به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

**شبکه­های کامپیوتری**

**گزارش پروژه اول**

**سجاد علی­زاده – 810197547**

**سارینا همدانی - 810197606**

# ساختار پروژه

این پروژه سه پوشه اصلی دارد. پوشه bin که فایل­های قابل اجرا در آن قرار می­گیرد. پوشه config که فایل کانفیگ مربوط به سرور و کلاینت در آن قرار می­گیرد. پوشه src که کدهای اصلی برنامه در آن قرار میگیرد. همچنین در کنار این پوشه­ها Makefile قرار دارد که برای تولید فایل­های قابل اجرا، اجرا کردن آن کافیست.

برای اجرا کردن سرور به شکل زیر عمل می­کنیم:



آرگومان اول آن آدرس فایل کانفیگ و آرگومان دوم آن آدرس پوشه مربوط به سرور ftp است. (همان پوشه­ای که کلاینت­ها با وصل شدن به سرور به آن­ها دسترسی پیدا میکنند) که فایل­های سرور در آن قرار دارد.

توجه کنید فایل log.txt که لاگ سرور است در همان محلی که از آنجا سرور را اجرا می­کنیم ساخته می­شود.

برای اجرا کردن کلاینت به شکل زیر عمل می­کنیم:



آرگومان کلاینت آدرس فایل کانفیگ است.

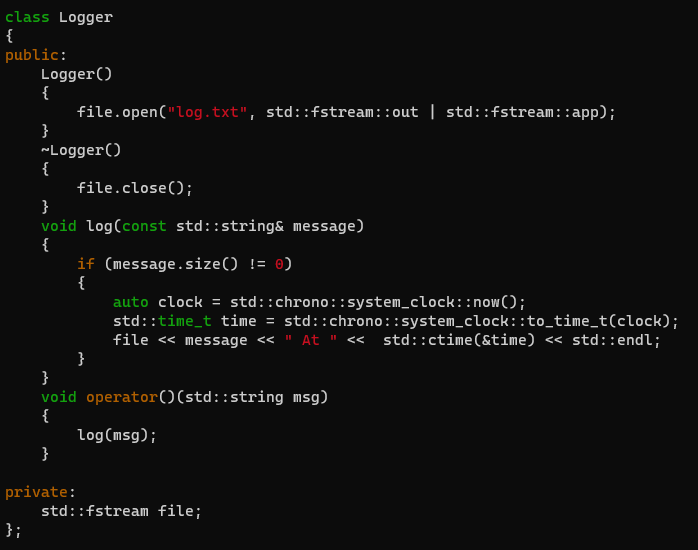
Makefile به گونه­ای نوشته شده است که سرور و کلاینت را به کتابخانه تبدیل کند (که در پوشه bin/lib هستند) و این کتابخانه­ها را به صورت ایستا به پروژه لینک کند.

# سرور و کلاس­های مربوط

حال به بررسی کدهای برنامه میپردازیم که در پوشه src هستند.

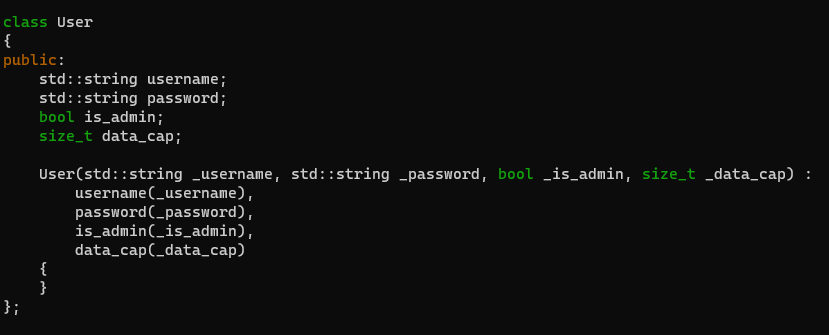
### فایل include/logger-inl.h:

این فایل مربوط به کلاس logger است که به شکل زیر است:



|  |  |
| --- | --- |
| Logger() | کانستراکتور این کلاس که یک فایل با نام log.txt باز میکند. توجه کنید تنظیمات به گونه­ای قرار داده شده که در صورتی که فایل موجود بود از انتهای آن شروع به نوشتن کند. |
| ~Logger() | دیستراکتور این کلاس که فایل باز شده را میبندد. |
| log() | آرگومان ورودی این تابع همراه با تاریخ و ساعت که با استفاده از کتابخانه chrono به دست می­آید در فایل نوشته می­شود. |
| operator()() | اپراتور پرانتز که برای راحتی کار overload شده است. |

### فایل include/user.h:



در این کلاس اطلاعات مربوط به کاربر نگهداری میشود که شامل چهار فیلد نام کاربری، رمز عبور، ادمین بودن یا نبودن و حجم مجاز مصرفی است. همچنین یک کانستراکتور برای مقدار دادن به این فیلدها قرار داده شده است.

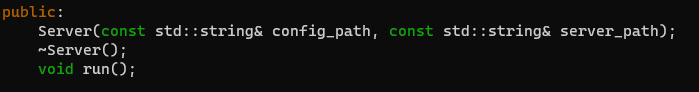
### فایل include/responses.h:



در این تابع به ازای هر پیامی که در صورت پروژه آمده است یک ماکرو Define شده است. هر کجا نیاز به این پیام­ها بود از ماکرو مخصوص به آن استفاده میکنیم. همچنین برای اینکه پیام­ها در ترمینال رنگی شود از دو ماکرو دیگر با نام­های RED و GREEN استفاده شده است که پیام را رنگی میکنند.

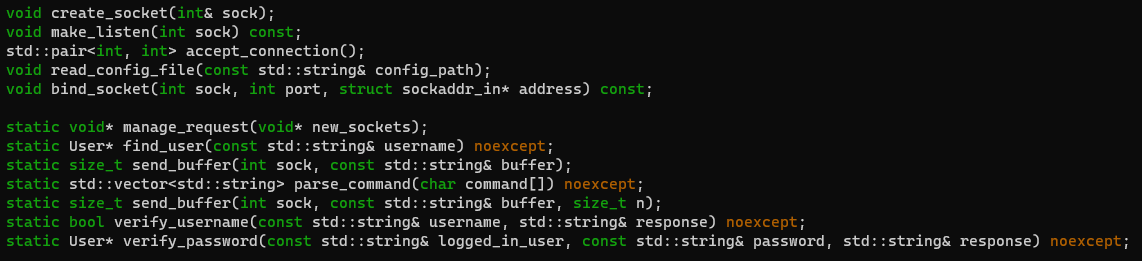
### فایل include/server.h:

این فایل header کلاس سرور است. متدهای عمومی آن به شرح زیر هستند:



|  |  |
| --- | --- |
| Server() | کانستراکتور کلاس سرور که آرگومان­های آن به ترتیب آدرس فایل کانفیگ و آدرس فایل­های سرور هستند. |
| ~Server() | دیستراکتور کلاس سرور |
| run() | وقتی میخواهیم سرور شروع به اجرا کند و connection ها را دریافت کند این متد را صدا میزنیم |

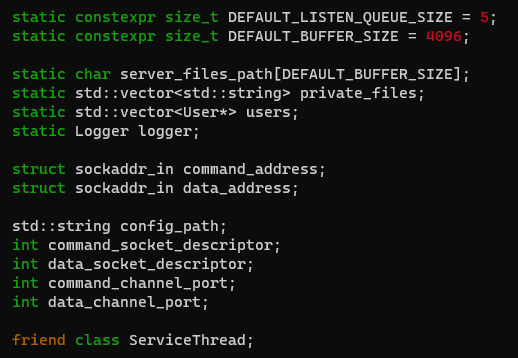
متدهای خصوصی نیز به شرح زیر هستند:



|  |  |
| --- | --- |
| create\_socket() | این تابع رفرنس یک int را دریافت می­کند و با استفاده از آن یک سوکت tcp ایجاد میکند. در صورت عدم موفقیت یک استثنا پرتاب میشود. |
| make\_listen() | این تابع یک سوکت دریافت میکند و بر روی آن گوش میکند. در صورت عدم موفقیت یک استثنا پرتاب میشود. |
| accept\_connection() | این تابع connection­هایی که دریافت میشود را accept میکند. خروجی آن یک دوتایی از سوکت است که یکی از آن­ها سوکت دستور و دیگری سوکت داده است. در صورت عدم موفقیت یک استثنا پرتاب میشود. |
| read\_config\_file() | این تابع وظیفه خواندن فایل کانفیگ را دارد و اطلاعات آن را ذخیره میکند. در صورت عدم موفقیت یک استثنا پرتاب میشود. |
| bind\_socket() | این تابع سوکت را به یک آدرس و پورت bind میکند. در صورت عدم موفقیت یک استثنا پرتاب میشود. |
| manage\_request() | هنگامی که یک کانکشن جدید می­رسد، یک thread جدید ایجاد میشود. این تابع همان تابع thread است و ورودی آن سوکت­های دستور و داده است. |
| find\_user() | این تابع با دریافت یک نام کاربری، تمام اطلاعات مربوط را برمیگرداند. درصورتی که نام کاربری یافت نشد nullptr برمیگردد. |
| send\_buffer() | این تابع با دریافت یک سوکت و یک رشته، رشته را به سوکت داده شده ارسال میکند. در صورت عدم موفقیت استثنا پرتاب میشود. |
| parse\_command() | این تابع یک رشته دستور را پارس کرده و آرگومان­های آن را در یک بردار میریزد و آن بردار را برمیگرداند. |
| send\_buffer() | این تابع overload تابع قبلی است با این تفاوت که علاوه بر سوکت و رشته برای ارسال، اندازه­ای که به آن مقدار ارسال صورت پذیرد هم داده میشود. |
| verify\_username() | این تابع با دریافت یک نام کاربری چک میکند که آیا این نام کاربری در سیستم وجود دارد یا خیر. در صورتی که وجود دارد true و در غیر این صورت false را برمیگرداند. همچنین یکی از آرگومان­های آن رفرنس رشته است که پاسخ متناظر با درخواست نام کاربری در آن ریخته میشود. |
| verify\_password() | با نام کاربری و رمز عبور داده شده کاربر مربوطه یافت میشود. در صورتی که اطلاعات غلط بود nullptr به عنوان خروجی بازگردانده میشود و response هم با توجه به سناریوی پیش آمده مقدار می­گیرد. |

پیاده سازی این توابع در فایل server.cpp آمده است که در بخش بعد به توضیح آن میپردازیم.

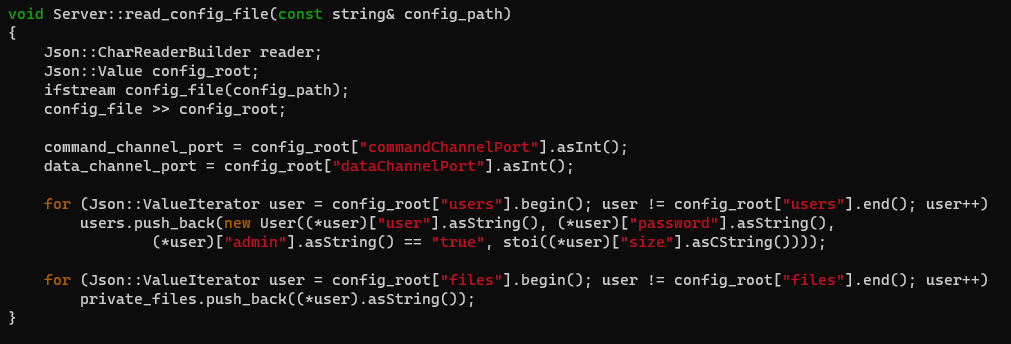
سایر فیلدهای این کلاس به شرح زیر هستند:



|  |  |
| --- | --- |
| DEFAULT\_LISTEN\_QUEUE\_SIZE | اندازه صف که در تابع listen استفاده میشود |
| DEFAULT\_BUFFER\_SIZE | اندازه دیفالت بافرها که در طول کد ساخته میشود |
| server\_files\_path | آدرس فایل­های سرور که یکی از آرگومان­های برنامه بود |
| private\_files | لیستی از فایل­های پرایویت که در فایل کانفیگ داده شده بود |
| users | لیست کاربرانی که اطلاعات آنها در فایل کانفیگ داده شده بود |
| logger | یک نمونه از کلاس Logger که از آن برای ثبت لاگ سیستم استفاده میشود |
| command\_address | آدرس کانال دستور |
| data\_address | آدرس کانال داده |
| config\_path | آدرس فایل کانفیگ که یکی از آرگومان­های برنامه بود |
| command\_socket\_descriptor | سوکت مربوط به کانال دستور |
| data\_socket\_descriptor | سوکت مربوط به کانال داده |
| command\_channel\_port | پورت مربوط به کانال دستور |
| data\_channel\_port | پورت مربوط به کانال داده |

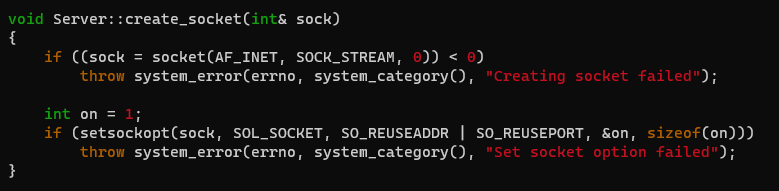
### فایل server.cpp:

تابع read\_config\_file:



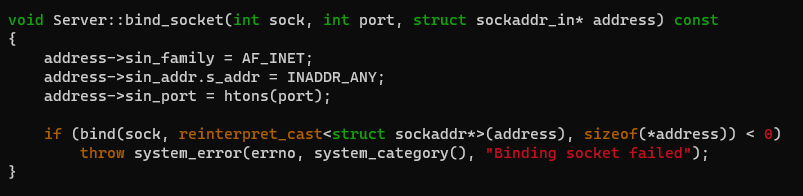
در این تابع قصد داریم تا اطلاعات موجود در فایل کانفیگ را استخراج کنیم. برای این کار از کتابخانه jsoncpp استفاده شده که کدهای آن در مسیر src/lib/json قرار دارد و از طریق makefile کتابخانه مربوط به آن ساخته می­شود. این تابع مسیر فایل کانفیگ را به عنوان آرگومان دریافت می­کند و سپس با توابع خود این فایل را میخواند و در یک مپ ذخیره میکند. با استفاده از این مپ مقادیر command\_channel\_port و data\_channel\_port و وکتورهای users و private\_files پر شده است و این مقادیر از فایل کانفیگ استخراج می­شود.

تابع create\_socket:



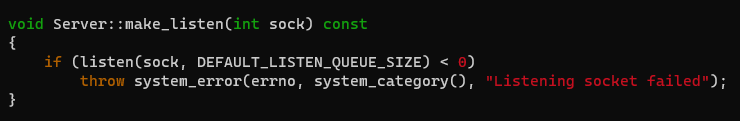
در این تابع ابتدا یک سوکت tcp ساخته میشود. در صورت عدم موفقیت در ساخت این سوکت یک استثنا پرتاب میشود. بعد از ساخت سوکت آپشنی بر روی سوکت تنظیم میشود برای اینکه بتوان از آدرس و پورت مربوطه دوباره استفاده کرد. این مورد برای وقتی است که بر روی یک سیستم چندین سرور داشته باشیم. اگر این عملیات ناموفقیت بود یک استثنا پرتاب میشود. توجه کنید سوکتی که ساخته شده به آرگومان تابع نسبت داده شده است.

تابع bind\_socket:



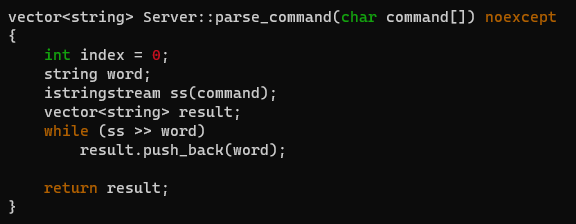
در این تابع ابتدا ساختار مربوط به آدرس داده شده در آرگومان پر میشود و بعد از آن سوکت داده شده در آرگومان به آن آدرس bind میشود. توجه کنید پورت مربوط به آدرس در آرگومان داده شده است و عملیات مقداردهی در این تابع انجام میشود. در صورتی که bind موفقیت آمیز نبود یک استثنا پرتاب می­شود.

تابع make\_listen:



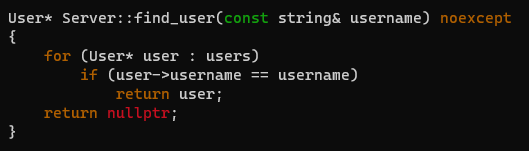
در این تابع، تابع listen بر روی سوکتی که در آرگومان داده شده صدا زده می­شود و در صورت عدم موفقیت یک استثنا پرتاب می­شود.

تابع parse\_command:



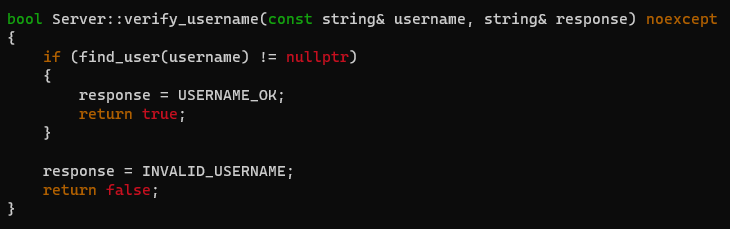
در این تابع ابتدا یک stringstream از ورودی ساخته میشود و با خواندن از آن و اضافه کردن رشته خوانده شده به یک وکتور نتیجه محاسبه میشود. توجه کنید کار این تابع این است که دستور را از حالت رشته به یک وکتور از آرگومان­ها تبدیل کند و این وکتور را برگرداند.

تابع find\_user():



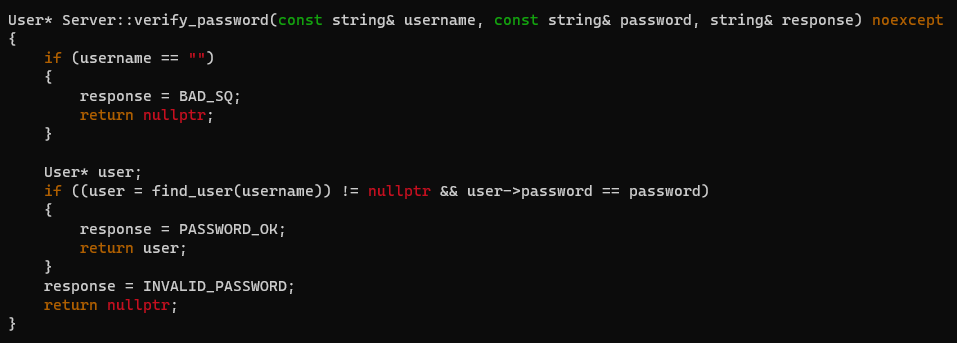
این تابع یک نام کاربری در ورودی می­گیرد و در وکتور مربوط به یوزرها به دنبال این نام کاربری میگردد. در صورتی که کلاس مربوط به آن پیدا شد آن کلاس را برمیگرداند و در غیر این صورت مقدار نال بازگردانده میشود.

تابع verify\_username():



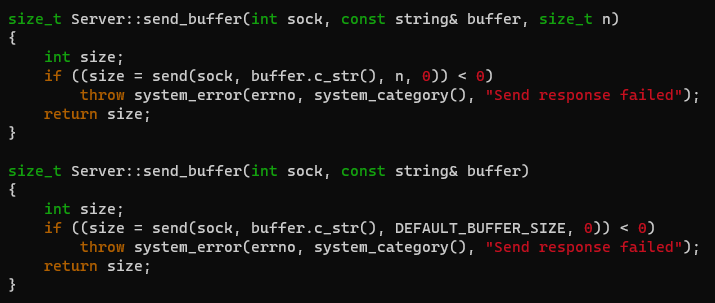
این تابع یک نام کاربری و یک رفرنس استرینگ دریافت میکند. این تابع در ابتدا به دنبال نام کاربری میگردد و در صورتی که نام کاربری یافت شد مقدار response را برابر USERNAME\_OK (یکی از همان ماکروهایی که قبلا اشاره شد) قرار میدهد و true بازمیگرداند. در صورتی که نام کاربری یافت نشد مقدار response برابر INVALID\_USERNAME قرار داده میشود و مقدار false بازگردانده میشود.

تابع verify\_password():



این تابع یک نام کاربری و رمز عبور به همراه رفرنس یک رشته که باید نتیجه در آن ریخته شود دریافت میکند. در صورتی که نام کاربری مقدار خالی باشد بدان معناست که کاربر هنوز نام کاربری را وارد نکرده پس مقدار response برابر BAD\_SEQUENCE قرار داده میشود و مقدار نال را برمیگرداند. بعد از آن به جستجوی نام کاربری پرداخته میشود و در صورتی که کاربر مربوط به آن رمز عبور درستی وارد کرده بود response برابر PASSWORD\_OK قرار داده میشود و کاربر مربوط به عنوان مقدار بازگشتی بازگردانده میشود. در صورتی که نام کاربری و رمز عبور مطابقت نداشتند INVALID\_PASSWORD در response ریخته میشود و مقدار نال بازگردانده می­شود.

توابع send\_buffer():



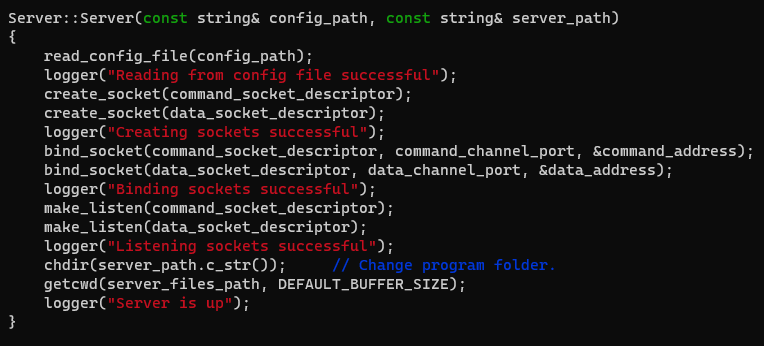
در این توابع یک سوکت و یک رشته گرفته میشود و رشته مربوط به سوکت داده شده ارسال میشود. در صورت عدم موفقیت یک استثنا پرتاب میشود و در صورت موفقیت تعداد بایت­هایی که به درستی ارسال شده است بازگردانده میشود. تفاوت این دو تابع در آرگومان سوم است که در یکی وجود دارد و بیان میکند چه تعداد بایت در تابع send ارسال شود. در تابع دیگر به مقدار پیش فرض، که در declaration کلاس تعریف شده، ارسال انجام می­پذیرد.

تابع accept\_connection():



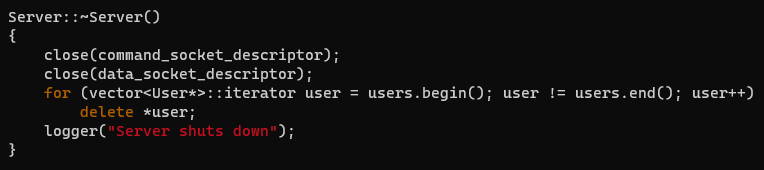
در این تابع دوبار تابع accept صدا زده میشود. در یکی از آنها سوکت دستور accept میشود و در دیگری سوکت داده accept میشود. در صورت عدم موفقیت در هر کدام از آنها یک استثنا پرتاب میشود و هنگامی که هر دوی آنها با موفقیت accept شدند logger یک پیغام مبنی بر اینکه یک کاربر جدید با موفقیت وارد شد ثبت میکند و در نهایت سوکت های accept شده بازگردانده میشود.

کانستراکتور Server():



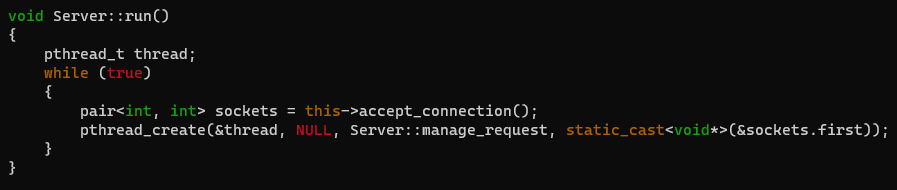
در کانستراکتور این تابع ابتدا اطلاعات از فایل کانفیگ خوانده می­شود (آدرس فایل کانفیگ قبل­تر توسط آرگومان­های خط فرمان به برنامه داده شده است) و پس از آن سوکت­های داده و دستور ایجاد میشود. بعد از آن به آدرس و پورت مربوط به خودشان bind میشود. سپس هر دوی سوکت­ها شروع به listen میکنند. در نهایت محل اجرای پردازه سرور به آدرس داده شده در آرگومان (که قبل­تر توسط آرگومان­های خط فرمان به تابع داده شده است) تغییر میکند و این محل در فیلد این کلاس به نام server\_files\_path (که قبل تر توضیح داده شد) ذخیره میشود. یعنی با استفاده از تابع chdir محل اجرای پردازه عوض میشود و با تابع getcwd این محل دریافت شده و در فیلد داده شده ذخیره میشود. در میان تمام این مراحل به تناسب کاری که انجام شده لاگ­های مناسب ذخیره می­شود.

دیستراکتور ~Server():



در ابتدا سوکت­های داده و دستور بسته میشوند. سپس حافظه­هایی که جهت ذخیره سازی اطلاعات کاربران (در تابع read\_config\_file) تخصیص پیدا کرده است آزاد میشود و در انتها لاگ مربوط ثبت می­شود.

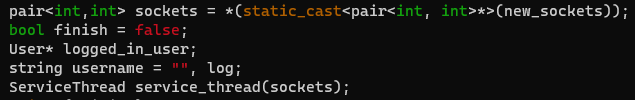
تابع run():



در این تابع یک حلقه بی­نهایت وجود دارد که در آن ابتدا یک کانکشن اکسپت می­شود و بعد از آن یک thread برای رسیدگی به این کانکشن اختصاص می­یابد. تابعی که مخصوص رسیدگی به thread است manage\_request نام دارد که در ادامه به توضیح آن می­پردازیم. به thread ساخته شده سوکت اکسپت شده را نیز پاس می­دهیم.

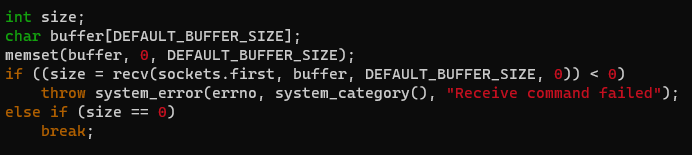
تابع manage\_request():

در ابتدا متغیرهای مورد نیاز تعریف می­شود.



در ابتدا سوکت­هایی که به عنوان ورودی داده شده است به نوع صحیح cast میشود. بعد از آن متغیرهای مورد نیاز تعریف شده و در نهایت یک نمونه از کلاس ServiceThread ایجاد می­شود که در آینده به توضیح آن خواهیم پرداخت.

بعد از تعریف متغیرها یک حلقه وجود دارد که تا وقتی پرچم finish برابر false است کارهای زیر انجام می­شود:

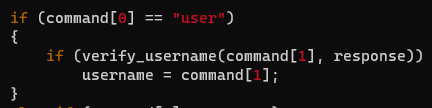


ابتدا از سوکت اول که سوکت دستور است دستوری که کاربر وارد میکند دریافت می­شود. در صورتی که تعداد بایت­های دریافت شده برابر صفر باشد از حلقه خارج میشویم.

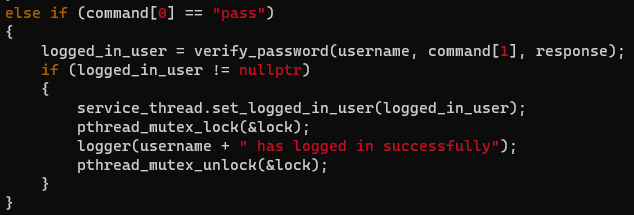


بعد از دریافت دستور نوبت به پارس کردن آن میرسد و یک رشته تعریف می­شود که نتیجه رسیدگی به این دستور است.

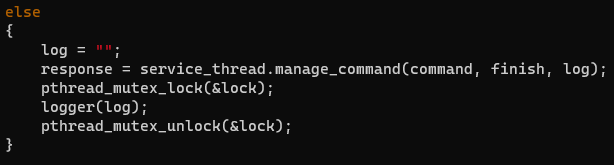
حال با توجه به نوع دستوری که وارد شده تصمیمات مختلفی اتخاد می­شود که به شرح زیر است.



اگر نوع دستور user باشد تابع verify\_username صدا زده می­شود و در صورت موفقیت آمیز بودن نام کاربری وارد شده در متغیر username ریخته می­شود.



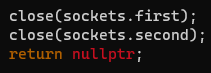
در صورتی که نوع دستور pass باشد نوبت به چک کردن صحت رمز عبور با تابع verify\_password میرسد. در صورتی که ورود موفقیت آمیز بود (یعنی مقدار خروجی مخالف نال بود) در نمونه service\_thread کاربر وارد شده ست میشود و بعد از آن در فایل لاگ پیامی مبنی بر اینکه ورود کاربر موفقیت آمیز بود ثبت میشود. توجه کنید چون ممکن است چندین کاربر (و بالتبع چندین thread) در حال نوشتن در فایل باشد یک قفل ایجاد شده که اینجا استفاده می­شود.



در صورتی که دستور وارد شده هیچ کدام از این دو دستور نبود از نمونه service\_thread استفاده می­کنیم. ابتدا متغیر لاگ ریست میشود. بعد از آن تابع manage\_command از این نمونه صدا زده میشود. پس از آن لاگی که توسط این تابع تولید شده است توسط logger ثبت میشود. توجه کنید به دلایلی که ذکر شد از قفل استفاده شده است. تابع manage\_command به عنوان ورودی دستور وارد شده، پرچم پایان و رشته لاگ را دریافت میکند و متغیرهای finish و log را مقداردهی میکند. همچنین به عنوان خروجی response متناظر با این دستور بازگردانده می­شود.



در نهایت response تولید شده از طریق سوکت دستور به کلاینت بازگردانده میشود.



هنگامی که از حلقه خارج می­شویم سوکت­ها را میبندیم و از thread خارج می­شویم. توجه کنید این سوکت­ها، سوکت­هایی هستند که accept شده­اند و نه سوکت­های اصلی.

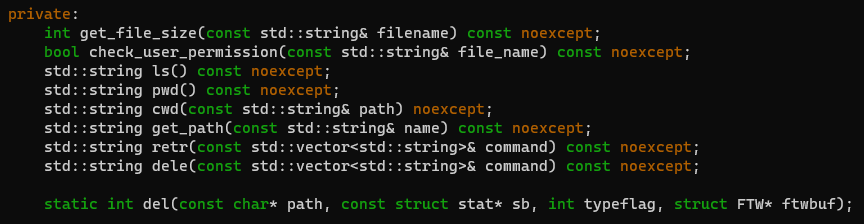
### فایل include/service\_thread.h:

این کلاس برای رسیدگی به یک thread به کار می­رود. متدهای عمومی این تابع به شرح زیر است:



|  |  |
| --- | --- |
| ServiceThread() | کانستراکتور این کلاس که یک جفت سوکت به عنوان ورودی می­گیرد. یکی از این سوکت­ها مربوط به دستور و دیگری مربوط به داده است. |
| set\_logged\_in\_user() | یک setter برای ست کردن logged\_in\_user که از فیلدهای این کلاس است. در ادامه این فیلد را خواهیم دید. |
| manage\_command() | برای مدیریت یک دستور که از سمت کلاینت ارسال شده است. ورودی این تابع دستور ارسال شده، یک پرچم برای پر کردن (این پرچم را قبلا دیدیم. اگر کار کلاینت تمام شود true می­شود) و یک رشته برای پر کردن که لاگ سرور است. خروجی این تابع یک رشته است که پاسخی است که در جواب دستور کلاینت. همانطور که دیدیم این پاسخ برای کلاینت ارسال می­شود. |

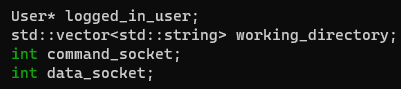
متدهای خصوصی نیز به شرح زیر هستند:



|  |  |
| --- | --- |
| get\_file\_size() | با دریافت نام فایل اندازه آن را در خروجی می­دهد. |
| check\_user\_permission() | نام یک فایل را دریافت میکند. اگر کاربر وارد شده اجازه دسترسی به فایل را داشت false و در غیر این صورت true برمیگرداند. |
| ls() | وقتی دستور ls ارسال شده باشد این تابع صدا می­شود. |
| pwd() | وقتی دستور pwd ارسال شده باشد این تابع صدا می­شود. |
| cwd() | وقتی دستور cwd ارسال شده باشد این تابع صدا می­شود. |
| get\_path() | با دریافت نام یک فایل، آدرسی که آن فایل در آن قرار دارد بازگردانده می­شود. توجه کنید این آدرس نسبت به فایل­های سرور داده می­شود. |
| retr() | وقتی دستور retr ارسال شده باشد این تابع صدا می­شود. |
| dele() | وقتی دستور dele ارسال شده باشد این تابع صدا می­شود. |
| del() | این تابع استاتیک هنگام پاک کردن دایرکتوری به کار میرود. در ادامه کاربرد این تابع را خواهیم دید. |

توجه کنید خروجی توابعی که متناظر با دستورات است response نسبت به آن دستور است.

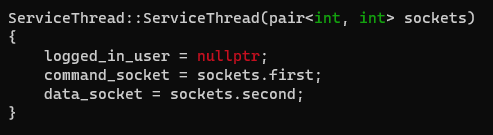
فیلدهای این کلاس نیز به شرح زیر است:



|  |  |
| --- | --- |
| logged\_in\_user | کاربری که وارد شده است در این فیلد نگهداری می­شود |
| working\_directory | محلی که در حال حاضر thread در آن مشغول کار است در این فیلد ذخیره می­شود. کارکرد دقیق­تر آن در ادامه توضیح داده خواهد شد |
| command\_socket | سوکت مربوط به کانال دستور |
| data\_socket | سوکت مربوط به کانال داده |

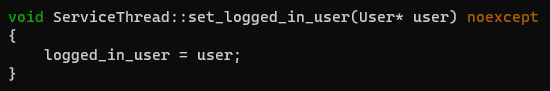
پیاده سازی این توابع در فایل service\_thread.cpp آمده است که در ادامه به توضیح آن می­پردازیم.

کاستراکتور ServiceThread():



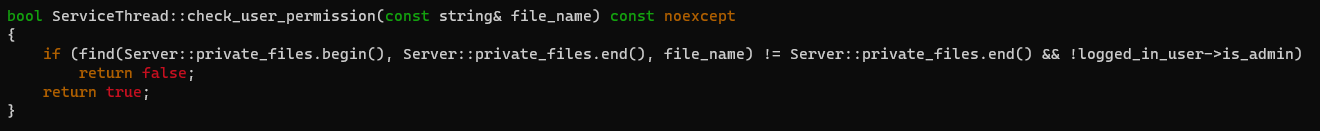
در کانستراکتور این کلاس فیلدهای logged\_in\_user و command\_socket و data\_socket مقداردهی می­شود.

تابع set\_logged\_in\_user():



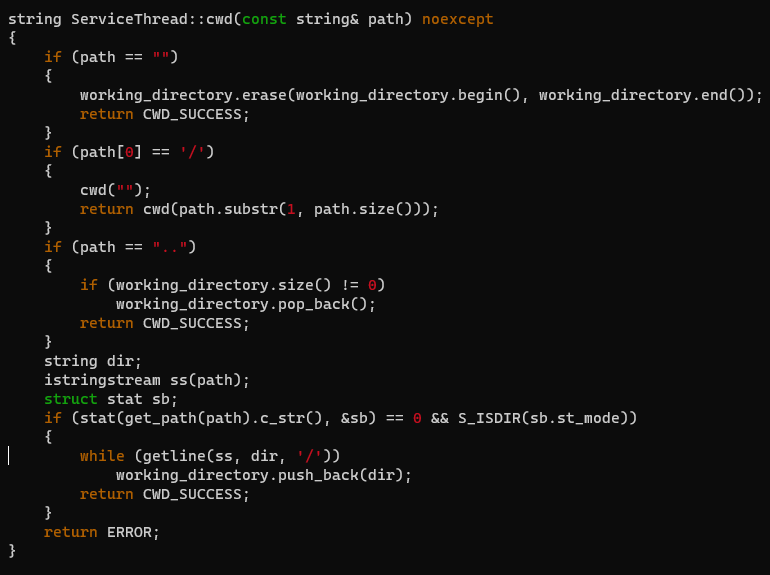
فیلد logged\_in\_user ست می­شود.

تابع check\_user\_permission():



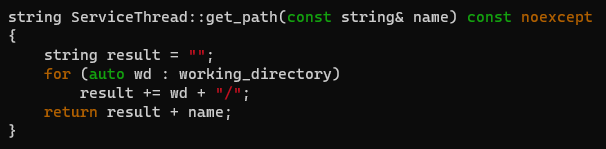
در این تابع ابتدا چک میشود فایل داده شده از فایل­های خصوصی هست یا خیر. اگر فایل خصوصی بود ادمین بودن کاربر وارد شده چک میشود. اگر این کاربر اجازه دسترسی به فایل را داشت false و در غیر این صورت true بازگردانده می­شود.

تابع cwd():



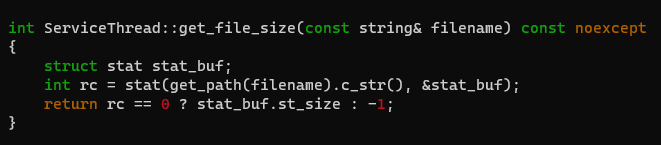
فیلد working\_directory مشخص میکند ما در چه مسیری هستیم. اگر وارد یک پوشه شویم نام آن پوشه به این وکتور اضافه می­شود و اگر یک پوشه به عقب برگردیم عضو آخر این وکتور pop میشود. با این منطق به توضیح این تابع می­پردازیم. اگر مسیر وارد شده خالی باشد یعنی باید به اولین پوشه بازگردیم پس working\_directory را خالی میکنیم و موفقیت را بازمیگردانیم. اگر در ابتدای مسیری که وارد شده / وجود داشت یعنی آدرس دهی از پوشه اولیه انجام شده است. پس ابتدا با صدا کردن cwd(“”) به پوشه اولیه بازمیگردیم و بعد از آن ادامه مسیر گفته شده را طی می­کنیم. اگر مسیر وارد شده .. باشد یعنی باید به پوشه قبلی بازگردیم. یعنی از working\_directory عضو آخر را pop میکنیم. (اگر خالی نباشد). اگر این وکتور خالی باشد یعنی در پوشه اولیه هستیم. در این صورت بازگشت به پوشه عقب ارور نمیدهد و صرفا اتفاقی نمی­افتد. این همان رفتاری است که سیستم عامل ubuntu 20.04 دارد. اگر هیچ کدام از حالات ذکر شده نبود، ابتدا چک میشود که مسیر وارد شده یک دایرکتوری است. پس از آن در یک حلقه فولدرهای آن به ترتیب جدا میشود و به working\_directory اضافه می­شود. در نهایت اگر فولدر وجود داشت پیام موفقیت و در غیر این صورت پیام شکست به عنوان response بازگردانده می­شود.

تابع get\_path():



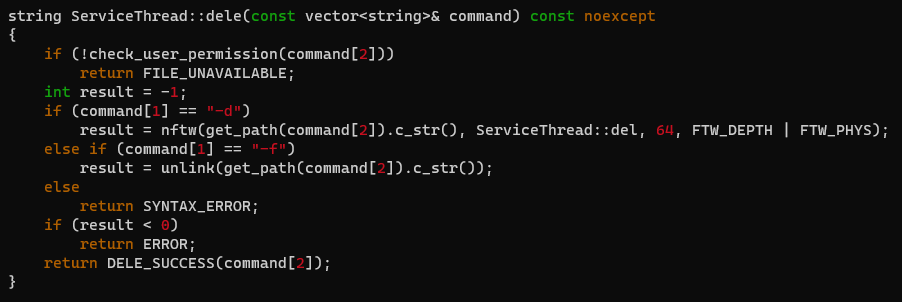
این تابع نام فایل را دریافت میکند و مسیر کامل این فایل نسبت به پوشه اولیه بازمیگرداند.

تابع get\_file\_size():



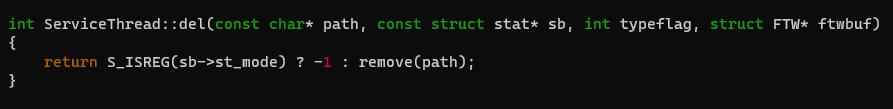
این تابع با دریافت نام فایل اندازه آن را بازمیگرداند. در صورتی که همچین فایلی وجود نداشت منفی یک بازگردانده می­شود. توجه کنید سایز به بایت است.

تابع dele():



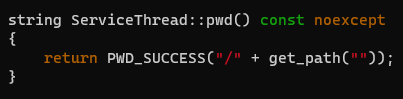
در این تابع ابتدا چک میشود کاربر اجازه دسترسی به فایل خواسته شده را دارد یا خیر. در صورت نداشتن دسترسی FILE\_UNAVALABLE به عنوان response بازگردانده میشود. بعد از آن به آرگومان دوم توجه میشود. اگر این آرگومان -d بود یعنی میخواهیم یک دایرکتوری را حذف کنیم. از تابع nftw برای حرکت روی فایل­ها استفاده میکنیم و تابع آن را del میدهیم که در ادامه این تابع توضیح داده خواهد شد. عمق حرکت را نیز 64 قرار میدهیم. اگر آرگومان دوم -f بود از تابع unlink استفاده میکنیم که با دریافت آدرس فایل آن را پاک میکند. اگر هیچ کدام از این دو مورد نبود SYNTAX\_ERROR را به عنوان response بازمیگردانیم و اگر هر کدام از توابع ذکر شده دچار ارور شدند ارور میدهیم. در غیر این صورت پیام موفقیت را به عنوان response باز میگردانیم.

تابع del():



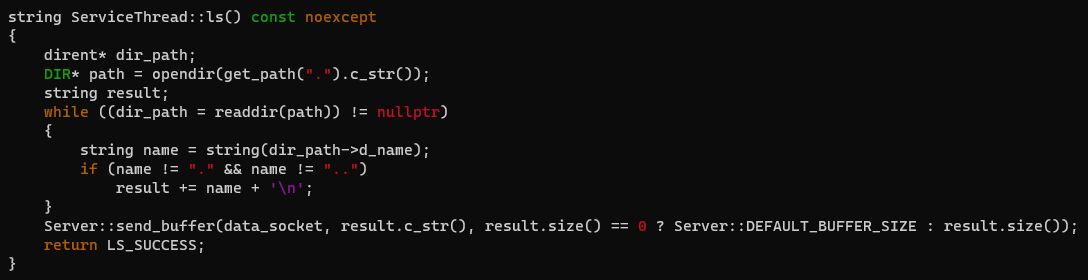
در این تابع چک میکنیم که مسیر داده شده مربوط به فولدر باشد. اگر مربوط به فولدر بود تابع remove را روی آن صدا میزنیم (این تابع مسیر داده شده را پاک میکند) و نتیجه آن را بازمیگردانیم و اگر این مسیر فولدر نبود منفی یک بازمیگردانیم.

تابع pwd():



در اینجا مسیری که در آن هستیم را بازمیگرداند. توجه کنید امکان خطا وجود ندارد.

تابع ls():



در این تابع با استفاده از تابع opendir دایرکتوری فعلی باز میشود و روی فایل­ها و فولدرهای آن پیمایش می­شود. در صورتی که آنها . یا .. نبودند به رشته نتیجه اضافه میشود. در نهایت نتیجه به دست آمده از طریق کانال داده برای کلاینت ارسال میشود و پیام موفقیت به عنوان response بازگردانده میشود.

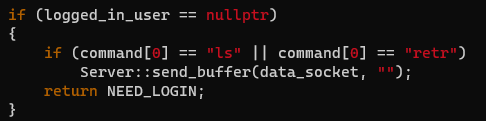
تابع retr():



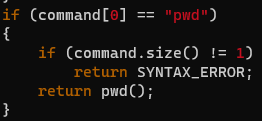
قرارداد بین کلاینت و سرور به این صورت است که اول اندازه فایل ارسال میشود و پس از آن فایل باز شده و قطعه قطعه خوانده میشود و هر قطعه برای کلاینت ارسال میشود. پس در ابتدا اندازه فایل را به دست می­آوریم. اگر این مقدار کمتر از صفر بود اول اندازه آن برای کلاینت ارسال میشود و بعد از آن ارور بازگردانده میشود. بعد از آن اجازه دسترسی بررسی میشود. اگر کاربر اجازه دسترسی به فایل را نداشت منفی یک برای آن ارسال میشود و ارور بازگردانده میشود. توجه کنید نیازمند این ارسال منفی یک هستیم. زیرا در سمت کلاینت تابع recv صدا شده و کلاینت منتظر دریافت است. اگر چیزی برای آن ارسال نشود بلاک میشود. بعد از اجازه دسترسی چک میکنیم آیا کاربر حجم مورد نیاز برای دانلود فایل را دارد یا خیر. در صورتی که حجمش کم بود DATA\_CAP\_ERROR به عنوان response بازگردانده میشود. بعد از آن حجم فایل از حجم کاربر کم میشود. فایل مورد نظر باز میشود و بلاک بلاک از آن خوانده میشود و در هر مرحله از خواندن بلاک خوانده شده ارسال میشود. در نهایت موفقیت به عنوان response بازگردانده میشود.

تابع manage\_command():

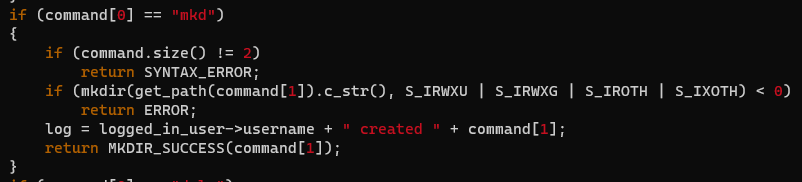
این تابع اصلی این کلاس است.



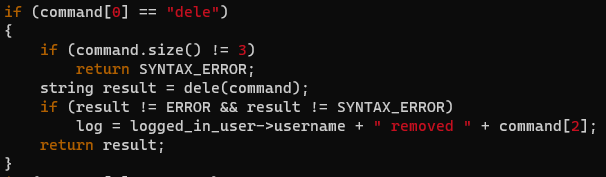
در ابتدای آن چک میشود آیا کاربری لاگین کرده است یا خیر. اگر کاربر لاگین نکرده بود خطای مربوطه بازگردانده میشود. توجه کنید یک رشته خالی در صورتی که دستور ls یا retr باشد از طریق کانال داده ارسال میشود تا کلاینت بلاک نکند.



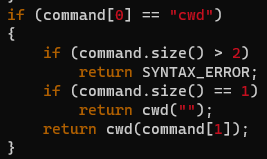
اگر دستور pwd باشد ابتدا تعداد آرگومان­ها چک میشود و در صورت نبود مشکل تابع pwd صدا زده میشود.



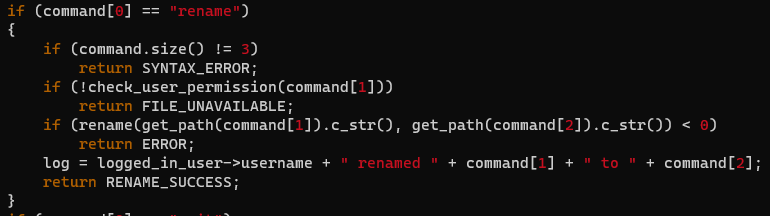
اگر دستور mkd باشد ابتدا تعداد آرگومان­ها چک میشود و در صورت نبود مشکل تابع mkdir با دسترسی­هایی که مشاهده میکنید صدا زده میشود. در صورت موفق بودن ساخت پوشه لاگ مناسب قرار داده میشود و پیامی مبنی بر موفقیت به عنوان response بازگردانده میشود.



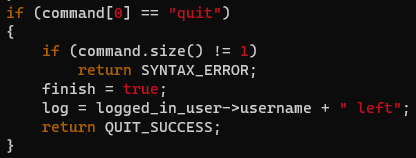
اگر دستور وارد شده dele باشد ابتدا تعداد آرگومان­ها چک میشود در صورت عدم وجود مشکل تابع dele صدا زده میشود در صورتی که خروجی این تابع هم مشکلی نداشت لاگ مناسب مقداردهی میشود.



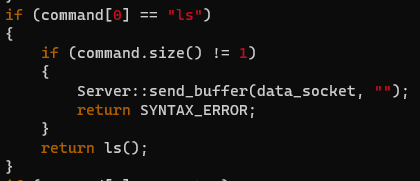
اگر دستور وارد شده cwd باشد ابتدا تعداد آگومان­ها چک میشود. اگر این دستور بدون آرگومان وارد شد تابع cwd با رشته خالی صدا زده میشود. در غیر این صورت تابع cwd با آرگومان داده شده صدا زده میشود.



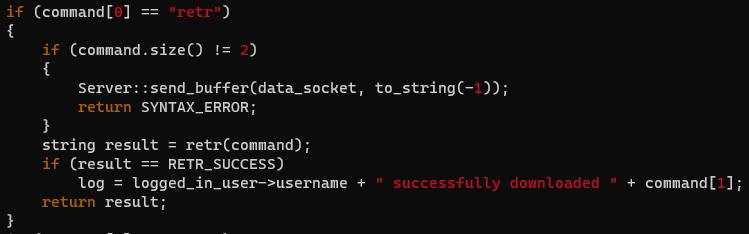
اگر دستور rename وارد شده باشد ابتدا تعداد آرگومان­ها چک میشود. بعد از آن دسترسی کاربر وارد شده چک میشود. اگر مشکلی وجود نداشت تابع rename صدا زده میشود و در صورت موفقیت این تابع لاگ مناسب تولید میشود. در انتها پیامی مبنی بر موفقیت بازگردانده می­شود. ارورهای مربوط به هر بخش نیز با توجه به کد مشخص است.



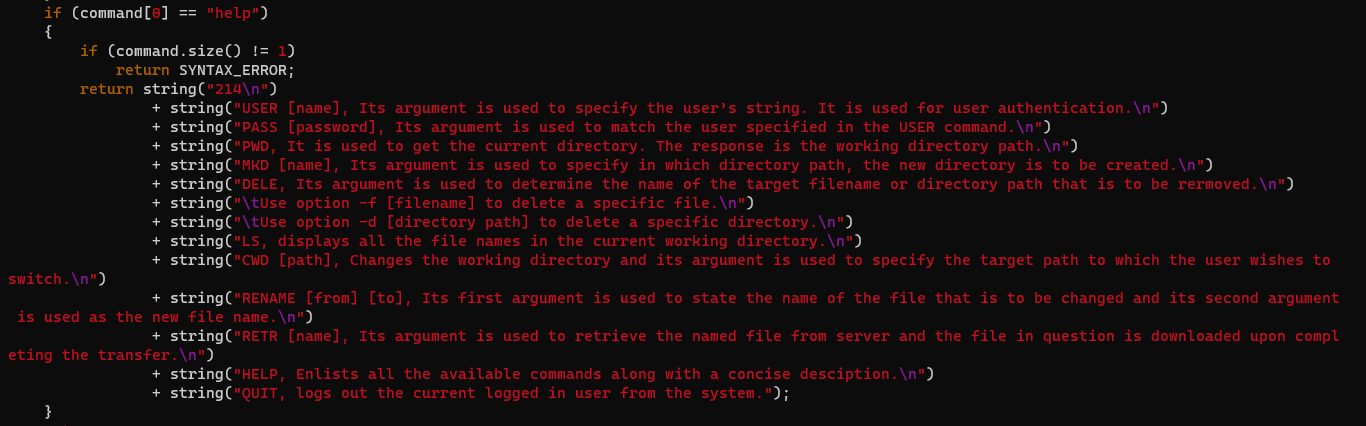
اگر دستور quit وارد شده باشد ابتدا تعداد آرگومان­ها چک میشود. پس از آن پرچم مربوط به تمام شدن thread برابر true می­شود و پس از آن لاگ مناسب تولید میشود.



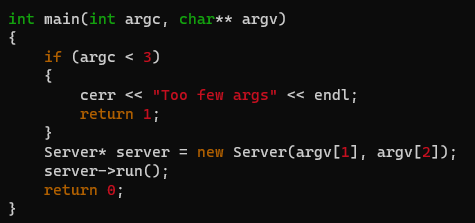
اگر دستور ls وارد شده باشد ابتدا تعداد آرگومان­ها چک میشود. در صورت عدم موفقیت یک پیام خالی از طریق کانال داده ارسال میشود تا کلاینت بلاک نشود. در صورت موفقیت تابع ls صدا زده میشود و خروجی تولید میشود.



اگر دستور retr وارد شده باشد ابتدا تعداد آرگومان­ها چک میشود. در صورت عدم موفقیت یک پیام خالی از طریق کانال داده ارسال میشود تا کلاینت بلاک نشود. در صورت موفقیت تابع retr صدا زده میشود و اگر این دستور موفقیت آمیز بود لاگ مناسب تولید میشود.

در صورتی که دستور وارد شده help باشد ابتدا تعداد آرگومان­ها چک میشود. بعد از آن پیامی تولید میشود که شرح کار سرور است و این پیام بازگردانده میشود.

### فایل server\_main.cpp

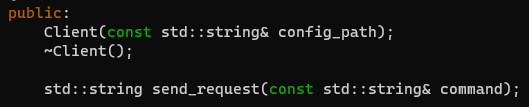


در این فایل main سرور قرار دارد. در ابتدا چک میشود تعداد آرگومان­های خط فرمان صحیح باشد و بعد از آن یک نمونه از سرور تولید میشود. بعد از آن هم متد run صدا زده میشود.

# کلاینت

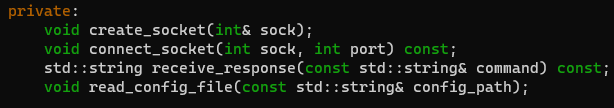
### فایل include/client.h:

این فایل شامل header کلاس کلاینت است. متدهای عمومی آن به شرح زیر است:



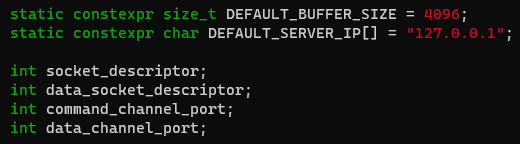
|  |  |
| --- | --- |
| Client() | کانستراکتور این کلاس که در ورودی آدرس فایل کانفیگ را میگیرد |
| ~Client() | دیستراکتور این کلاس |
| send\_request() | این تابع یک رشته میگیرد که همان دستوری است که باید برای سرور ارسال شود و نتیجه این دستور را به عنوان مقدار بازگشتی برمیگرداند |

متدهای خصوصی این کلاس به شرح زیر است:



|  |  |
| --- | --- |
| create\_socket() | این تابع یک رفرنس به عنوان ورودی میگیرد و یک سوکت باز میکند و سوکت دیسکریپتور تولید شده را به ورودی assign میکند |
| connect\_socket() | سوکت به آدرس متصل میشود |
| receive\_response() | این تابع پاسخ متناظر با دستور ارسال شده را از سرور دریافت میکند |
| read\_config\_file() | این تابع فایل کانفیگ را میخواند و فیلدهای مربوط را پر میکند |

فیلدهای این کلاس به شرح زیر هستند:

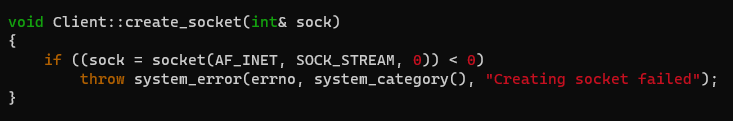


|  |  |
| --- | --- |
| DEFAULT\_BUFFER\_SIZE | اندازه بافر دیفالت که کاربردهای متعددی دارد |
| DEFAULT\_SERVER\_IP | آدرس پیش فرض سرور که لوکال هاست است |
| socket\_descriptor | سوکت مربوط به کانال دستور |
| data\_socket\_descriptor | سوکت مربوط به کانال داده |
| command\_channel\_port | پورت مربوط به کانال دستور |
| data\_channel\_port | پورت مربوط به کانال داده |

حال به بررسی پیاده سازی این توابع میپردازیم.

### فایل client.cpp:

تابع create\_socket:



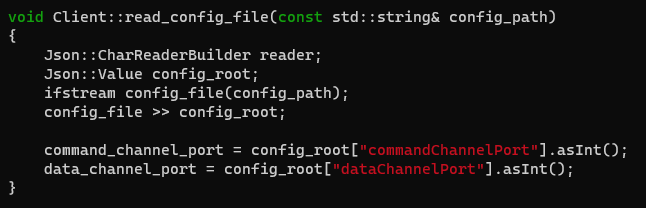
این تابع یک سوکت جدید میسازد و در صورت عدم موفقیت یک استثنا پرتاب میکند.

تابع conncet\_socket:



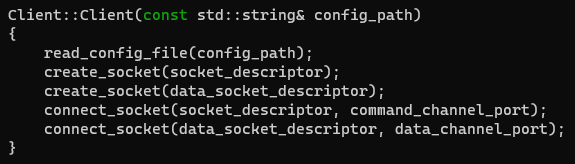
در ابتدا یک ساختار آدرس ساخته میشود و مقادیر آن ست می­شود. سپس به آدرس داده شده اتصال صورت میگیرد. در صورت عدم موفقیت یک استثنا پرتاب میشود.

تابع read\_config\_file:



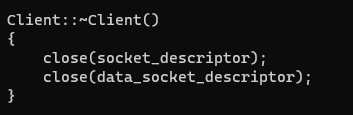
در این تابع فایل کانفیگ خوانده میشود (مانند سمت سرور) و پورت دو کانال داده و دستور از آن استخراج میشود.

کانستراکتور Client:

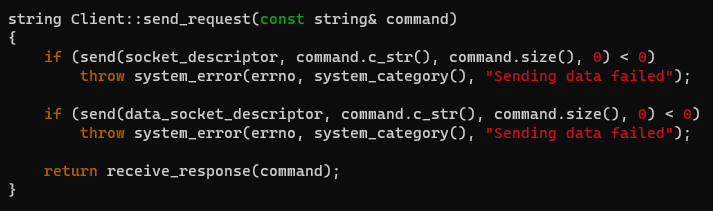


ابتدا فایل کانفیگ خوانده میشود. بعد از آن سوکت­های داده و دستور ایجاد میشوند و هردوی آنها متصل میشوند.

دیستراکتور ~Client:

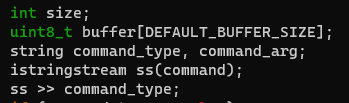


در دیستراکتور سوکت­های تولید شده بسته میشوند.

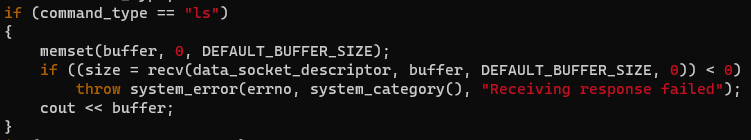


در این تابع ابتدا دستور داده شده به هر دو سوکت داده و دستور ارسال میشود. علت اینکه به هر دو ارسال میشود این است که در سمت سرور هر دو اکسپت شوند و بتوان پاسخ را در صورت نیاز به هر دو داد. (دستوراتی مانند ls از هر دو سوکت پاسخ ارسال میکنند). در انتها نیز تابع receive\_response صدا زده میشود تا پاسخ دستور ارسال شده از سرور دریافت شود و به عنوان مقدار بازگشتی بازگردانده شود.

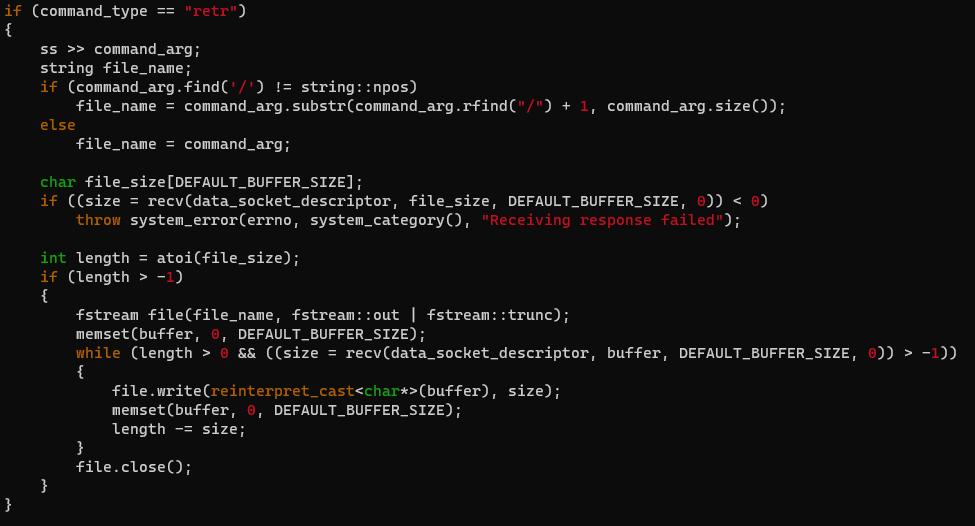
تابع receive\_response:



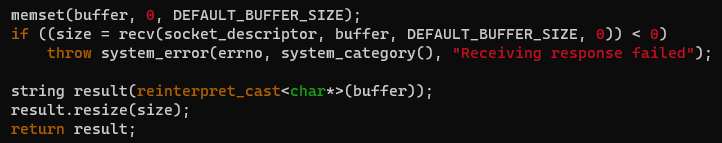
در ابتدای تابع تعدادی متغیر تعریف میشود. همچنین نوع دستور استخراج میشود و در متغیر command\_type ذخیره میشود.



اگر دستور ls بود لیست فایل­ها از کانال داده ارسال شده است. پس این قسمت پاسخ را از کانال داده میخواند و در صورت عدم موفقیت یک استثنا پرتاب میکند. بعد از دریافت پاسخ آن را چاپ میکند.

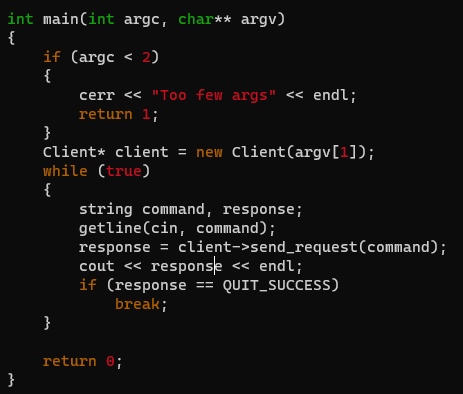


اگر دستور retr بود ابتدا اسم فایل استخراج میشود (توجه کنید پیاده سازی به صورتی است که میتوان مسیر یک فایل را برای دانلود در این دستور استفاده کرد) بعد از آن اندازه فایل که از سمت سرور ارسال شده دریافت میشود. حال اگر به خاطر داشته باشید وقتی این مقدار از منفی یک بیشتر است یعنی مشکلی پیش نیامده است. پس یک فایل با نام فایل خواسته شده ایجاد میشود و هر بلاک از داده که از سمت سرور ارسال میشود در آن ذخیره میشود. در نهایت فایل ذخیره میشود. توجه کنید هنگامی دریافت داده به اتمام میرسد که تعداد بایت­های خوانده شده با اندازه فایل برابر باشد. (هر دفعه تعداد بایت­های خوانده شده از اندازه کم میشود تا به صفر برسد)



در نهایت نیز پاسخ به دستور از کانال دستور خوانده میشود و در صورت عدم موفقیت استثنا پرتاب میشود. این پاسخ به صورت یک رشته در آورده میشود و به عنوان مقدار بازگشتی بازگردانده میشود.

### فایل client\_main.cpp



در این فایل main کلاینت وجود دارد. ابتدا چک میشود که آرگومان­های خط فرمان به تعداد باشد. سپس یک کلاینت با آدرس فایل کانفیگ داده شده در خط فرمان ساخته میشود. در یک حلقه بی نهایت از ورودی دستور خوانده میشود و تابع send\_request صدا زده میشود. پاسخ این تابع به عنوان نتیجه دستور به کاربر نمایش داده میشود. همچنین اگر پاسخ خروج موفقیت آمیز بود از حلقه خارج میشویم و اجرای برنامه خاتمه می­یابد.