

# تمرین برنامه نویسی اول

## SOCKET PROGRAMMING

سید محمد امین اطیابی (۸۱۰۱۹۸۵۵۹)

مرتضی نوری (۸۱۰۱۹۸۴۸۱)

## نحوه اجرا

با وارد کردن دستور زیر در پوشه اصلی پروژه ، فایل اجرایی کلاینت و سرور در محل فعلی ایجاد می شود .

```
1. make
```

ابتدا نیاز داریم سرور را اجرا کرده و سپس به سراغ کلاینت برویم . هر دو برنامه نیاز به آدرس فایل کانفیگ دارند که باید به صورت Programmer Argument به برنامه ها داده شوند . برای مثال اجرا برنامه ها به صورت زیر است .

```
1. ./server.out Source/config.json
2. ./client.out Source/config.json
```

در صورتی که آدرس فایل کانفیگ وارد نشده یا اشتباه وارد شده باشد ، برنامه اروری به کاربر نمایش می دهد و خاتمه پیدا می کند .

## ساختار

کدهای برنامه ها در پوشه Source قرار گرفته است . این پوشه دارای سه پوشه دیگر است :

- Client : محتوی این پوشه شامل فایل های مورد نیاز برای ایجاد برنامه client می باشد
- Common : برخی فایل های مشترک میان برنامه server و client در این پوشه جای گرفته است
- Server : فایل های مورد نیاز برنامه server در این پوشه قرار دارد

## شرح کدهای سرور و عملکرد آن ها

### کلاس Command

این کلاس وظیفه بررسی دستورات وارد شده توسط کلاینت و پاسخ به آن ها را دارد .

- `bool verify(string msg, bool count)` : در این تابع به طور کلی صحت ورودی دستور ورودی بررسی می شود . این بررسی در دو حالت انجام می گیرد . حالت اول بررسی موجود بودن دستور بدون توجه به دیگر پارامترهای ورودی است . و در حالت دوم دستور با تعداد آرگومان های آن بررسی می شود که آیا تعداد آرگومان ها با نوع دستور همخوانی دارند یا خیر .
- `bool verify(string msg, string cmd, int count)` : این تابع بررسی صحت دستور ورودی با دستوری که ما انتظار داریم را ارزیابی می کند . همچنین تعداد آرگومان دستور در این تابع بررسی خواهد شد .

- `bool verify(string msg, string cmd, string branch, int count)` : این تابع نیز همانند تابع قبلی صحت ورودی را بررسی می‌کند اما برای دستوراتی مانند دستور `delete` که دارای دو شاخه هستند کاربر بیشتری دارد .
- `void enterCredential(string msg, User *user)` : در صورتی که یکی از دستورات `user` یا `pass` وارد شود ، این تابع فرخوانی خواهد شد و با ارسال دستور و کاربر مربوطه به این تابع ، مقادیر نام کاربری و کلمه عبور به آن کاربر اضافه خواهد شد .
- `void response(int fd, int code)` : این تابع وظیفه ارسال پاسخ دستوراتی که پاسخ آن‌ها ثابت است را دارد . با دریافت `File Descriptor` کاربر و کد دستور ، پاسخ دستور به کاربر ارسال می‌شود.
- `void response(int fd, int code, string branch)` : این تابع پاسخ دو دستور `ls` و `retr` را به دلیل مشابهت در کد پاسخ آن‌ها به کاربر ارسال می‌کند .
- `void response(int fd, int code, string branch, string name)` : در این تابع پاسخ به دستورات `pwd` ، `cwd` ، `rename` و `delete` ارسال می‌شود . این دستورات در پاسخ خود نام فایل یا پوشه‌ای دارند که باید در پاسخ باشد . برای همین این تابع یک ورودی نام دارد .
- `string getPath(string msg, int i)` : به طوری کلی این تابع هر آرگومان از دستور را جدا می‌کند و می‌توان با دادن شماره آن را از این تابع دریافت کرد .

## کلاس `CommandExecutor`

در این کلاس دستوراتی که سیستمی هستند و با فایل‌ها سر و کار دارد انجام می‌شود .

- `string pwd()` : به کمک توابع کتابخانه ای لینوکس ، مسیر فعلی را به صورت یک رشته برمی‌گرداند
- `string cwd(string currentPath, string destination)` : با گرفتن مسیر فعلی به مسیر مقصد می‌رود و مسیر جدید را به صورت یک رشته برمی‌گرداند . در صورتی که مسیر مقصد .. باشد به پوشه بالایی می‌رود و این کار را تا زمانی انجام می‌دهد که به مسیر / برسد . در صورتی که مسیر وارد شده .. نباشد بررسی می‌کند آیا مسیر وارد شده صحیح است یا خیر ، در صورت صحیح بودن مسیر جدید را برمی‌گرداند و در غیر این صورت رشته ثابت ارور را می‌دهد .
- `bool mkd(string currentPath, string path)` : این تابع در مسیر فعلی پوشه‌ای ایجاد می‌کند . در صورتی که عملیات موفقیت آمیز باشد `true` و در غیر این صورت `false` می‌دهد .
- `bool dele(string currentPath, string branch, string name)` : این تابع فایل‌ها را به کمک توابع سیستمی لینوکس پاک می‌کند که در صورت موفقیت `true` و در غیر این صورت `false` برمی‌گرداند . برای پاک کردن یک پوشه ابتدا به کمک تابع `cwd` از صحت وجود پوشه اطمینان

حاصل می‌کنم ، سپس به کمک تابع system و اجرا دستور rm با شاخه rf- در سیستم لینوکس ، پوشه مد نظر را پاک می‌کنیم .

- string ls(string currentPath) : لیست فایل‌ها و پوشه‌های مسیر فعلی را در قالب یک رشته برمی‌گرداند .
- bool rename(string currentPath, string currentName, string afterName) : در مسیر فعلی ، نام فایلی را تغییر می‌دهد و در صورت موفقیت true و در غیر این صورت false برمی‌گرداند .
- string help() : راهنما برنامه را در قالب یک رشته برمی‌گرداند .
- string getFileName(string path) : در صورتی که مسیری توسط کاربر وارد شده باشد که انتهای آن نام فایلی باشد ، این تابع نام فای را جدا کرده و برمی‌گرداند .
- long getFileSize(string path) : این تابع حجم یک فایل را بر مبنی بایت برمی‌گرداند . در صورتی که فایل وجود نداشته باشد ، -۱ برگردانده می‌شود .
- string getFileContent(string path) : در صورتی که نیاز به ارسال فایل به کلاینت باشد ، این تابع محتوی فایل را به صورت یک رشته در آورده و برمی‌گرداند .

## کلاس Data

وظیفه ارسال داده در کانال داده به کاربران بر عهده این کلاس است .

- void response(int fd, string data) : این تابع File Descriptor کانال داده کاربر و دیتا مربوطه را گرفته و به کاربر ارسال می‌کند .

## کلاس Logger

این کلاس وظیفه نمایش و ذخیره کردن رخدادهای برنامه را دارد . یک فایل با نام log.txt ساخته شده و در آن ذخیره می‌شود .

- string getDateTime() : این تابع تاریخ و ساعت فعلی را در قالب یک رشته برمی‌گرداند .
- void log(string username, string msg) : این تابع یک رشته رخداد تولید کرده و آن را در فایل رخدادها ذخیره کرده و آن را در کنسول سرور نمایش می‌دهد . رخداد تولید شده دارای تاریخ و ساعت ، نام کاربری و عملیات انجام شده است .
- void log(string username, string msg, string argument) : این تابع یک رشته رخداد تولید کرده و آن را در فایل رخدادها ذخیره کرده و آن را در کنسول سرور نمایش می‌دهد . رخداد تولید شده دارای تاریخ و ساعت ، نام کاربری ، عملیات انجام شده و نام پوشه یا فایل است .

## کلاس Server

این کلاس ، کلاس کلی و مدیریت کننده سرور ما می باشد . پیاده سازی این کلاس به صورت Singleton است ، چرا که ما نباید بیش از یک نمونه از کلاس سرور در برنامه داشته باشیم .

در این کلاس کاربرها و فایل های ادمین وجود دارد .

- `int acceptClientCommand()` : این تابع یک File Descriptor برای ارتباط در کانال دستور به کاربر اختصاص می دهد .
- `int acceptClientData()` : این تابع یک File Descriptor برای ارتباط در کانال داده به کاربر اختصاص می دهد .
- `User *findUser(int fd, fileDescriptor type, vector<User *> _users)` : این تابع به دنبال کاربری با File Descriptor مشخصی در لیست ورودی می گردد .
- `User *findUser(string username, vector<User *> _users)` : این تابع به دنبال کاربری با نام کاربری مشخص در لیست ورودی می گردد .
- `User *findUser(string username, string password, vector<User *> _users)` : این تابع برای احراز هویت کاربر انجام می شود . در این تابع بررسی می کنیم آیا کاربری با نام کاربری و رمز عبور داده شده وجود دارد یا خیر .
- `void removeUser(int fd, fileDescriptor type)` : زمانی که کاربر از سامانه خارج شود ، کاربر به کمک این تابع از لیست حذف می شود .
- `bool canAccess(User *user, string name)` : در این تابع بررسی می کنیم آیا کاربر اجازه دسترسی به فایل مورد نظر را دارد یا خیر . ادمین بودن یا نبودن کاربر در این بررسی اهمیت دارد .
- `void init(string path)` : این کلاس با دریافت مسیر فایل کانفیگ ، داده های مورد نیاز را به کمک کتابخانه متن باز json خوانده و در کلاس سرور ذخیره می کند .
- `void startServer()` : در این کلاس دو کانال ارتباطی ، یکی برای دستور و دیگری برای داده ، ایجاد می شود و آن ها برای استفاده های بعدی ذخیره می شود .
- `void listenCommand()` : این کلاس نقطه ارتباط کانال دستور با کلاینت ها است . در این تابع منتظر دریافت دستور از کاربران می مانیم ، در صورتی که کاربر هنوز وارد نشده باشد ، در لیست newUsers قرار گرفته و File Descriptor به آن اختصاص داده می شود و پس از ورودی موفقیت آمیز به لیستی اصلی کاربرها منتقل می شود . این تابع به کمک فراخوانی سیستمی select ارتباط با چندین کلاینت را مدیریت می کند . هر کاربر در یک مرحله است که جلوتر به توضیح مراحل خواهیم پرداخت . بر اساس مرحله کاربر ، پاسخ ها و عملکرد سرور متفاوت خواهد بود . در صورتی که

کاربر هنوز وارد نشده باشد ، پاسخ به بسیاری از دستورات او با ارور مواجه می شود . به ازای هر دستور یک پاسخ ارسال و یک رخداد در سمت سرور ایجاد می شود . نحوه عملکرد ارسال پاسخ و ذخیره رخداد پیشتر توضیح داده شده است .

ارسال داده با توجه قراردادی بودن این ارتباط به این شرح است . خط اول داده نوع آن را مشخص می کند . این نوع می تواند نمایش در کنسول ، ذخیره به صورت فایل یا ارور باشد که این نوع در سمت کلاینت مورد استفاده قرار خواهد گرفت . در صورتی که نوع فایل باشد ، محل ذخیره و نام فایل در خط دوم خواهد آمد . در ادامه داده ارسالی ، فرستاده می شود .

- void listenData () : این تابع منتظر برقراری ارتباط برای کانال داده است . سوکت کانال داده زمانی شکل می گیرد که کاربر با موفقیت وارد شده باشد . قبل از این ، ارتباط کانال داده برقرار نیست . عملکرد این تابع نیز مشابه تابع قبل است .

## کلاس User

این کلاس به عنوان یک کلاس جهت نگهداری داده های هر کاربر استفاده می شود .

این کلاس حاوی نام کاربری ، کلمه عبور ، ادمین بودن یا نبودن ، حجم کاربر ، مسیر فعلی کاربر ، مرحله کاربر و File Descriptor دستورات و داده ها است .

مرحله یک کاربر عبارت است از :

- ENTER\_USER : این مرحله ، مرحله ابتدایی هر کاربر است که منتظر ورودی نام کاربری هستیم
- ENTER\_PASSWORD : این مرحله پس از ورود نام کاربری است و منتظر وارد کردن کلمه عبور هستیم . در صورتی که نام کاربری و رمز عبور درست باشد به مرحله LOGGED\_IN می رویم ، در غیر این صورت به مرحله ENTER\_USER می رویم .
- LOGGED\_IN : در این مرحله کاربر وارد سامانه شده است و می تواند بقیه دستورات را وارد کند . کاربر با وارد کردن دستور quit به مرحله ENTER\_USER می رود .

## فایل main

در این فایل تابع init سرور صدا می شود تا سرور پیکربندی شود سپس در صورت پیکربندی صحیح ، سرور شروع به کار می کند . در این تابع دو Thread جدید به برنامه اضافه می شود . یکی برای دریافت دستورات کانال دستور و دیگری برای دریافت درخواست اتصال کانال داده . دلیل استفاده از دو Thread دریافت همزمان دستور و درخواست اتصال کانال داده است .

## شرح کدهای کلاینت و عملکرد آنها

### کلاس Client

پیاده سازی این کلاس به صورت Singleton است ، چراکه نباید بیش از یک نمونه از کلاس کلاینت در برنامه موجود باشد . در این کلاس نام کاربری و کلمه عبور کاربر به همراه File Descriptor کانال داده و دستور ذخیره می شود .

- `int connectServer(int port)` : این تابع با دریافت پورت ، به سرور مربوطه وصل می شود و File Descriptor را برمی گرداند .
- `bool responseCode(string msg, int code)` : این تابع کد پاسخ کانال دستور را با کد مورد نظر مقایسه و نتیجه برابر بودن یا نبودن را برمی گرداند .
- `void receiveDataResponse()` : این تابع پاسخ کانال داده را دریافت و تفسیر می کند . در صورتی که نوع داده نمایش در کنسول باشد ، در کنسول نمایش می دهد و در صورتی که فای باشد، فایلی در مسیر فعلی با نام مشخص ساخته و محتوی دریافتی را در آن ذخیره می کند . سپس به دریافت پاسخ از کانال دستور می پردازد .
- `void init(string path)` : این تابع با ورودی گرفتن آدرس فایل کانفیگ پورت اتصال کانال داده و دستور را به کمک کتابخانه متن باز json خوانده و ذخیره می کند .
- `void startCommand()` : این تابع ، تابع `connectServer` را با پورت کانال دستور صدا می زند و File Descriptor کانال دستور را ذخیره می کند .
- `void startData()` : این تابع زمانی صدا زده می شود که نام کاربری و کلمه عبور کاربر تاییده شده باشد . این تابع نام کاربری و کلمه عبور کاربر را به کانال دستور ارسال می کند تا ارتباط برقرار شود .
- `void sendCommand()` : این تابع ، ورودی کاربر را از کنسول خوانده و به سرور ارسال می کند، این تابع همچنان نام کاربری و کلمه عبور کاربر را ذخیره می کند . در انتها برای دستوراتی که نیاز به پاسخ در کانال داده باشد ، ادامه کار را به تابع `receiveDataResponse` محول می کند .
- `void receiveCommandResponse()` : این تابع پاسخ دریافتی در کانال دستور را از سرور دریافت و در کنسول چاپ می کند . همچنین با بررسی پاسخ دستور ، در صورت صحت نام کاربری و کلمه عبور ارتباط برای برقراری ارتباط کانال داده اقدام می کند و در صورت دریافت پاسخ دستور `quit` برنامه را خاتمه می دهد .

### فایل main

در تابع اصلی این فایل پس از صحت پیکربندی ، حلقه از ارسال و دریافت دستورات شکل می گیرد .