گزارش پروژه اول شبکه

ملیکا مرافق 810197581

نازنین یوسفیان 810197610

برنامه سرور :

ابتدا در فایل main.cpp یک instance از کلاس سرور ساخته شده و تابع start() آن فراخوانی می شود.

در constructor سرور یک instance از کلاس RequestHandler ساخته می شود که پوینتر به سرور دارد. هم چنین دایرکتوری فعلی که سرور در آن است ست می شود و زمان کنونی نیز ثبت می شود.

در سرور یک instance از کلاس دیتابیس داریم و در تابع start() ابتدا آن را ران می کنیم که اطلاعات مورد نیاز از فایل config.json توسط کتابخانه jsoncpp خوانده شود. به این منظور متد start() از کلاس دیتابیس اجرا می شود.

برای ذخیره اطلاعات کاربران تابع addUserInf() فراخوانی می شود که یک مپ از کاربران می سازد به این صورت که کلید آن نام کاربر و مقدار آن اطلاعات کاربر است که خود آن یک مپ از string به string است به همان صورتی که در فایل config آمده. یک set از فایل هایی که نیاز به دسترسی ادمین دارند نیز در تابع addFiles() ساخته می شود. پس از اتمام این فرآیند دو متد دیگر دیتابیس توسط سرور فراخوانی می شوند که پورت های داده و دستور مشخص شود. توابع getCommandPort() و getDataPort() بدین منظور در دیتابیس پیاده سازی شده اند. سپس سوکت های مربوطه در تابع های createDataSocket() و createCommandSocket() ساخته می شوند. برای اینکه سرور بتواند چند اتصال همزمان را مدیریت کند از select() استفاده شده است.

createCommandSocket

ابتدا با socket() یک سوکت را ایجاد می‌کنیم .

نوع آدرسی را که سوکت میتواند با آن ارتباط برقرار کند (حوزه ارتباطی) مشخص می‌کند .
(address_family) را برابر AF_INET (پروتکل اینترنت (IPv4)) قرار می دهیم. و نوع ارتباط آن را TCP (stream socket) قرار می دهیم.

شماره پورت را برابر پورت دستور قرار می‌دهیم(commandChannelPort از فایل config)

INADDR_ANY سوکت را به تمام رابط (interface) های موجود bind می‌کند.

با تابع bind سوکت را به آدرس و شماره پورت دستور bind می‌کنیم.

سپس گوش می‌دهیم تا client برای برقراری ارتباط با سرور درخواست دهد.حداکثر طول صف درخواست های در انتظار ارتباط با سوکت سرور را برابر ۵ قرار می‌دهیم.

serviceClients

Client_socket حاوی مقادیر سوکت دیسکریپتور هاست که در ابتدا مقدار اولیه صفر گرفته اند.

در یک حلقه ی نا متناهی ابتدا به readfds که مجموعه ای از socket descriptorها است توسط تابع FD_ZERO مقدار اولیه صفر می‌دهیم. سپس با FD_SET دیسکریپتور سوکت سرور را به این مجموعه اضافه می‌کنیم. سپس تابع addTofdSet را فراخوانی می کنیم که بیشترین مقدار سوکت دیسکریپتور را برمی گرداند.سپس select را فراخوانی می‌کنیم که این مجموعه از سوکت ها و بیشترین مقدار سوکت دیسکریپتور را دریافت می کند و منتظر می ماند تا یک فعالیتی در یکی از سوکت ها رخ دهد. timeout را null قرار می دهیم تا مدت انتظار آن نا محدود باشد.

سپس در صورتی که مقدار file descriptor سوکت سرور در readfds ست شده باشد (توسط FD_ISSET بررسی می کنیم) یعنی یک اتصال ورودی داریم پس تابع newConnection() را فراخوانی می کنیم در غیر این صورت عملیات IO بر روی دیگر سوکت ها رخ داده و تابع socketIO() فراخوانی می شود. در صورتی که این تابع true برگرداند تابع dataTransfer() فراخوانی می شود که در ادامه توضیح داده شده است.

addTofdSet

در یک حلقه در صورتی که سوکت ایجاد شده بود (مقدار سوکت دیسکریپتور بزرگتر از صفر بود) آن را به مجموعه ی readfds (شامل socket descriptor) اضافه می کنیم. در این حلقه بیشترین مقدار سوکت دیسکریپتور محاسبه شده و ریترن می شود.

newConnection

connection را accept می کنیم که یک socket descriptor جدید برمی گرداند.آن را در اولین عنصری از client_socket که هنوز مقدار نگرفته ذخیره می کنیم.

socketIO

با حلقه بر روی socket_clientها اگر مقدار عنصر فعلی در مجموعه socket descriptor ها وجود داشت پیام دریافتی آن را می خوانیم و متغیر curId را برابر با این socket descriptor قرار می دهیم تا بدانیم کدام کلاینت پیامی فرستاده که جواب را ارسال کنیم. اگر read صفربرگرداند یعنی به انتهای فایل رسیدیم و ارتباط قطع شده پس سوکت را می بندیم و مقدار آن را در client_socket صفر می کنیم. در این حالت تابع مقدارtrue بر می گرداند.در غیر این صورت در انتهای رشته Null قرار می دهیم و متد tokenizer() را صدا می زنیم. هم چنین در این حالت تابع true بر می گرداند.

کانال داده:

همانطور که برای کانال دستور توضیح داده شد، همین فرآیند را برای ساختن کانال داده استفاده کرده و آنجا نیز از select() استفاده می کنیم اما فراخوانی های مربوط به این کانال نیز در همان while(true) که در تابع serviceClients() است صورت می گیرد. در حالتی که کلاینت جدیدی متصل شود، تابع dataTransfer() نیز فراخوانی می شود تا کاربر به کانال داده نیز متصل گردد. برای ادامه کار برای اینکه بتوانیم هم در کانال داده هم دستور برای کاربر پیام ارسال کنیم، یک مپ به نام ports ساخته شده که کلید آن socket descriptor برای کانال دستور و مقدار آن socket descriptor برای کانال داده است که به این در تابع newDataConnection اضافه می شود. هم چنین در حالتی که کلاینتی disconnect شود (همان حالتی که socketIO مقدار true برمی گرداند) تابع dataTransfer دوباره فراخوانی شده و این بار datasocketIO() فراخوانی می شود که curId را از مپ ports حذف می کند.

هرگاه که دستور جدیدی از سمت کاربر به سرور بیاید (در تابع socketIO()) ، متد tokenizer() از کلاس RequestHandler صدا زده می شود که پیام را بر اساس space جدا کرده و در یک وکتور ذخیره می کند. سپس متد getCommand() از همین کلاس فراخوانی می شود که متناسب با هر دستور، تابع متناظر آن در سرور را صدا می زند تا عملیات لازم برای اجرای دستور صورت گیرد. قبل از آن باید چک شود که درخواست وارد شده از سمت کاربر ارور نداشته باشد و درخواست درستی باشد. تابع reqExist() چک می کند که درخواست وارد شده جزو درخواست های تعریف شده باشد و تعداد آرگومان های دستور نیز درست باشد.

به ازای هر دستور به غیر از user و pass ابتدا تابع checkLog() در سرور فراخوانی می شود که مشخص شود کاربر وارد سیستم شده و مجاز به وارد کردن دستورات هست یا خیر.

در ادامه هر دستور به تفکیک توضیح داده شده است (نام تابع متناظر هر دستور در سرور با نام دستور یکسان است) :

:User

برای نگهداری اطلاعات هر کاربر کلاسی به نام User تعریف شده که در آن نام، پسورد و دایرکتوری که کاربر در آن است ذخیره می شود. یک مپ به نام users در سرور داریم که کلید آن شماره کاربری است که در حال تبادل اطلاعات به سرور است(curId) که در تابع socketIO() مقدار دهی شده است و مقدار آن instance از کلاس کاربر است.

ابتدا باید چک شود که کاربری که این دستور را وارد کرده از قبل در سیستم نباشد به همین منظور تابع alreadyLogged() فراخوانی می شود که چک می کند در مپ کلید curId وجود نداشته باشد. اگر این شرط برقرار بود باید چک شود که نام کاربری صحیح است. متد findUser() از کلاس دیتابیس فراخوانی می شود که چک می کند در مپی که از کاربران ساخته ایم کلید با این نام وجود داشته باشد. در صورت وجود یک User جدید تعریف می شود که دایرکتوری فعلی را همان دایرکتوری سرور قرار می دهیم و این کاربر را به مپ users اضافه می کنیم. برای اینکه مشخص شود کاربر هنوز رمز را وارد نکرده، یک فیلد با نام logged در کلاس User وجود دارد که در ابتدا مقدار false را دارد.

:Pass

در اینجا نیز ابتدا باید چک شود که کاربر وارد سیستم نشده باشد. سپس باید چک شود که کاربر نام کاربری خود را وارد کرده است. پس در مپ users به دنبال این کاربر با کلید curId می گردیم. قدم بعدی این است که چک کنیم پسورد وارد شده صحیح باشد. برای این کار پسورد صحیح را از تابع getPassword() از دیتابیس می گیریم و با پسورد وارد شده مقایسه می کنیم. اگر یکی بودند، فیلد logged در کلاس User مربوطه را true می کنیم و به این معنی است که کاربر وارد سیستم شده است.

همان طور که گفته شد برای دستورات بعدی ابتدا تابع checkLog() فراخوانی می شود که چک می کند در users کلید curId وجود داشته باشد و logged برابر با true باشد.

:Pwd

دایرکتوری ای که در User ذخیره شده است را با فراخوانی تابع getDir() در این کلاس می گیریم.

:Mkd

در اینجا فرض شده مسیری که کاربر وارد می کند در ادامه مسیر سرور است. (یعنی برای ساختن دایرکتوری a در سرور فقط a را به عنوان ارگومان وارد می کند.)

در ادامه بررسی می شود که دایرکتوری با موفقیت ساخته شده باشد و پیغام مناسب را ارسال می کند.

:Ls

ابتدا دایرکتوری ای که کاربر در آن قرار دارد گرفته می شود. سپس با استفاده از struct dirent نام فایل های درون دایرکتوری را به دست می آوریم. ابتدا در کانال دستور یک پیغام success میفرستیم به این معنا که عملیات موفقیت آمیز بوده و کاربر می تواند لیست فایل ها را از کانال داده دریافت کند. در انتها پیام متناظر نیز در کانال دستور ارسال می شود.

:Cwd

اگر دستور بدون آرگومان وارد شده باشد، دایرکتوری سرور به عنوان دایرکتوری فعلی کاربر با استفاده از فراخوانی تابع setDir() در کلاس User ذخیره می شود.

اگر ارگومان برابر با .. باشد، دایرکتوری فعلی کاربر را می گیرد و از آخرین / به بعد را حذف کرده و به عنوان دایرکتوری جدید ثبت می کند. البته اگر دایرکتوری فعلی کاربر همان دایرکتوری سرور باشد این عملیات ممکن نیست و ارور می دهد.

در بقیه حالات نیز چک می کند که دایرکتوری وجود داشته باشد و پیام مناسب را ارسال می کند.

:Dele

:-d

با دستور rmdir دایرکتوری را حذف می کنیم و اگر موفقیت امیز بود دستور مناسب را ارسال می کنیم و در غیر این صورت ارور ارسال می شود.

:-f

ابتدا تابع hasAccess() فراخوانی می شود تا چک شود که کاربر مجاز به دسترسی به فایل است. اگر دایرکتوری فعلی کاربر برابر با دایرکتوری سرور بود ابتدا چک می شود که فایل جزو فایل هایی است که دسترسی ادمین را نیاز دارد یا خیر. پس تابع restrictedFile() از دیتابیس فراخوانی می شود که در لیست فایل های ذخیره شده می گردد. اگر true برگردانده شد تابع isAdmin() از این کلاس فراخوانی می شود که اسم کاربر را داده و چک می کنیم ادمین هست یا نه. اگر نبود پیام متناسب ارسال می شود. در اگر کاربر دسترسی مجاز داشت تابع removeFile() از سرور فراخوانی می شود و با استفاده از remove() فایل را حذف می کنیم و اگر موفقیت آمیز بود دستور متناظر و در غیر این صورت ارور ارسال می شود.

:Rename

در اینجا نام تابعی که در سرور فراخوانی میشود changeName است. برای اجرای این دستور نیز ابتدا تابع hasAccess() فراخوانی می شود و سپس اگر کاربر مجاز بود با استفاده از rename() نام فایل را تغییر می دهیم و در صورت بروز مشکل پیغام ارور برای کاربر ارسال می شود.

:Retr

در اینجا هم اجازه دسترسی به فایل بررسی می شود و سپس تابع downloadFile() فراخوانی می شود.

ابتدا چک می شود که فایل موجود باشد. در غیر این صورت ارور ارسال می شود. سپس باید بررسی شود کاربر حجم کافی دارد یا خیر. با استفاده از فراخوانی تابع checkTraffic() در سرور ابتدا حجم باقی مانده کاربر را با استفاده از تابع getUserTraffic() دردیتابیس به دست می آوریم. سپس حجم فایل را محاسبه کرده و این دو را مقایسه می کنیم. اگر حجم کاربر کافی بود حجم جدید را برای او ست می کنیم و true بر می گردانیم.

در صورتی که کاربر حجم کافی نداشته باشد پیام متناسب ارسال می شود. در غیر این صورت ابتدا در کانال دستور پیغام success فرستاده می شود و سپس فایل از طریق کانال داده و در انتها دستور مناسب برای کاربر ارسال می شود.

:Help

به ازای هر دستور توضیحات لازم را نوشته و برای کاربر در کانال دستور ارسال می کنیم.

:Quit

curId را از مپ users حذف کرده و پیغام مناسب را برای کاربر ارسال می کنیم.

تابع printLog() در سرور برای ثبت اطلاعات کاربران و کارهایی که کرده اند تعریف شده است.
تاریخ، ساعت و پیام مناسب در فایل log.txt ثبت می شود. اطلاعاتی که ثبت می شوند شامل:

ورود کاربر، ساختن دایرکتوری جدید، حذف فایل یا دایرکتوری، تغییر نام فایل، دانلود فایل و خروج است.

برنامه کلاینت:

در کلاینت ابتدا از فایل config.json کانال های داده و دستور مشخص می شوند و به سرور از طریق این دو کانال وصل می شود. سپس تابع communicate() اجرا می شود. در این تابع از کنسول دستور خوانده می شود و برای سرور ارسال می شود. پس از خواندن پاسخ از سرور آن را در کنسول چاپ می کند. دو دستور ls و retr متفاوت هستند و برای هرکدام تابع جداگانه ای تعریف شده است. در این دستورات نیاز است که از کانال داده نیز پاسخ دریافت شود. پس ابتدا از کانال دستور خوانده می شود. اگر پیام ارسال شده برابر با success بود داده از کانال داده دریافت می شود و سپس از کانال دستور خوانده می شود تا پیام مناسب گرفته شده و چاپ شود. در غیر این صورت همان پیام اولیه که از کانال دستور ارسال شده بود چاپ می شود که به این معنا است که اروری رخ داده است.

در اجرای دستور retr تفاوت دیگری وجود دارد و آن این است که در صورت ارسال success از طرف سرور ابتدا یک فایل با نامی که درخواست داده شده بود ساخته می شود و سپس داده ای که از طرف سرور در کانال داده ارسال شده است در این فایل نوشته می شود.

Connect_to_server

ابتدا یک سوکت می‌سازیم حوزه ارتباطی و نوع ارتباط را مانند سوکت سرور قرار می دهیم سپس در صورت موفقیت در ایجاد سوکت شماره پورت را برابر پورت دستور قرار می دهیم که سوکت سرور دستور در آن پورت گوش می‌دهد. با دستور inet_pton آدرس های پروتکل اینترنت را از متن به فرم باینری تبدیل می‌کنیم و در نهایت با فراخوانی سیستمی connect سوکت ایجاد شده را به سوکت سرور که آدرس و شماره پورت آن مشخص شده وصل می‌کنیم