                 tmp.GetCurrentTime().c_str());
```

Networking Code

```
225)     } else {
226)         gNonBlockingMode = false;
227)         fprintf(tmp.LogFD, "%sClient with 'Blocking' sockets\n",
228)                 tmp.GetCurrentTime().c_str());
229)     }
230)                                   نوع رمزنگاری را تعیین و متغیر را مقداردهی می‌کند.
231)     gEncryptionType = gtk_combo_box_get_active(GTK_COMBO_BOX(tmp.GW(
232)             "TypeClientEncryption")));
233)     fprintf(tmp.LogFD, "%sClient with '%s' Encryption Type\n",
234)             tmp.GetCurrentTime().c_str(),
235)             (char*) gtk_combo_box_get_active_text(GTK_COMBO_BOX(tmp.GW(
236)                     "TypeClientEncryption"))));
237) نام کاربری را به متغیر مربوطه کپی می‌کند. متغیر UserName از کلاس Socket مشتق شده است و مستقاما
                                   در آنجا مقدار دهی می‌شود.
238)     If (!strcmp(strcpy(tmp.UserName,
239)             (char*) (GTK_ENTRY(tmp.GW("TypeClientUserName")))->text);
240)         markup = g_markup_printf_escaped("<span
                     foreground=\"red\">%s</span>"," User Name:          ");
241)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL8")), markup);
242)         g_free(markup);
243)         InfoCheck = false;
244)     } else {
245)         markup = g_markup_printf_escaped("<span
                     foreground=\"black\">%s</span>"," User Name:       ");
246)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL8")), markup);
247)         g_free(markup);
248)     }
249)     fprintf(tmp.LogFD, "%sClient UserName: %s\n",
tmp.GetCurrentTime().c_str(),
250)             tmp.UserName);
251)                                   رمز عبور را به متغیر مربوطه کپی می‌کند.
252)     If (!strcmp(strcpy(gPassword, (char*)
(GTK_ENTRY(tmp.GW("TypeClientPassword")))->text);
253)         markup = g_markup_printf_escaped("<span
                     foreground=\"red\">%s</span>","Password:  ");
254)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL9")), markup);
255)         g_free(markup);
256)         InfoCheck = false;
257)     } else {
258)         markup = g_markup_printf_escaped("<span
                     foreground=\"black\">%s</span>"," Password:          ");
259)         gtk_label_set_markup(GTK_LABEL(tmp.GW("TypeClientL9")), markup);
260)         g_free(markup);
261)     }
262)     fprintf(tmp.LogFD, "%sClient Password: %s\n",
tmp.GetCurrentTime().c_str(),
263)             tmp.md5(gPassword).c_str());
264)
```

Networking Code

متغیرهای مقداردهی شده در دستور Init کلاس Communication مورد استفاده قرار می‌گیرند

```
    if (InfoCheck)
        tmp.init(gProgramMode, gConnectionType, gIPversion, gIPAddress, gPort,
            gPassword, gNonBlockingMode, gEncryptionType);
    else
        Return;

    fprintf(tmp.LogFD, "%sClient Initialized...\n",
            tmp.GetCurrentTime().c_str());
```

دستورات زیر برای پنجره ClientWindow عنوان را انتساب می‌کنند.

```
    char* title = new char();
    strcpy(title, "Client: ");
    strcat(title, tmp.UserName);
    gtk_window_set_title(GTK_WINDOW(tmp.GO("ClientWindow")), title);
    gtk_widget_hide(tmp.GW("TypeClientWindow"));
    gtk_widget_show(tmp.GW("ClientWindow"));
}
extern "C" void ConfigureClientCancel(GtkWidget *widget, gpointer data) {
```

از دو قسمت می‌توان وارد پنجره تنظیمات سرور (TypeClientWindow) شد، یکی از پنجره ModeWindow و دیگری از پنجره ClientWindow است. تشخیص اینکه از کدام پنجره وارد شده‌ایم از طریق تعیین Modal بودن است. بدین صورت که اگر پنجره TypeClientWindow دارای خصوصیت modal باشد پس حتما از ClientWindow وارد آن شده‌ایم و با زدن دکمه Cancel باید به بازگردیم و در غیر این صورت یعنی modal نبودن، باید به پنجره ModeWindow برگردیم. پیاده‌سازی به صورت زیر انجام می‌گیرد.

```
    if (GTK_WINDOW(tmp.GW("TypeClientWindow"))->modal) {
        gtk_widget_hide(tmp.GW("TypeClientWindow"));
        GTK_WINDOW(tmp.GW("TypeClientWindow"))->modal = false;
    } else {
        gtk_widget_hide(tmp.GW("TypeClientWindow"));
        gtk_widget_show(tmp.GW("ModeWindow"));
    }
}
//---------------------------------------------------------------------------
//Server*************************************************************************
```

توابع زیر در پنجره ServerWindow مورد استفاده قرار می‌گیرند.

```
extern "C" void RunServer(GtkWidget *widget, gpointer data) {
```

سرور را اجرا می‌کند. دقت کنید که تمام داده‌های مورد نیاز برای اجرای سرور در پنجره قبلی توسط دکمه Configure و دستور Init مقداردهی شده‌اند. اجرای سرور در یک thread انجام می‌شود. دلیل این کار حلقه بینهایت موجود در آن است. دلیل دیگر نیز بلوکه شدن بعضی از توابع مربوط به سوکت است. این thread تمام عملیات Manage را به صورت همروند با رابط گرافیکی انجام می‌دهد تا از freeze شدن برنامه جلوگیری شود.

```
    pthread_t th;
    int errcode; /* holds pthread error code */
    if (errcode = pthread_create(&th, NULL, RunServerThread, &tmp)) {
        errexit(errcode,"pthread_create");
    }
    fprintf(tmp.LogFD, "%sServer Started...\n", tmp.GetCurrentTime().c_str());
```

Networking Code

```
303)  }
304)  void* RunServerThread(void* arg) {
305)      ((Communications*) arg)->Run();
306)      pthread_exit(arg);
307)  }
308)  نابع زیر اجرای سرور را از کار می‌انداز، برای اینکار فقط کافیست دستور FlushAll را اجرا کنیم تا برنامه reset
      شود.
309)  extern "C" void StopServer(GtkWidget *widget, gpointer data) {
310)      if (tmp.AppStat)
311)          tmp.FlushAll();
312)  }
313)  تابع زیر واسطی برای اجرای متد DisconnectClients است.
314)  extern "C" void DisconnectClients(GtkWidget *widget, gpointer data) {
315)      tmp.DisconnectClients();
316)      fprintf(tmp.LogFD, "%sServer Disconnected Clients...\n",
317)              tmp.GetCurrentTime().c_str());
318)  }
319)  پیغامی مبنی بر ذخیره شدن تراکنشها چاپ می‌کند. (میتوان از وجود آن صرفه نظر کرد)
320)  extern "C" void SaveLog(GtkWidget *widget, gpointer data) {
321)      cout << "Log is autosaving...\n";
322)  }
323)  پنجره تنظیمات سرور را به صورت modal نمایش می‌دهد.
324)  extern "C" void ServerPreferences(GtkWidget *widget, gpointer data) {
325)      GTK_WINDOW(tmp.GW("TypeServerWindow"))->modal = true;
326)
      gtk_window_set_transient_for(GTK_WINDOW(tmp.GW("TypeServerWindow")),
327)              GTK_WINDOW(tmp.GW("ServerWindow")));
328)      gtk_widget_show(tmp.GW("TypeServerWindow"));
329)  }
330)  //-------------------------------------------------------------------------
331)  //Client*****************************************************************
332)  توابع زیر در پنجره ClientWindow مورد استفاده قرار می‌گیرند.
333)  extern "C" void RunClient(GtkWidget *widget, gpointer data) {
334)  Client را اجرا می‌کند. دقت کنید که تمام داده‌های مورد نیاز برای اجرای Client در پنجره قبلی توسط دکمه
      Configure و دستور Init مقداردهی شده‌اند. اجرای Client در یک thread انجام می‌شود. دلیل این کار
      حلقه بینهایت موجود در آن است. دلیل دیگر نیز بلوکه شدن بعضی از توابع مربوط به سوکت است. این thread
      تمام عملیات Receive را به صورت همروند با رابط گرافیکی انجام می‌دهد تا از freeze شدن برنامه جلوگیری
      شود.
335)      pthread_t th;
336)      int errcode; /* holds pthread error code */
337)      if (errcode = pthread_create(&th, NULL, RunClientThread, &tmp)) {
338)          errexit(errcode,"pthread_create");
339)      }
340)      fprintf(tmp.LogFD, "%sClient Started...\n", tmp.GetCurrentTime().c_str());
341)  }
342)  void* RunClientThread(void* arg) {
343)      ((Communications*) arg)->Run();
```

Networking Code

```
344)     pthread_exit(arg);
345) }
346) این تابع نیز واسطی برای اجرا متن ClientSendMode است. اجرای آن در thread انجام شده تا به صورت
     همروند ارسال متن موجود در جعبه متن ClientSnedText انجام شود. این کار بدین درلیل صورت می‌گیرد تا
     اختلالی در اجرای برنامه از جمله بلوکه شدن صورت نگیرد.
347) extern "C" void SendMessage(GtkWidget *widget, gpointer data) {
348)     pthread_t th;
349)     int errcode; /* holds pthread error code */
350)     if (errcode = pthread_create(&th, NULL, SendMessage, &tmp)) {
351)         errexit(errcode,"pthread_create");
352)     }
353) }
354) void* SendMessage(void* arg) {
355)     ((Communications*) arg)->ClientSendMode();
356)     gtk_widget_grab_focus(tmp.GW("ClientSendText"));
357)     gdk_window_process_all_updates();
358)     pthread_exit(arg);
359) }
360) پنجره تنظیمات Client را به صورت modal نمایش می‌دهد.
361) extern "C" void ClientPreferences(GtkWidget *widget, gpointer data) {
362)     GTK_WINDOW(tmp.GW("TypeClientWindow"))->modal = true;
363)
     gtk_window_set_transient_for(GTK_WINDOW(tmp.GW("TypeClientWindow")),
364)             GTK_WINDOW(tmp.GW("ClientWindow")));
365)     gtk_widget_show(tmp.GW("TypeClientWindow"));
366) }
367) نابع زیر اجرای سرور را از کار می‌انداز، برای اینکار فقط کافیست دستور FlushAll را اجرا کنیم تا برنامه reset
     شود.
368) extern "C" void StopClient(GtkWidget *widget, gpointer data) {
369)     if (tmp.AppStat)
370)         tmp.FlushAll();
371) }
372) تابع زیر فقط برای این طراحی شده که به هنگام خالی بودن جعبه متن دکمه Send را غیر فعال کند، تا از ارسال
     پیغام خالی جلوگیری شود.
373) extern "C" void on_ClientSendBuffer_changed(GtkWidget *widget, gpointer data)
     {
374)     GtkTextIter start, end;
375)     GtkTextBuffer *buffer = GTK_TEXT_BUFFER(tmp.GO("ClientSendBuffer"));
376)     gtk_text_buffer_get_bounds(buffer, &start, &end);
377)     if (!strcmp((char*) gtk_text_buffer_get_text(buffer, &start, &end, false),
378)             "")) {
379)         gtk_widget_set_sensitive(tmp.GW("ClientSend"), false);
380)     } else {
381)         gtk_widget_set_sensitive(tmp.GW("ClientSend"), true);
382)     }
383) }
384) //---------------------------------------------------------------------------
```

Networking Code

```
385)  تابعی که برای دکمه‌های خروج تعبیه شده، پیغامی چاپ می‌کند (برای زیبایی :D)
386)  extern "C" void print_hello(GtkWidget *widget, gpointer data) {
387)      g_print("Hello World\n");
388)      gtk_main_quit();
389)  }
390)  تابع main از اینجا آغاز می‌شود.
391)  int main(int argc, char *argv[]) {
392)      fprintf(tmp.LogFD, "%sNetworking Program Loaded...\n",
393)              tmp.GetCurrentTime().c_str());
394)      tmp.GetMachineIPs();    چاپ آدرسهای ماشین
395)      gtk_init(&argc, &argv);    تابع فراخوانی گرافیک
396)      tmp.Builder = gtk_builder_new();    اختصاص فضا برای سازنده گرافیکی برنامه
397)  تولید Objectهای گرافیکی؛ فایل طراحی شده توسط نرم‌افزار Glade حاوی دستوراتی است که بیان می‌کند
      طراحی چگونه است، با استفاده از دستور پایین، پارسر تمام کدها را تبدیل به GtkObjectها می‌کند.
398)      gtk_builder_add_from_file(tmp.Builder, "Networking.glade", NULL);
399)  در طراحی صورت گرفته به بعضی از Objectها، سیگنال تفویض کردیم؛ این سیگلاها در بالا تعریف شدند. اگر
      یادمان باشد منظور توابعی است که در ابتدای آنها "C" extern اضافه شده بود.
400)      gtk_builder_connect_signals(tmp.Builder, NULL);
401)  Object مورد نظر را به Widget، cast کرده و نمایش می‌دهیم.
402)      gtk_widget_show_all(tmp.GW("ModeWindow"));
403)      fprintf(tmp.LogFD, "%sNetworking Program Initialized...\n",
404)              tmp.GetCurrentTime().c_str());
405)      //tags****************************************************************
406)  در زیر tagهایی را تولید کرده‌ایم، آنها را به جعبه متن مربوط به سرور الصاق دادیم. هدف استفاده از این Tagها به
      این منظور است که در هنگام اضافه کردن متن به جعبه متن، این tagها حالت اضافه شدن متن را تعیین می‌کنند.
407)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
408)              "italic", "style", PANGO_STYLE_ITALIC, NULL);
409)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
410)              "bold", "weight", PANGO_WEIGHT_BOLD, NULL);
411)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
412)              "error", "background", "white", "foreground", "#C72C20", "size",
413)              12000, "style", PANGO_STYLE_ITALIC, NULL);
414)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
415)              "success", "background", "white", "foreground", "#75A672", "size",
416)              12000, "style", PANGO_STYLE_ITALIC, NULL);
417)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
418)              "lost", "background", "white", "foreground", "#C7A020", "size",
419)              12000, "style", PANGO_STYLE_ITALIC, NULL);
420)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
421)              "time", "background", "white", "foreground", "#B1BEFF", "size",
422)              9000, NULL);
423)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ServerTextBuffer")),
      "",
424)              NULL);
425)      //********************************************************************
426)      //tags****************************************************************
```

Networking Code

```
427)  همانند بالا tagهای زیر را نیز به جعبه متن اصلی در پنجره Client الصاق می‌کنیم.
428)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
429)              "italic", "style", PANGO_STYLE_ITALIC, NULL);
430)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
431)              "bold", "weight", PANGO_WEIGHT_BOLD, NULL);
432)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
433)              "error", "background", "white", "foreground", "#C72C20", "size",
434)              12000, "style", PANGO_STYLE_ITALIC, NULL);
435)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
436)              "success", "background", "white", "foreground", "#75A672", "size",
437)              12000, "style", PANGO_STYLE_ITALIC, NULL);
438)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
439)              "lost", "background", "white", "foreground", "#C7A020", "size",
440)              12000, "style", PANGO_STYLE_ITALIC, NULL);
441)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
442)              "time", "background", "white", "foreground", "#B1BEFF", "size",
443)              9000, NULL);
444)      gtk_text_buffer_create_tag(GTK_TEXT_BUFFER(tmp.GO("ClientTextBuffer")),
      "",
445)              NULL);
446)      //**********************************************************************
447)      fprintf(tmp.LogFD, "%sText Tags Added...\n", tmp.GetCurrentTime().c_str());
448)
449)      fprintf(tmp.LogFD, "%sNetworking Program Started...\n",
450)              tmp.GetCurrentTime().c_str());
451)
452)      gtk_main();    اجرای حلقه اصلی گرافیکی
453)
454)      fprintf(tmp.LogFD, "%sNetworking Program Ended...\n",
455)              tmp.GetCurrentTime().c_str());
456)                                                  پایان برنامه با نمایشی جانانه :D
457)      cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
458)      return 0;
459)  }
```

فصل ۵:

جمع‌بندی و پیشنهادها

۵-۱- مقدمه

امروزه در دنیایی زندگی می‌کنیم که همه فرایندها بر اساس ارتباطات از راه دور انجام می‌شوند. نامه‌نگاریهای الکترونیکی، پیامهای از راه دور نمونه‌های بسیار کوچک و در عین حال با اهمیتی در این زمینه‌اند. برنامه‌نویسی تحت شبکه روشی به عنوان تولید ابزار مناسب برای برقراری ارتباطات به کار می‌رود.

مهمترین مسئله در ارتباطات شبکه‌ای امنیت داده‌ها می‌باشد. سیستم عامل لینوکس بر پایه شبکه طوری پیاده‌سازی شده است که از ابتدا مسئله امنیت در آن گنجانده شود. در واقع سیستم عاملهای تحت یونیکس نقطه شروعی بر برنامه‌نویسی شبکه بودند.

هدف اصلی این پروژه گام برداشتن در زمینه نرم‌افزارهای ارتباط جمعی و گسترش این فناوری بوده است. پیاده‌سازی نرم‌افزاری ساده برای برقراری ارتباط بین چند کاربر با استفاده از روشهایی خلاقانه در ادامه کار این پروژه اهمیت بسزایی داشت.

تحت لینوکس بودن این نرم‌افزار ویژگی جالب آن می‌باشد؛ زیرا در کشوری زندگی می‌کنیم که سیستم عاملهای ویندوز دارای محبوبیت هستند ولی از طرف دیگر نقض قوانین کپی‌رایت زیان بزرگی در پیشرفت فناوریهای برنامه‌نویسی خصوصا در زمینه شبکه را وارد می‌کند. لینوکس نرم‌افزاری OpenSource است و اکثر نرم‌افزارهای آن نیز همینطور هستند. و این بدین معناست که قانون کپی‌رایت در سطح لینوکس طور دیگری تفسیر می‌شود. نوشتن نرم‌افزارهای OpenSource زیان مالی را متضرر نمی‌شوند بلکه باعث ایجاد رغبت برای برنامه‌نویسی نیز می‌شود.

۵-۲- نرم‌افزار Networking در یک نگاه

در این نرم‌افزار سعی شد نسخه‌ای بسیار ساده از برنامه Yahoo! Messenger طراحی و پیاده‌سازی شود. این تناسب فقط برای درک هرچه بهتر و راحت‌تر مفاهیم ارتباطی بود که در سطح یک شبکه محلی مورد استفاده واقع می‌شود. البته خصوصیات ویژه این نرم‌افزار، تعیین ویژگیهای ارتباطی از جانب کاربر سیستم است که منجر به تخصصی شدن نرم‌افزار می‌شود. این ویژگی‌ها کارب را قادر می‌سازند که مفاهیم شبکه‌ای را به صورت واقعی لمس کند. به عنوان مثال تفاوت بین ارتباط TCP را با UDP عملا درک می‌کند.

۵-۲-۱- نوآوری

در طی چند سال تحصیل دانشگاهی تجربه برنامه‌نویسی تحت شبکه در سیستم عامل لینوکس دارای جذابیتهایی بود. نرم‌افزار تهیه شده دارای رابط گرافیکی با پوسته بسیار زیبا است که امکانات مناسبی را برای برنامه‌نویسان فراهم آورده است. در سمت دیگر توابع کتابخانه‌ای مربوط به امنیت داده‌ها با نام OpenSSL خدمات جالبی را برای استفاده به عرضه گذاشته است. همه این‌ها در کنار هم باعث تولید نرم‌افزاری حائز اهمیت مخصوصا برای خودم را به ارمغان آورد.

۵-۲-۲- پیشنهادها

اولین نکته‌ای که باید ذکر کنم انتخاب محیط برنامه‌نویسی مناسب است. زیرا کمک قابل توجهی به روند برنامه‌نویسی می‌کند. در برنامه‌نویسی فکر کنید و از قلم و کاغذ استفاده کنید. ساخت نرم‌افزارهای تحت لینوکس بسیار ساده است، ولی به دلیل جا

نیفتادن فرهنگ استفاده از آن، یادگیریش نیز کمی همت می‌خواهد. اما مشخص است که در آینده ویندوز در اکثر مکانها جایگزین لینوکس خواهد شد. به همین دلیل نیاز است که یادگیری این سیستم عامل و مخصوصا برنامه‌نویسی تحت آن را یاد گرفت. امروزه اکثر سرورهای جهان تحت سیستم عاملهای خانواده یونیکس اجرا می‌شوند. در لینوکس در حقیقت دست برنامه‌نویس باز است، لینوکس منبع تمام توابع کتابخانه‌ای است. در لینوکس در بسته‌ای وجود ندارد. ولی این نکته را نیز باید ذکر کنم: Bug [9] تنها مشکل پیش روی شما در این سیستم عامل است. ولی برای رسیدن به موفقیت باید دست‌اندازهایی را رد کرد.

مراجع

مراجع

[1] Mani Radhakrishnan and Jon Solworth, “Socket Programming in C/C++”, 2004

[2] https://secure.wikimedia.org/wikipedia/en/wiki/Berkeley_sockets, visited july 2011

[۳] احسان ملکیان، "اصول مهندسی اینترنت "، عضو هیئت علمی دانشگاه تربیت معلم تهران، انتشارات نص، ویراست دوم، ۱۳۸۴

[4] [3] Smith, Lucie; Lipner, Ian (3 February 2011). "Free Pool of IPv4 Address Space Depleted". Number Resource Organization. Retrieved 3 February 2011.

[5] [4] ICANN,nanog mailing list. "Five /8s allocated to RIRs - no unallocated IPv4 unicast /8s remain".

[6] RFC 791, Internet Protocol - DARPA Internet Program Protocol Specification (September 1981)

[7] http://beej.us/guide, visited july 2011

[8] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, "UNIX Network Programming: The sockets networking API", Addison-Wesley Professional, 2004

[9] https://bugzilla.redhat.com/show_bug.cgi?id=617977, visited july 2011

پیوست‌ها

پیوست الف

Header files

The Berkeley socket interface is defined in several header files. The names and content of these files differ slightly between implementations. In general, they include:

`<sys/socket.h>`
Core BSD socket functions and data structures.
`<netinet/in.h>`
AF_INET and AF_INET6 address families and their corresponding protocol families PF_INET and PF_INET6. Widely used on the Internet, these include IP addresses and TCP and UDP port numbers.
`<sys/un.h>`
PF_UNIX/PF_LOCAL address family. Used for local communication between programs running on the same computer. Not used on networks.
`<arpa/inet.h>`
Functions for manipulating numeric IP addresses.
`<netdb.h>`
Functions for translating protocol names and host names into numeric addresses. Searches local data as well as DNS.

پیوست ب

Socket API functions

This list is a summary of functions or methods provided by the Berkeley sockets API library:

- `socket()` creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.
- `bind()` is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.
- `listen()` is used on the server side, and causes a bound TCP socket to enter listening state.
- `connect()` is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.
- `accept()` is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
- `send()` and `recv()`, or `write()` and `read()`, or `sendto()` and `recvfrom()`, are used for sending and receiving data to/from a remote socket.
- `close()` causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.
- `gethostbyname()` and `gethostbyaddr()` are used to resolve host names and addresses. IPv4 only.
- `select()` is used to prune a provided list of sockets for those that are ready to read, ready to write, or that have errors.
- `poll()` is used to check on the state of a socket in a set of sockets. The set can be tested to see if any socket can be written to, read from or if an error occurred.
- `getsockopt()` is used to retrieve the current value of a particular socket option for the specified socket.
- `setsockopt()` is used to set a particular socket option for the specified socket.

پیوست ج

Protocol and address families

The socket API is a general interface for Unix networking and allows the use of various network protocols and addressing architectures.

The following lists a sampling of protocol families (preceded by the standard symbolic identifier) defined in a modern Linux or BSD implementation:

```
PF_LOCAL, PF_UNIX, PF_FILE
                Local to host (pipes and file-domain)
PF_INET         IP protocol family
PF_AX25         Amateur Radio AX.25
PF_IPX          Novell Internet Protocol
PF_APPLETALK    Appletalk DDP
PF_NETROM       Amateur radio NetROM
PF_BRIDGE       Multiprotocol bridge
PF_ATMPVC       ATM PVCs
PF_X25          Reserved for X.25 project
PF_INET6        IP version 6
PF_ROSE         Amateur Radio X.25 PLP
PF_DECnet       Reserved for DECnet project
PF_NETBEUI      Reserved for 802.2LLC project
PF_SECURITY     Security callback pseudo AF
PF_KEY          PF_KEY key management API
PF_NETLINK, PF_ROUTE
                routing API
PF_PACKET       Packet family
PF_ASH          Ash
PF_ECONET       Acorn Econet
PF_ATMSVC       ATM SVCs
PF_SNA          Linux SNA Project
PF_IRDA         IRDA sockets
PF_PPPOX        PPPoX sockets
PF_WANPIPE      Wanpipe API sockets
PF_BLUETOOTH    Bluetooth sockets
```

پیوست د

RFCs

RFCs—the real dirt! These are documents that describe assigned numbers, programming APIs, and protocols that are used on the Internet. I've included links to a few of them here for your enjoyment, so grab a bucket of popcorn and put on your thinking cap:

RFC 1—The First RFC; this gives you an idea of what the "Internet" was like just as it was coming to life, and an insight into how it was being designed from the ground up. (This RFC is completely obsolete, obviously!)

RFC 768—The User Datagram Protocol (UDP)

RFC 791—The Internet Protocol (IP)

RFC 793—The Transmission Control Protocol (TCP)

RFC 854—The Telnet Protocol

RFC 959—File Transfer Protocol (FTP)

RFC 1350—The Trivial File Transfer Protocol (TFTP)

RFC 1459—Internet Relay Chat Protocol (IRC)

RFC 1918—Address Allocation for Private Internets

RFC 2131—Dynamic Host Configuration Protocol (DHCP)

RFC 2616—Hypertext Transfer Protocol (HTTP)

RFC 2821—Simple Mail Transfer Protocol (SMTP)

RFC 3330—Special-Use IPv4 Addresses

RFC 3493—Basic Socket Interface Extensions for IPv6

RFC 3542—Advanced Sockets Application Program Interface (API) for IPv6

RFC 3849—IPv6 Address Prefix Reserved for Documentation

RFC 3920—Extensible Messaging and Presence Protocol (XMPP)

RFC 3977—Network News Transfer Protocol (NNTP)

RFC 4193—Unique Local IPv6 Unicast Addresses

RFC 4506—External Data Representation Standard (XDR)

Golestan University
Computer Engineering Department

Designing and implementing network communication class through Linux OS sockets in C++ Programming Language

A Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Bachelor of Science in Computer Engineering

By:
Beneil Eisavi

Supervisor:
Dr. Alireza Mahini

September 2011