



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه‌های کامپیوتری

پروژه‌ی برنامه‌نویسی ۱

نرگس غلامی - ریحانه احمدپور	نام و نامخانوادگی
810198494 - 810198447	شماره دانشجویی
1400/12/27 - جمعه ۲۷ اسفند	تاریخ ارسال گزارش

فهرست گزارش سوالات:

بخش ۰- توضیحات اولیه راجع به منطق برنامه ۲	
بخش ۱- مقدمه ۱۱	
بخش ۲- احراز هویت و دسترسی ۱۲	
بخش ۳- دستورات ۱۵	
بخش ۴- مدیریت خطاهای ۲۶	
بخش ۵- مدیریت حجم کاربران ۲۶	
بخش ۶- کلاینت ۲۷	
بخش ۷- لایکن ۲۷	

بخش 0: توضیحات اولیه راجع به منطق برنامه

این برنامه به زبان C++ و به صورت object oriented پیاده سازی شده است. برای هر موجودیت در برنامه کلاس های جداگانه در نظر گرفته شده است که به شرح زیر است:

کلاس Configuration

کلاس Server

کلاس Client

کلاس User

کلاس Command

که هر کدام به تفصیل توضیح داده می شود.

اول از توضیح **کلاس Configuration** شروع می کیم:

```
class Configuration
{
public:
    Configuration(std::string path);
    int getCommandPort();
    int getDataPort();
    std::vector<User*> getUser();
    std::vector<std::string> getFile();
    std::string getOrgAddr();

private:
    std::vector<User*> users;
    std::vector<std::string> files;
    std::string orgAddr;
    int commandPort;
    int dataPort;
};
```

این کلاس کلاسی است که برنامه‌ی ما با آن شروع می شود و از فایل config.json

اطلاعات دریافت می کند و این اطلاعات را در فیلد های

پرایوت users , files, commandPort, dataPort سمت می کند. در فیلد

پرایوت orgAddr نیز آدرس اولیه برنامه را سیو می شود.

فانکشن های پابلیک این برنامه نیز getter های فیلد های پرایوت به علاوه

. constructor است.

کلاس بعدی **کلاس سرور** است. این کلاس وظیفه‌ی سنت کردن سوکت‌ها را دارد. کانال داده و کانال دستور در این بخش ساخته می‌شوند. این کلاس

به کمک وکتوری از کامندها انجام دستورات کلاینت‌های ورودی را آغاز می‌کند. تعریف کلاس سرور را مشاهده می‌کنید:

```
35 class Server
36 {
37 public:
38     Server(Configuration configuration);
39     bool initiate();
40     int setupServer(int port);
41     void distinguishCommand(int socketFD, char* cmd);
42     bool isLogin();
43     void setDirectory(std::string newDir);
44
45 private:
46     void createLogFile();
47     void addToLog(std::string action);
48
49     Configuration config;
50     std::vector<Command> commands;
51     std::map<int, int> allClients;
52     int lastClientIndex = 0;
53     std::fstream logfile;
54     MP allClientsFD;
55     int commandPort;
56     int dataPort;
57 };
```

فیلد‌های پایه‌ی این کلاس شامل پورت داده(dataPort) و پورت کانال دستور(commandPort) است که با استفاده از این پورت‌ها کانال‌های داده و دستور ساخته می‌شوند.

فیلد `logFile` فایلی است که در آن اطلاعات فعالیت‌های سیستم ذخیره می‌شود (این بخش در `logging` مفصل‌تر توضیح داده خواهد شد).

فیلد وکتوری از کامندها(`vector<*Command>`) در این جهت پیاده سازی شده که هر چند بزرگی که از کلاینت‌های متفاوت وارد می‌شوند بتوانند دستورات متعلق به خودشان را به صورت مستقل پیاده سازی کنند. در هر کلاینت تنها یک کاربر به صورت زنده می‌تواند از خدمات استفاده کند و کاربر دیگر این کلاینت یا باید منتظر `quit` کردن کاربر آنلاین باشد یا از کلاینت دیگر وارد شود. همین‌طور یک کاربر می‌تواند از چند کلاینت وارد شود و از خدمات استفاده کند.

فیلد `allClients` اینگونه است که سوکت هر کلاینت جدیدی که شروع به کار می‌کند را به صورت کلیدی در مپ ذخیره می‌کند که مقدار آن فیلد `lastClientIndex` است، و به این دلیل که بتواند نتیجه‌ی دستورات را بر روی پورت‌های نام بردۀ شده نمایش دهد، زیرا در این صورت است که متوجه می‌شود چه کاربر بر روی چه کلاینتی خروجی دستورش را می‌خواهد.

همان‌طور که گفته شد فیلد `lastClientIndex` ایندکس آخرین کلاینت کامند را نگه می‌دارد در واقع آن `file decryptor` که ساخته می‌شود و به عنوان کلید مپ در `allClients` استفاده می‌شود با توجه به این فیلد می‌تواند کلاینت کامندش را بیابد.

تابع خصوصی `createLogFile` در ابتدای ساخت سرور شروع به کار می‌کند و فایل `log.txt` گفته شده را ایجاد می‌کند یا باز می‌کند، نکته اینجاست ما به دلیل اینکه سرور از کاربرانش اطلاع داشته باشد به گونه‌ای برنامه ریزی کردیم که اگر فایل از قبل ایجاد شده بود و داده داشت، در ادامه‌ی آن فعالیت‌های جدید را ذخیره کند.

```

void Server::createLogFile()
{
    logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::app);

    if (!logfile)
    {
        cout << NO_LOG_FILE;
        logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::trunc);
        logfile.close();
    }
}

```

تابع خصوصی addToFile فعالیت هم اکنون که از کامند می آید را در فایل لگ ذخیره می کند.

```

void Server::addToFile(string action)
{
    system_clock::time_point currentTime = system_clock::now();
    time_t cT = system_clock::to_time_t(currentTime);
    logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::app);

    logfile << ctime(&cT);
    logfile << action;
    logfile.close();
}

```

به توضیح فیلد های پابلیک این کلاس می پردازیم:

اولین مورد که constructor کلاس سرور است. ورودی آن کلاس configuration است که با استفاده از آن فیلد های commandPort و dataPort ساخته می شوند و فایل لگ که داده های مهم سرور را ذخیره می کند شناسایی یا ساخته می شود.

```

Server::Server(Configuration configuration) : config(configuration)
{
    commandPort = configuration.getCommandPort();
    dataPort = configuration.getDataPort();
    createLogFile();
}

```

تابع مهم در این کلاس initiate است. در حقیقت منطق اصلی برنامه در تابع setupServer شروع می شود و در حین آن از تابع initiate استفاده می شود.

در این تابع به علت طولانی بودن کد منطق اصلی برنامه قرار داده می شود و بقیه تنها توضیح داده می شوند. در ابتدای فانکشن با استفاده از تابع setUPServer که کد آن را در زیر مشاهده می کنید کانال های اصلی داده و دستور ساخته می شوند.

```

int Server::setupServer(int port)
{
    struct sockaddr_in address;
    int serverFd;
    serverFd = socket(AF_INET, SOCK_STREAM, 0);

    int opt = 1;
    setsockopt(serverFd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(port);

    bind(serverFd, (struct sockaddr *)&address, sizeof(address));

    listen(serverFd, 4);

    return serverFd;
}

```

سپس در یک حلقه بینهایت می‌رویم. در این حلقه ممکن است دو اتفاق بیفتد:

یک. وارد شدن کلاینت جدید

در صورت وارد شدن کلاینت جدید دو کانال مخصوص این کلاینت از طریق تابع `acceptClient` می‌سازیم که یکی به کانال دستور و دیگری به کانال دیتا وصل می‌شود. سپس این دو کانال مربوطه را به کلاینت در قالب یک `pair` نسبت می‌دهیم. (به این مورد بعداً که می‌خواهیم اطلاعات را بفرستیم نیاز پیدا خواهیم کرد). سپس به این کلاینت جدیدی که وارد شده است یک کلاس `Command` نسبت می‌دهیم که کامندهای ورودی اش را مدیریت بکند. این منطقی است که اجازه می‌دهد چند کاربر همزمان وارد شوند و همه با هم همزمان بتوانند دستور بددهد و هر کس برنامه‌ی خودش اجرا شود.

```
if (i == commandServerFd)
{ // new client

    newCommandSocket = acceptClient(commandServerFd);
    newDataSocket = acceptClient(dataServerFd);
    allClientsFD[lastClientIndex] = make_pair(newCommandSocket, newDataSocket);
    allClients[newCommandSocket] = lastClientIndex++;
    commands.push_back(new Command(config.getUser(), config.getFile(), config.getOrgAddr()));
    FD_SET(newCommandSocket, &master_set);
    if (newCommandSocket > max_sd)
        max_sd = newCommandSocket;
}
```

تابع `:acceptClient`

دو. پیغام دریافت کردن از سمت کلاینت

```
else
{ // client sending msg
    int bytes_received;
    bytes_received = recv(i, buffer, 1024, 0);

    if (bytes_received == 0)
    { // EOF
        commands[allClients[i]]->setUserQuit();

        close(i);
        FD_CLR(i, &master_set);
        if (i == max_sd)
            while (FD_ISSET(max_sd, &master_set) == 0)
                max_sd -= 1;
        continue;
    }
    else if (bytes_received > 0)
    {
        distinguishCommand(allClients[i], buffer);

        //cmd
        send(allClientsFD[allClients[i]].first,
             (commands[allClients[i]]->getCmdChannel().c_str()),
             strlen(commands[allClients[i]]->getCmdChannel().c_str()), 0);
        //data
        send(allClientsFD[allClients[i]].second,
             (commands[allClients[i]]->getDataChannel().c_str()),
             strlen(commands[allClients[i]]->getDataChannel().c_str()), 0);
    }
}
```

حال اگر تعداد بیتی که از کاربر دریافت شده برابر با صفر است به این معنا است که کاربر خارج شده است در نتیجه باید این کلاینت و اطلاعاتش حذف شود.

اگر تعداد بیت دریافت شده بیشتر از صفر بود به این معنا است که دستور وارد شده است، پس باید دستور با استفاده از تابع `distinguishCommand` تشخیص داده شود که دستور مورد نظر چیست. سپس خروجی دستور و دیتا از کانال‌های مربوطه و با استفاده از سیستم کال `send` به کلاینت‌ها ارسال می‌شود.

تابع مهم بعدی تابع distinguishCommand است که کمک می کند دستورات ورودی کاربر با استفاده از switch case تشخیص داده شود. دستورات به ترتیب ورودی های صورت پرتوه هستن و کامند در ابتدا با توجه به اینکه در حال اجرایی روی کدام کلاینت است خروجی ها را آماده می کند. خط ۱۹۹ به این حاطر است اگر کامند ورودی پشتیبانی نشد در خروجی خیر بدیم کامند اشتباه وارد شده و اگر کلا کامند قابل پشتیبانی نبود خط ۲۰۲ اتفاق بی افتاد.

```
source > Server.cpp > ...
158 void Server::distinguishCommand(int socketFD, char* cmd)
159 {
160     vector<string> wordsOfCmd = splitter(cmd);
161     commands[socketFD] ->setWordOfCommand(wordsOfCmd);
162     CMDtype order = getCMDType(wordsOfCmd);
163
164     switch (order)
165     {
166         case USER:
167             addToLog(commands[socketFD] ->loginUser());
168             break;
169         case PASS:
170             addToLog(commands[socketFD] ->checkPass());
171             break;
172         case PWD:
173             commands[socketFD] ->getCurrentDirectory();
174             break;
175         case MKD:
176             addToLog(commands[socketFD] ->makeNewDirectory());
177             break;
178         case DELE:
179             addToLog(commands[socketFD] ->delDirectory());
180             break;
181         case LS:
182             commands[socketFD] ->getFileList();
183             break;
184         case CWD:
185             commands[socketFD] ->changeDirectory(wordsOfCmd.size());
186             break;
187         case RENAME:
188             commands[socketFD] ->renameFile();
189             break;
190         case RETR:
191             addToLog(commands[socketFD] ->downloadFile());
192             break;
193         case HELP:
194             commands[socketFD] ->help();
195             break;
196         case QUIT:
197             commands[socketFD] ->quit();
198             break;
199         case ARGPAR_ERR:
200             commands[socketFD] ->setInvalActivity(PARA_SYNTAX_ERROR_A);
201             break;
202         case NOT_EXIST:
203             commands[socketFD] ->setInvalActivity(TOTAL_ERRORS_A);
204             break;
    }
```

کلاس بعدی **کلاس client** است.

```
class Client
{
public:
    Client() = default;
    bool initiate(int commandPort, int dataPort);

private:
    bool invalidInitiating(int commandPort, int dataPort);
    char cmd[CMD_MAX_LEN];
    char outputCmd[DATA_OUT_LEN] = {0};
    char outputData[CMD_OUT_LEN] = {0};
    int clientCmdFileDes;
    int clientDataFileDes;
};
```

اول به توضیح فیلدهای پرایویت این کلاس می‌پردازیم. `clientCmdFileDes` و `clientDataFileDes` که پورت کانال‌های داده و دستور هستند. به ترتیب

سه آرایه `char` برای ذخیره `command` ، خروجی کانال دستور و خروجی کانال داده است.

تابع `invalidInitiating` بررسی می‌کند آیا یک کلاینت می‌تواند تشکیل شود یا نه و در صورت عدم وجود مشکل، کلاینت جدید به کانال دینا و به کانال

دستور وصل می‌شود.

```
bool Client::invalidInitiating(int commandPort, int dataPort)
{
    clientCmdFileDes = socket(AF_INET, SOCK_STREAM, 0);
    if (clientCmdFileDes < 0)
        return true;
    struct sockaddr_in serverCmdAddr;
    serverCmdAddr.sin_family = AF_INET;
    serverCmdAddr.sin_port = htons(commandPort);
    serverCmdAddr.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    if (inet_pton(AF_INET, IP_ADDRESS, &serverCmdAddr.sin_addr) <= 0)
        return true;
    if (connect(clientCmdFileDes, (struct sockaddr*)&serverCmdAddr, sizeof(serverCmdAddr)) < 0)
        return true;

    clientDataFileDes = socket(AF_INET, SOCK_STREAM, 0);
    if (clientDataFileDes < 0)
        return true;
    struct sockaddr_in serverDataAddr;
    serverDataAddr.sin_family = AF_INET;
    serverDataAddr.sin_port = htons(dataPort);
    serverDataAddr.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    if (inet_pton(AF_INET, IP_ADDRESS, &serverDataAddr.sin_addr) <= 0)
        return true;
    if (connect(clientDataFileDes, (struct sockaddr*)&serverDataAddr, sizeof(serverDataAddr)) < 0)
        return true;

    return false;
}
```

به توضیح تابع initiate می پردازیم:

```
bool Client::initiate(int commandPort, int dataPort)
{
    // Unsuccessful in initiating.
    if (invalidInitiating(commandPort,dataPort))
        return false;

    while (true)
    {
        // Receive command from command line.
        cout << CMD_SIGN;
        memset(cmd, 0, MAX_COMMAND_LENGTH);
        cin.getline(cmd, CMD_MAX_LEN);

        // Send command to server.
        send(clientCmdFileDes, cmd, CMD_MAX_LEN, 0);

        // Receive command output from server and show in command line.
        memset(outputCmd, 0, sizeof outputCmd);
        recv(clientCmdFileDes, outputCmd, sizeof(outputCmd), 0);
        cout << CMD_OUTPUT_COUT << outputCmd << endl;

        // Receive data output from server and show in command line.
        memset(outputData, 0, sizeof outputData);
        recv(clientDataFileDes, outputData, sizeof(outputData), 0);
        cout << DATA_OUTPUT_COUT << outputData << endl;
    }

    return true;
}
```

در آن کلاینت با استفاده از تابع invalidInitiating چک می کند که آیا می تواند به سرور وصل شود یا خیر. اگر بتواند کلاینت ساخته می شود و بعد از آن وارد حلقه‌ی بی‌پایانی می شود که دستورات خود را وارد می کند و از طریق دو recv خروجی کانال دستور و بعد از آن خروجی کانال داده را دریافت می کند.

کلاس بعدی که به توضیح آن می پردازیم **کلاس user** است.

```
10  class User
11  {
12  public:
13      User(std::string username_, std::string pass_, bool isAdmin_,
14            int downladSize, std::string orgAddress_);
15      std::string getName();
16      std::string getPass();
17      bool getisAdmin();
18      int getDSize();
19      int getStatus();
20      void setUserStatus(int status);
21      void decDownloadSize(int decSize);
22      bool isAbleToDownload(int decSize);
23      std::string getOrgAddr();
24      void setClientID(int clientID_);
25      int getClientID();
26
27 private:
28     std::string orgAddress;
29     std::string username;
30     std::string pass;
31     bool isAdmin;
32     int downladSize;
33     int userStatus;
34     int clientID;
35 };
```

توضیح فیلد های پرایوت:

Username : اسم کاربر

Password : رمز کاربر

orgAddress : آدرس کاربر

: آیا این کاربر ادمین است یا خیر isAdmin

: مقدار سایزی که این کاربر می‌تواند دانلود بکند. downloadSize

: وضعیت این کاربر به چه صورت است. آیا به سیستم وارد نشده است. یا این که منتظر رمز ورود است و یا این که به سیستم وارد شده. userStatus

سه حالت بوزر استوری گفته شده در کد به کمک تصویر زیر تشخیص داده می‌شوند.

```
#define NOT_ENTERED 0  
#define WAIT_FOR_PASS 1  
#define ENTERED 2
```

فیلد clientID به این دلیل است که اگر کاربر از طرف دیگر نام کاربری اش را وارد کرده است از همان کلاسیت رمز ورودی را وارد کند اگر از کلاسیت دیگر رمز وارد شد، کنترل شود.

به جز مورد آخر تمام فیلدہای دیگر با استفاده از configuration از فایل config.json استخراج می‌شود.

توضیح فیلدہای پابلیک:

به جز 4 فانکشن اول بقیه در حقیقت getter فیلدہای پرایوت هستند.

به توضیح سه فانکشن بعد از constructor می‌پردازیم.

setUserStatus: این فانکشن وضعیت بوزر را آپدیت می‌کند. (در حقیقت این فانکشن setter است)

decDownloadSize: این فانکشن از مقدار مجاز دانلود کاربر کم می‌کند.

```
void User::decDownloadSize(int decSize)  
{  
    downloadSize -= decSize;  
}
```

isAbleToDelete: این فانکشن در صورتی که کاربر حجم مناسب برای دانلود فایل داشته باشد مقدار true برمی‌گرداند و در صورتی که حجم مناسب برای دانلود

وجود نداشته باشد مقدار false را برمی‌گرداند.

```
bool User::isAbleToDelete(int decSize)  
{  
    if(downloadSize - decSize > 0)  
        return true;  
    return false;  
}
```

کلاس **command** آخرین کلاس برنامه است که به توضیح آن می‌پردازیم. این کلاس وظیفه مدیریت دستورات ورودی کاربر را بر عهده دارد.

```

11  class Command
12  {
13  public:
14      Command(std::vector<User*> users_, std::vector<std::string> files_, std::string addr_);
15      std::string loginUser(int socket);
16      std::string checkPass(int socket);
17      void getCurrentDirectory();
18      std::string makeNewDirectory();
19      std::string delDirectory();
20      void getFileList();
21      void changeDirectory(int isOrg);
22      void renameFile();
23      std::string downloadFile();
24      void help();
25      void quit();
26
27      void setWordOfCommand(std::vector<std::string> wordsOfCmd_);
28      void setDirectory(std::string newDir);
29      bool isLogin();
30      bool fileIsSecure(std::string fileName);
31      void setUserQuit();
32      void setInvalActivity(std::string activity);
33      std::string getDataChannel();
34      std::string getCmdChannel();
35
36  private:
37      void resetChannels();
38
39      std::vector<std::string> wordsOfCmd;
40      std::vector<std::string> files;
41      std::string currentDirectory;
42      std::vector<User*> users;
43      std::string dataChannel;
44      std::string cmdChannel;
45      int onlineUser;
46  };

```

تابع مربوط به دستورات پروژه در این قسمت توضیح داده شده‌اند، اینکه دستورات چگونه هندل می‌شود و به جواب می‌رسند و سپس جواب مخصوص کانال داده و پاسخ و لاغ فایل آماده می‌شود و فرستاده یا گرفته می‌شوند.

```

bool Command::isLogin()
{
    if(onlineUser == -1)
    {
        cmdChannel = NO_USER_FOR_CMD_A;
        return false;
    }
    else
        return true;
}

```

تابع isLogin برای این است که تشخیص دهیم کاربر وارد شده است یا خیر.

می‌توان با استفاده از متغیر onlineUser آن را تشخیص داد. در حقیقت این مقدار شماره یوزری که آنلاین است را در خود دارد و اگر یوزری لاغین نکرده باشد مقدار منفی یک می‌باشد.

تابع setWordOfCommand: برای ست کدن مقدار پرایویت wordsOfCmd معمولی است. یک تابع setter معمولی است و از گذاشتن کد آن صرف نظر می‌کنم.

تابع setDirectory: برای ست کدن دایرکتوری کنونی برنامه است. یک تابع setter معمولی است.

تابع fileIsSecure: بررسی می‌کند آیا فایل جزو فایل‌هایی است که فقط ادمین باید به آن دسترسی داشته باشد یا این که بقیه هم می‌توانند به آن دسترسی داشته باشند.

```

bool Command::fileIsSecure(string fileName)
{
    for(int i = 0 ; i < files.size() ; i++)
    {
        if(files[i].compare(fileName))
            return true;
    }
    return false;
}

```

به این صورت عمل می کند که روی تمامی فایل ها فور می زند و اگر یکی از فایل ها اسمش هم نام با این فایل بود به عنوان فایل امنیتی اعلام می شود در نتیجه مقدار درست را برمی گرداند. ولی از آن طرف اگر این اسم فایل با هیچ یک از فایل ها یکسان نبود مقدار false بازگردانده می شود.

تابع setUserQuit: در این تابع status یوزر به لگین نبودن تغییر پیدا می کند و onlineUser (که نشان دهنده این است که چه شخصی در لحظه آنلاین است و در این کلاینت حضور دارد) را برابر با -1 می کند.

```

void Command::setUserQuit()
{
    for(int i = 0 ; i < users.size() ; i++)
        users[i]->setUserStatus(NOT_ENTERED);
    onlineUser = -1;
}

```

تابع setInvalActivity برای این است که اگر کامند یا دستور ورودی طبق استاندارد تعیین شده نبود(آگومان و پارامترها) یا اینکه کلا دستوری وارد نکرد و دادهی پرت بود متوجه باشیم و خروجی مربوطه را آماده کنیم.

دو تابع گستر چنل ای که زدم برای این است خروجی های مناسب از کامند به کلاینت انتقال پیدا کند. تابع getDataChannel برای کانال داده است و تابع بعدی getCmdChannel برای کانال پاسخ است.

نقیه توابع نیز عملیات های مربوط به دستورات ورودی را انجام می دهد که توضیح آن ها در بخش های بعد انجام می شود.

تابع resetChannels برای این است که در بخش دستورات استرینگ چنل ها را به حالت اولیه ریست کند و سپس دستورات آن را با پاسخشان مقداردهی کنند و به کلاینت بفرستند.

بخش 1 : مقدمه

```

int main(int argc, char * argv[])
{
    Configuration config = Configuration(PATH);
    Server server(config);
    if(!server.initiate())
        cout << "The server could not be built" << endl;

    return 0;
}

```

```

{
    "commandPort": 8080,
    "dataPort": 8081,
    "users": [
        {
            "user": "Reyhane",
            "password": "123",
            "admin": "true",
            "size": "1000000"
        },
        {
            "user": "Narges",
            "password": "321",
            "admin": "false",
            "size": "1000000"
        }
    ],
    "files": [
        "config.json"
    ]
}

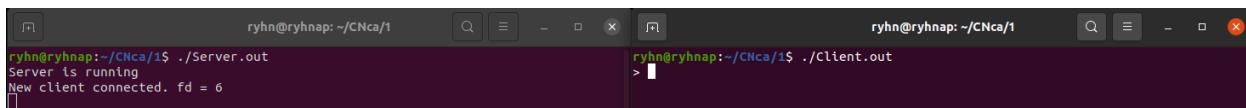
```

در عکس سمت راست خط ۲ و ۳ فایل config.json دو پورت دلخواه را انتخاب کردیم که به ترتیب متعلق به کانال‌های دستور و پاسخ (command) و داده (data) هستند.

در عکس سمت چپ خط ۳ به کمک فایل cpp از کانفیگ استفاده می‌کنیم و آن را به سرور می‌دهیم.

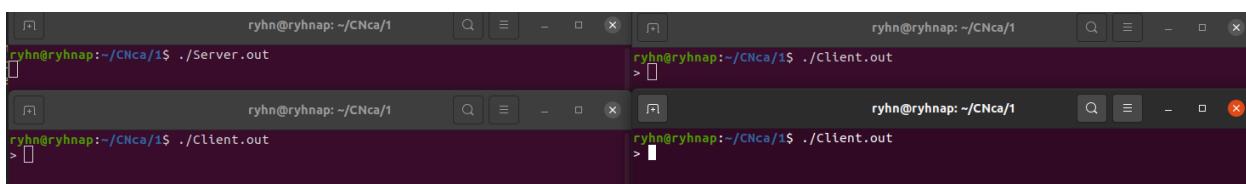
سپس کلاینت به این دو کانال متصل می‌شود.

و در نتیجه به کمک make دو بخش کلاینت و سرور را ایجاد می‌کنیم و هر کدام را در ترمینالی ران می‌کنیم.



```
ryhn@ryhnap:~/CNca/1$ ./Server.out
Server is running
New client connected. fd = 6
ryhn@ryhnap:~/CNca/1$ ./Client.out
> 
```

همین‌طور خروجی با ران کردن چندین کلاینت:



```
ryhn@ryhnap:~/CNca/1$ ./Server.out
ryhn@ryhnap:~/CNca/1$ ./Client.out
> 
ryhn@ryhnap:~/CNca/1$ ./Client.out
> 
ryhn@ryhnap:~/CNca/1$ ./Client.out
> 
```

بخش 2: احراز هویت و دسترسی

```
5   {
6     "user": "Reyhane",
7     "password": "123",
8     "admin": "true",
9     "size": "1000000"
10    },
11   {
12     "user": "Narges",
13     "password": "321",
14     "admin": "false",
15     "size": "1000000"
16   }
```

با توجه به عکس بالا در config.json اطلاعات کاربران سیستم را ایجاد می‌کنیم.

از آنجایی که احراز هویت در سرور آغاز می‌شود، کلاس configuration را در سرور می‌سازیم و از آن استفاده می‌کنیم تا اطلاعات کاربران را داشته باشیم و به صورت وکتوری از کلاس کاربران آن را ذخیره می‌نماییم.

در سرور تقسیم وظیفه صورت می‌گیرد و بقیه کارها به کامند command.cpp سپرده می‌شود که با توجه به وظیفه/فعالیت‌ای که سرور به او داده است کارها را انجام دهد.

حال به بررسی دستورات user و pass می‌پردازیم.

دستور :user

در صورتی که کاربر دستور user را وارد نماید ابتدا چک می‌شود که قبل از این کاربر کاربری وارد نشده باشد زیرا اگر وارد شده باشد دیگر سیستم دوباره وارد شدن را قبول نمی‌کند(زیرا کامندها آیدی ندارند و آیپی و امثال‌هم وجود ندارد که تمایز کامندها را تشخیص دهیم). بعد از این چک می‌کند که آیا یوزرنیم در سیستم وجود دارد یا خیر. که در صورت وجود نداشتن خطای مربوطه چاپ می‌شود. اگر یوزر وجود داشته باشد پیغام ورود فرستاده می‌شود همچنین وضعیت

کاربر به انتظار برای دریافت رمز عبور تغییر پیدا می‌کند. همین طور آیدی کلاینت‌ای که نام کاربر از آن وارد شده را ذخیره می‌کنیم تا مطمئن شویم در یک کلاینت لاجین می‌شوند.

در همین قسمت استینگی به نام `action` داریم که طبق صورت پروژه اطلاعاتی چون فردی که وارد شده است را برای لاج فایل آماده می‌کند.

```
string Command::loginUser(int socket)
{
    resetChannels();
    string action = "";
    if(!isLogin())
    {
        bool check = false;
        for(int i = 0 ; i < users.size() ; i++)
            if(!wordsOfCmd[PARA1].compare(users[i]->getName()) && !users[i]->getStatus())
            {
                cmdChannel = USER_ACCEPTED_A;
                users[i]->setClientID(socket);
                users[i]->setUserStatus(WAIT_FOR_PASS);
                action += (users[i]->getName() + COLON + "Has entered username." + NEWL);
                check = true;
            }
        if(!check)
        {
            cmdChannel = INVALID_USER_PASS_A;
            action += ("User entered wrong username.\n");
        }
    }
    else
    {
        cmdChannel = TOTAL_ERRORS_A;
        action += ("User disturbing another user.\n");
    }
    return action;
}
```

:pass دستور

همراه با دستور `pass` رشته رمز کاربر وارد می‌شود. اول چک می‌کنیم رمز و نام کاربری در یک کلاینت باشند. در صورتی که کاربری وجود داشته باشد که وضعیت آن انتظار برای رمز ورود است، رمز ثبت شده در سیستم برای آن کاربر را با مقدار رمز وارد شده کنار دستور `pass` چک می‌نماییم. اگر یکی بودند وضعیت کاربر را به وارد شده تغییر می‌دهیم و پیغام ورود موفقیت آمیز را ارسال می‌کنیم و مقدار `onlineUser` را نیز به شماره این کاربر تغییر می‌دهیم. در غیر این صورت اگر کلاینت‌ها یکی نبود و رمز اشتباه بود پیغام خطای رمز را می‌دهیم. همچنین ممکن است هیچ کاربری لاجین نکرده باشد که در این صورت پیغام هیچ کاربری لاجین نکرده است ارسال می‌شود.

همین طور به طور مشابه چون در مرحله‌ی لاجین هستیم این اطلاعات را داخل متغیر `action` ذخیره می‌کنیم تا در لاج فایل ذخیره شود.

```

string Command::checkPass(int socket)
{
    resetChannels();
    string action = "";
    if(!isLogin())
    {
        bool check = false;
        for(int i = 0 ; i < users.size() ; i++)
            if(socket == users[i]->getClientID())
                if(users[i]->getStatus() == WAIT_FOR_PASS)
                {
                    check = 1;
                    if(!wordsOfCmd[PARA1].compare(users[i]->getPass()))
                    {
                        cmdChannel = USER_SUCCESSFUL_LOGIN_A;
                        users[i]->setUserStatus(ENTERED);
                        onlineUser = i;
                        action += (users[i]->getName() + COLON + "Has successfully logged in." + NEWL);
                        break;
                    }
                    else
                    {
                        cmdChannel = INVALID_USER_PASS_A;
                        action += (users[i]->getName() + COLON + "Has entered wrong password." + NEWL);
                    }
                }
        if(!check)
        {
            cmdChannel = CLIENT_NOT_LOG_A;
            action += ("User hasn't enter username.\n");
        }
    }
    else
    {
        cmdChannel = TOTAL_ERRORS_A;
        action += ("User disturbing another user.\n");
    }
    return action;
}

```

دیدن خروجی های کد:

The screenshot shows three windows. The top window is a terminal titled 'log.txt - 1 - Visual Studio Code' displaying a log file with entries like 'User entered wrong password.' and 'User has successfully logged in.'. The middle window is a terminal titled 'ryhn@ryhnapi:~/CNca/1\$./Server.out' showing the server's command output. The bottom window is a terminal titled 'ryhn@ryhnapi:~/CNca/1\$./Client.out' showing the client's command input and the server's response.

در ابتدا در ترمینال بالا یوزر را وارد می کیم و نامی خارج از نامهای کانفیگ می دهیم و ارور گفته شده را می گیریم. سپس در همان ترمینال آرگومان یوزر را نمی دهیم و ارور مربوطه را می گیریم. حال نام یوزر درست را وارد می کیم در ترمینال پایین رمز همان یوزر را وارد می کنیم تا ببینیم از کلاینت متفاوت وارد نشود، طبق خروجی این نتیجه هم حاصل شد و ارور اینکه باید به ترتیب نام وارد کن و سپس رمز بزند منتقل شده است. سپس در ترمینال پایین یکی از خدمات را می خواهیم و از آنجایی که در کلاینت پایین کسی لگین نکرده، ارور وارد کردن اکانت خروجی داده می شود. سپس در ترمینال پایین دستوری

وارد می‌کنیم که اصلاً پشتیبانی نمی‌شود(این برای قسمت‌ها بعد است) و طبق انتظار ارور مربوطه را می‌گیریم. سپس در ترمینال بالا رمز کاربر را وارد می‌کنیم و خروجی را طبق انتظار می‌گیریم و در ترمینال پایین دوباره رمز کاربر را می‌زنیم که ارور عدم رعایت ترتیب را به درستی نشان دهیم.

در این بخش به تفاوت کاربر و ادمین اشاره شده و درسته که دستور استفاده شده در بخش‌های بعد معرفی می‌شود، اما برای نشون دادن خروجی این بخش از آن دستور استفاده می‌کنیم. همان‌طور که در فایل config.json دیدیم، فایلی که فقط به آن دسترسی دارد همین فایل است و ادمین Reyhane نام دارد و کاربران دیگر نباید به این فایل دسترسی داشته باشند و منطقاً انتظار نداریم که کاربر Narges به آن دسترسی داشته باشد. کد هم از قبیل توضیح داده شده که تابعی به نام isAdmin که از آن تشخیص می‌دهیم کاربر قادر به انجام دستور است یا خیر.

حال خروجی را می‌بینیم:

```

log.txt - 1 - Visual Studio Code
Selection View Go Run Terminal Help
makefile log.txt M X
log.txt
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:33 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19

```

بخش سه: دستورات

در ابتدای تمام دستورات بعدی لایکین بودن کاربر چک می‌شود.

در بیشتر توابع زیر برای اجرای کامند از فانکشن system کمک گرفته شده است. یک استرینگ از آن دستور مورد نظر ساخته می‌شود و به عنوان ورودی به system داده می‌شود. اگر دستور قابل اجرا بود مقدار صفر را باز می‌گرداند. همچنین در بخشی از کد دستورات عملیات logger انجام شده است که در بخش آخر به توضیح مفصل آن می‌پردازیم.

3.1: دایرکتوری فعلی

در این تابع currentDirectory را که یکی از فیلد های پرایوت کلاس command است و با توجه به اینکه در آینده تغییر می‌کند یا خیر را برای کاربر ارسال می‌نماییم.

```
void Command::getCurrentDirectory()
{
    resetChannels();
    if(isLogin())
        cmdChannel = PWD_A + currentDirectory;
}
```

حال خروجی کد:

The screenshot shows a Visual Studio Code interface with two terminal panes and a code editor. The code editor contains `Command.cpp`. The left terminal pane shows the log file `log.txt` with the following content:

```
Selection View Go Run Terminal Help
makefile log.txt M X C: Command.cpp
log.txt
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane Has entered username.
7 Sun Mar 20 16:26:04 2022
8 Reyhane Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges Has entered wrong username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
Reyhane: Has successfully logged in.
20 Sun Mar 20 17:29:41 2022
21 Narges: Has entered username.
22 Sun Mar 20 17:30:05 2022
23 Narges: Has successfully logged in.
24
25
```

The right terminal pane shows the server's output:

```
ryhn@ryhnapi:~/CNca/1$ ./Server.out
ryhn@ryhnapi:~/CNca/1
> user Reyhane
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> [REDACTED]
```

The bottom terminal pane shows the client's output:

```
ryhn@ryhnapi:~/CNca/1$ ./Client.out
ryhn@ryhnapi:~/CNca/1
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> [REDACTED]
```

3.2: ساختن فایل با دایرکتوری جدید

در این تابع برای ایجاد دایرکتوری جدید از دستور `mkdir` استفاده می‌کنیم و این رشته را به عنوان ورودی به دستور `system` می‌دهیم که آن را ایجاد بکند.

```
string Command::makeNewDirectory()
{
    resetChannels();
    string action = "";
    if(isLogin())
    {
        string newDirectory = "mkdir " + currentDirectory + wordsOfCmd[PARA1];
        int status = system(newDirectory.c_str());
        if (status == SUCCESS)
        {
            cmdChannel = MAKE_A + wordsOfCmd[PARA1] + SPACE + CREATE_A;
            action += (users[onlineUser]->getName() + COLON +
                      "Has successfully create directory named " + wordsOfCmd[PARA1] +
                      " in " + currentDirectory + "." + NEWL);
        }
        else
        {
            cmdChannel += TOTAL_ERRORS_A;
            action += (users[onlineUser]->getName() + COLON +
                      "Hasn't create directory named " + wordsOfCmd[PARA1] +
                      " in " + currentDirectory + "." + NEWL);
        }
    }
    return action;
}
```

حال خروجی کد را بررسی می‌کنیم:

ابتدا لورکیشن پروژه و جایی که می‌خواهیم فایل را ایجاد کنیم نشان می‌دهیم:

The screenshot shows a Visual Studio Code interface with two terminal panes and a code editor. The code editor contains `Command.cpp`. The left terminal pane shows the log file `log.txt` with the following content:

```
Selection View Go Run Terminal Help
makefile log.txt M X C: Command.cpp
log.txt
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
Reyhane: Has successfully logged in.
20 Sun Mar 20 17:29:41 2022
21 Narges: Has entered username.
22 Sun Mar 20 17:30:05 2022
23 Narges: Has successfully logged in.
24
25
```

The right terminal pane shows the server's output:

```
ryhn@ryhnapi:~/CNca/1$ ./Server.out
ryhn@ryhnapi:~/CNca/1
> user Reyhane
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> [REDACTED]
```

The bottom terminal pane shows the client's output:

```
ryhn@ryhnapi:~/CNca/1$ ./Client.out
ryhn@ryhnapi:~/CNca/1
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> [REDACTED]
```

حال خروجی را پس از ساخت دایرکتوری نشان می‌دهیم:

The screenshot shows a Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with files like .vscode, build, check, library, Client.h, Command.h, Config.h, Constant.h, Server.h, User.h, source, Client.cpp, Command.cpp, Config.cpp, Server.cpp, User.cpp, check.txt, Client.out, config.json, log.txt, makefile, README.md, and Server.out. The log.txt file is open in the editor, displaying a timestamped log of user interactions. The terminal window on the right shows a session where a user named 'Reyhane' logs in, creates a directory named 'check', and then successfully deletes it.

```
log.txt - 1 - Visual Studio Code
Edit Selection View Go Run Terminal Help
RENDERER
log.txt M makefile M log.txt M
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home
27

ryhn@ryhnnap:~/CNca/1$ ./Server.out
ryhn@ryhnnap:~/CNca/1$ ./client.out
> user Reyhane
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> mkd check
Command output: 257: check created.
Data output:
> l
> l
```

3.3: پاک کردن فایل یا دایرکتوری

در این تابع برای حذف فایل از دستور rm استفاده می‌کیم و برای حذف دایرکتوری از دستور rm -r استفاده می‌کیم. بنا بر این که دستور کدام باشد رشته مورد نظر را می‌سازیم و به عنوان ورودی به تابع system می‌دهیم.

برای حذف فایل باید حتماً چک شود که آیا کاربر مورد نظر به آن فایل دسترسی دارد. اول چک میکیم آیا فایل جزو فایلهایی است که فقط ادمین به آن دسترسی دارد و اگر جزو آن دسته از فایل‌ها بود چک می‌کیم که آیا کاربر مورد نظر ادمین است یا نه. اگر هر دوی این شروط صحیح باشد کاربر می‌تواند فایل را حذف نماید.

```
string Command::delDirectory()
{
    resetChannels();
    string action = "";
    if(isLogin())
    {
        if(!wordsOfCmd[PARA1].compare(DELETE_DIRT))
        {
            string delCommand = "rm -r " + currentDirectory + wordsOfCmd[PARA2];
            int status = system(delCommand.c_str());
            if (status == SUCCESS)
            {
                cmdChannel = DELETE_A + wordsOfCmd[PARA2] + SPACE + DELETED_A;
                action += (users[onlineUser]->getName() + COLON +
                           "Has successfully deleted directory named " + wordsOfCmd[PARA2] +
                           " in " + currentDirectory + "." + NEWL);
            }
        }
    }
}
```

```

        else if(!wordsOfCmd[PARA1].compare(DELETE_FILET))
    {
        if (!users[onlineUser]->getIsAdmin() && fileIsSecure(wordsOfCmd[PARA1]))
        {
            cmdChannel = FILE_ACCESS;
            action += (users[onlineUser]->getName() + COLON +
                "Permission denied on file named " + wordsOfCmd[PARA2] + "." + NEWL);
            return action;
        }
        string bash_command = "rm " + currentDirectory + wordsOfCmd[PARA2];
        int status = system(bash_command.c_str());
        if (status == SUCCESS)
        {
            cmdChannel = DELETE_A + wordsOfCmd[PARA2] + SPACE + DELETED_A;
            action += (users[onlineUser]->getName() + COLON +
                "Has successfully deleted file named " + wordsOfCmd[PARA2] +
                " in " + currentDirectory + "." + NEWL);
        }
        else
        {
            cmdChannel = TOTAL_ERRORS_A;
            action += (users[onlineUser]->getName() + COLON +
                "Hasn't successfully deleted file named " + wordsOfCmd[PARA2] +
                " in " + currentDirectory + "." + NEWL);
        }
    }
    return action;
}

```

حال خروجی‌های کد را بررسی می‌کنیم:

ابتدا دایرکتوری check که در مثال قبل ایجاد کردیم را حذف می‌کنیم و سپس فایل notneeded.txt که به صورت دستی ایجاد کردیم را پاک می‌کنیم تا خروجی را ببینیم:

درستی نشان دهیم:

The screenshot shows a Visual Studio Code interface with three terminal tabs. The leftmost tab displays the contents of `log.txt`, which contains a timestamped log of user interactions. The middle tab shows the output of the `Server.out` executable, where a user named `Reyhane` logs in and creates a directory named `check`. The rightmost tab shows the output of the `Client.out` executable, where a user named `Narges` logs in and deletes the same directory.

```

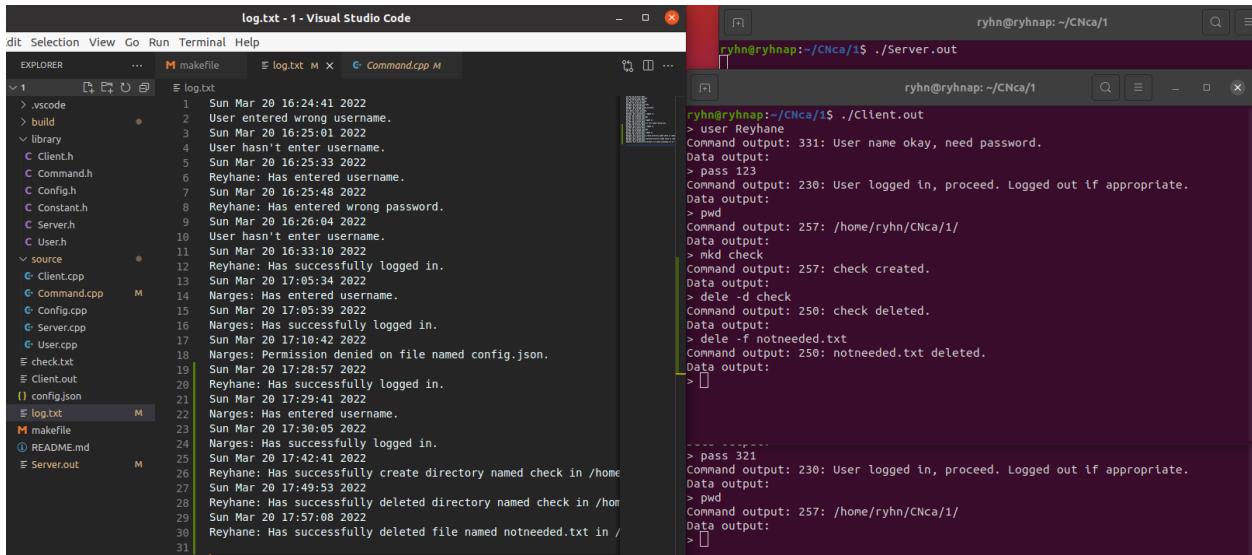
log.txt - 1 - Visual Studio Code
File Selection View Go Run Terminal Help
EXPLORER ... M makefile M log.txt M ...
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:18 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:42 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:08 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home/ryhn
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home/ryhn

ryhn@ryhnnap:~/CNca/1$ ./Server.out
ryhn@ryhnnap:~/CNca/1$ ./Client.out
> user Reyhane
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> mka check
Command output: 257: check created.
Data output:
> dele -d check
Command output: 250: check deleted.
Data output:
> 

ryhn@ryhnnap:~/CNca/1$ ./client.out
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 321
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> 

```

حال پس از حذف موفقیت آمیز دایرکتوری فایل را حذف می کنیم:



The screenshot shows two windows from Visual Studio Code. The left window is titled 'log.txt - 1 - Visual Studio Code' and displays the contents of 'log.txt'. The right window is a terminal window titled 'ryhn@ryhn:~/CNca/1\$./Server.out' showing command-line interactions between a client and a server.

Contents of log.txt:

```
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home
29 Sun Mar 20 17:57:08 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /
31
```

Terminal Output:

```
ryhn@ryhn:~/CNca/1$ ./client.out
> user Reyhane
Command output: 331: User name okay, need password.
Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1/
Data output:
> mkd check
Command output: 257: check created.
Data output:
> dele -d check
Command output: 256: check deleted.
Data output:
> dele -f notneeded.txt
Command output: 250: notneeded.txt deleted.
Data output:
> [ ]
```

> []

```
> [ ]
```

> pass 321
Command output: 230: User logged in, proceed. Logged out if appropriate.
26
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1/
Data output:
> []

3.4: لیست فایل‌های موجود در دایرکتوری

برای گرفتن لیست فایل‌ها از دستور ls استفاده می‌کنیم و به عنوان ورودی به system می‌دهیم.

```
void Command::getFileList()
{
    resetChannels();
    if(isLogin())
    {
        string lsCommand = "ls " + currentDirectory + " > filelist.txt";
        int status = system(lsCommand.c_str());
        if (status == SUCCESS)
        {
            cmdChannel = LIST_TRANSFER_A;
            dataChannel = NEWL;
            dataChannel += toStringConverter("filelist.txt");
            int status_ = system("rm filelist.txt");
            if (status != SUCCESS)
                cmdChannel = TOTAL_ERRORS_A;
        }
        else
            cmdChannel = TOTAL_ERRORS_A;
    }
}
```

حال خروجی کد را چک می‌کنیم:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar lists files: .vscode, build, library, Client.h, Command.h, config.h, Constant.h, Server.h, User.h, source, client.cpp, Command.cpp, config.cpp, Server.cpp, User.cpp, check.txt, Client.out, config.json, log.txt, makefile, README.md, and Server.out. The Command.cpp file is open in the center editor. On the right, there are two terminal windows. The top terminal shows command-line output from a user named 'Reyhane' interacting with a program. The bottom terminal shows the command-line environment with various commands like 'makefile', 'check', 'Client.out', 'config.json', 'filelist.txt', 'library', 'log.txt', 'makefile', 'README.md', 'Server.out', and 'source'.

3.5: عرض کردن دایرکتوری

در اینجا ابتدا چک می‌شود که دستور آرگومان دوم دارد یا نه. اگر نداشت به این معناست که باید به دایرکتوری اولیه بازگردیم و اگر داشت به آدرس گفته شده در آرگومان می‌رویم. هر کدام از این آدرس‌ها که به عنوان مقصدمان انتخاب کردیم را با استفاده از دستور `realpath` در یک فایل می‌بینیم سپس از داخل آن فایل آدرس را می‌خوانیم.

بعد از آن که خواندیم آن فایل موقت را حذف می‌کنیم. با استفاده از تابع `setDirectory` دایرکتوری جدید را که از فایل موقت خواندیم سمت می‌کنیم.

```
void Command::changeDirectory(int isOrg)
{
    resetChannels();
    if(isLogin())
    {
        string cmd;
        if (isOrg == CWD_NO_PARA)
            cmd = "realpath " + users[onlineUser]->getOrgAddr() + "> temp.txt";
        else
            cmd = "realpath " + currentDirectory + wordsOfCmd[PARA1] + "> temp.txt";
        int status = system(cmd.c_str());
        if (status == SUCCESS)
        {
            ifstream myFile("temp.txt");
            string addr;
            getline(myFile, addr);
            myFile.close();
            string delCommand = "rm -r temp.txt";
            int status = system(delCommand.c_str());

            setDirectory(addr + BACK_SLASH);
            // cout << addr << " 111 " << endl;
            cmdChannel = SUCCESSFUL_CHANGE_A;
        }
        else
            cmdChannel = TOTAL_ERRORS_A;
    }
}
```

حال خروجی کد را بررسی می‌کنیم برای حالت‌های گفته شده و در دو کلاینت:

متدی که برای نشان دادن خروجی در نظر گرفتیم اینگونه است که ابتدا لوکیشن کاربر ترمینال بالا را نشان می‌دهیم و سپس یک لوکیشن به عقب می‌روم و سپس لوکیشن هم‌اکنون کاربر را نشان می‌دهیم، از طرفی لوکیشن کاربر دیگر که ترمینال پایین است را نشان می‌دهیم تا مشخص کیم لوکیشن برای هر کاربر به خصوص تغییر می‌کند نه برای تمام کاربران و در نهایت کاربر ترمینال بالای را به لوکیشن اولیه‌اش می‌بریم.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows files like .vscode, build, library, Client.h, Command.h, Config.h, Constant.h, Server.h, User.h, source, client.cpp, command.cpp, config.cpp, Server.cpp, User.cpp, check.txt, Client.out, configuration, log.txt, makefile, README.md, and Server.out. The log.txt file is open in the editor, displaying a timestamped log of events. The right side of the interface has two terminal panes. The top terminal pane shows the command `./Server.out` running on a host named 'ryhn@ryhnnap: ~/CNca/1'. It lists files in the current directory: fileList.txt, library, log.txt, Makefile, README.md, Server.out, and source. It then runs `pwd` and `ls` commands, showing the directory structure. The bottom terminal pane shows the command `./Client.out` running on the same host. It prompts for a user name ('> user Narges') and password ('> pass 321'), then logs the user in ('Command output: 230: User logged in, proceed. Logged out if appropriate.') and lists files in the directory ('> ls').

3.6: عرض کردن نام فایل

در اینجا مانند توضیحاتی که در حذف کردن فایل ابتدا چک می‌شود فایل مجاز است یا نه. اگر مجاز بود با استفاده از دستور mv آن را rename می‌کنیم.

```
void Command::renameFile()
{
    resetChannels();
    if(isLogin())
    {
        if (!users[onlineUser] ->getisAdmin() && fileIsSecure(wordsOfCmd[PARA1]))
        {
            cmdChannel = FILE_ACCESS;
            return;
        }
        string renameCommand = "mv " + currentDirectory + wordsOfCmd[PARA1] + " " +
                               currentDirectory + wordsOfCmd[PARA2];
        int status = system(renameCommand.c_str());
        if (status == SUCCESS)
            cmdChannel = SUCCESSFUL_CHANGE_A;
        else
            cmdChannel = TOTAL_ERRORS_A;
    }
}
```

حال خروجی کد را بررسی می‌کنیم:

همان فایل check.txt که در مثال‌های قبل وجود داشت را با موفقیت به c.txt تغییر می‌دهیم.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a file tree with files like .vscode, build, library, Client.h, Command.h, Config.h, Constant.h, Server.h, User.h, source, Client.cpp, Command.cpp, Config.cpp, Server.cpp, User.cpp, ctxt, Client.out, config.json, log.txt, makefile, README.md, and Server.out. The log.txt file is open in the center editor, displaying a log of events from March 20, 2022. The terminal window on the right shows the command line with the user 'ryhn' running 'makefile' and then executing './Server.out'. The terminal output shows the server's response to the client's actions, including logging in and renaming 'check.txt' to 'c.txt'.

3.7: دانلود فایل

ابتدا مجاز بودن فایل برای کاربر بررسی می‌شود. سپس سایر فایل را بدست می‌آوریم و به عنوان ورودی بهتابع isAbleToDelete می‌دهیم. اگر مقدار true برگردانده شود به این معناست که حجم کافی برای دانلود فایل داریم.

سپس محتویات فایل را می‌خوانیم و برای کاربر از طریق کاتال داده ارسال می‌کنیم.

برای نشان دادن نتیجه‌ی کد از این متده رویم که کاربر نرگس حجم کمی دارد و نمی‌تواند فایل c را دانلود کند اما کاربر ریحانه آنقدری حجم دارد که فایل را دانلود کند و همین‌طور می‌توانیم فایل قفل شده را توسط افراد دانلود کنیم و نتیجه را مشاهده کنیم.

```
string Command::downloadFile()
{
    resetChannels();
    string action = "";
    if(isLogin())
    {
        if (fileIsSecure(wordsOfCmd[PARA1]) && !users[onlineUser]->getisAdmin())
        {
            cmdChannel = FILE_ACCESS;
            action += (users[onlineUser]->getName() + COLON +
                       "Permission denied on file named " + wordsOfCmd[PARA1] + "." + NEWL);
            return action;
        }
        string directory = currentDirectory + wordsOfCmd[PARA1];
        ifstream readFile(directory, ios::binary);
        readFile.seekg(0, ios::end);
        int fileSize = readFile.tellg();
        if (!users[onlineUser]->isAbleToDelete(fileSize))
        {
            cmdChannel = DOWNLOAD_LIMIT_A;
            action += (users[onlineUser]->getName() + COLON + "Download failed cuz connection." + NEWL);
            return action;
        }
        string whole = toStringConverter(directory);

        cmdChannel = SUCCESSFUL_DOWNLOAD_A;
        dataChannel = whole;
        action += (users[onlineUser]->getName() + COLON +
                   "Has successfully downloaded file named " + wordsOfCmd[PARA1] + "." + NEWL);
        users[onlineUser]->decDownloadSize(fileSize);
    }
    return action;
}
```

نتیجه‌ی کد را ببینیم:

ایندا نشان می‌دهیم که کاربر ریحانه موفق بوده فایل سنتگین config.json را دانلود کرده و سپس نشان می‌دهیم موفق بوده که فایل قفل شده config.json را دانلود کرد.

log.txt - 1 - Visual Studio Code

File Selection View Go Run Terminal Help

EXPLORER

- .vscode
- build
- library
- C Client.h
- C Command.h
- C Config.h
- C Constant.h
- C Server.h
- C User.h
- source
 - C Client.cpp
 - C Command.cpp M
 - C Config.cpp
 - C Server.cpp
 - C User.cpp
- c.txt U
- Client.out
- config.json
- log.txt M
- makefile
- README.md
- Server.out M

log.txt

```
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 16:55:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 16:55:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home
29 Sun Mar 20 17:57:08 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /
31 Sun Mar 20 18:25:57 2022
32 Reyhane: Has successfully downloaded file named c.txt.
33
```

ryhn@ryhnnap:~/CNca/1\$./Server.out

Kapshuk, Anders Kaseorg, kehao4, Wolfgang Keller, Eddie Kohler, Austin Liew, Imbar Marinescu, Yandong Mao, Matan Shabtay, Hitoshi Mitake, Carmel Merlitchov, Mark Morrissey, mtsam, Joel Nider, Greg Price, Yuan Shafqat, Eldar Shuhayek, Yongning Shen, Cam Tenney, tyfkda, Rafael Ubal, Warren Toomey, Stephen Tu, Pablo Ventura, Xl Wang, Keilichi Watanabe, Nicolas Wolovick, wwdxao, Grant Wu, Jindong Zhang, Icenowy Zheng, and Zou Chang Wei.

The code in the files that constitute xv6 is
Copyright 2006-2018 Frans Kaashoek, Robert Morris, and Russ Cox.

ERROR REPORTS

We don't process error reports (see note on top of this file).

BUILDING AND RUNNING XV6

To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run "make". On non-x86 or non-ELF machines (like OS X, even on x86), you will need to install a cross-compiler gcc suite capable of producing x86 ELF binaries (see <https://pdos.csail.mit.edu/6.828/>). Then run "make TOOLPREFIX=L386-jos-elf-". Now install the QEMU PC simulator and run "make qemu".

ryhn@ryhnnap:~/CNca/1\$./Client.out

```
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 321
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> 
```

The screenshot shows a Visual Studio Code interface with two terminal panes and an Explorer sidebar.

Explorer Sidebar:

- File tree showing files like `makefile`, `Command.cpp`, `Client.h`, `Config.h`, `Constant.h`, `Server.h`, `User.h`, `Client.cpp`, `Command.cpp`, `Config.cpp`, `Server.cpp`, `User.cpp`, `c.txt`, `Client.out`, `config.json`, `log.txt`, `makefile`, `README.md`, and `Server.out`.

Terminal 1 (Top Right): Shows the command `./Server.out` running on the CNca server.

```
ryhn@ryhnnap:~/CNca/1$ ./Server.out
```

Terminal 2 (Bottom Right): Shows the command `./Client.out` running on the CNca client.

```
ryhn@ryhnnap:~/CNca/1$ ./Client.out
```

Code Editor: Displays the `Command.cpp` file with code related to handling user logins and directory operations.

```
> retr config.json
Command output: 226: Successful Download.
Data output: [
    "commandPort": 8080,
    "dataPort": 8081,
    "users": [
        {
            "user": "Reyhane",
            "password": "123",
            "admin": "true",
            "size": "1000000"
        },
        {
            "user": "Narges",
            "password": "321",
            "admin": "false",
            "size": "1000"
        }
    ],
    "files": [
        "config.json"
    ]
}
```

```
ryhn@ryhnnap:~/CNca/1$ ./Client.out
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 321
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> pwd
Command output: 257: /home/ryhn/CNca/1
Data output:
> [ ]
```

حال کاربر نرگس را نشان می‌دهیم که در این امور موفق نیست:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a file tree with files like `makefile`, `log.txt`, `Client.cpp`, `Command.cpp`, `Config.cpp`, `Server.cpp`, `User.cpp`, `c.txt`, `Client.out`, `config.json`, and `README.md`. The `log.txt` file is open in the editor, displaying a log of interactions between a client and a server. The right side features two terminal panes. The top terminal pane shows the command `./Server.out` running on the server, and the bottom terminal pane shows the command `./Client.out` running on the client, demonstrating a successful connection and data exchange.

```

log.txt - 1 - Visual Studio Code
File Selection View Go Run Terminal Help
EXPLORER ... M makefile E log.txt X
> vscode
> build
library
C Client.h
C Command.h
C Config.h
C constant.h
C Server.h
C User.h
source
E Client.cpp
E Command.cpp M
E Config.cpp
E Server.cpp
E User.cpp
E c.txt
E Client.out
{} config.json
E log.txt
M makefile
D README.md
E Server.out M
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:22 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home
29 Sun Mar 20 17:57:06 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /
31 Sun Mar 20 18:25:57 2022
32 Reyhane: Has successfully downloaded file named c.txt.
33 Sun Mar 20 18:27:47 2022
34 Reyhane: Has successfully downloaded file named config.json.
35 Sun Mar 20 18:48:09 2022
36 Narges: Has successfully logged in.
37 Sun Mar 20 18:48:23 2022
38 Narges: Download failed cuz connection.
39 Sun Mar 20 18:48:44 2022
40 Narges: Permission denied on file named config.json.
41

```

راهنما 3.8

در اینجا تنها توضیحات مربوط به هر دستور ارسال می‌شود.

```

void Command::help()
{
    resetChannels();
    if(isLogin())
    {
        cmdChannel += HELP_TAG_A;
        cmdChannel += USER_HELP_DEC;
        cmdChannel += PASS_HELP_DEC;
        cmdChannel += PWD_HELP_DEC;
        cmdChannel += MKD_HELP_DEC;
        cmdChannel += DELE_HELP_DEC;
        cmdChannel += LS_HELP_DEC;
        cmdChannel += CWD_HELP_DEC;
        cmdChannel += RENAME_HELP_DEC;
        cmdChannel += RETR_HELP_DEC;
        cmdChannel += HELP_HELP_DEC;
        cmdChannel += QUIT_HELP_DEC;
    }
}

```

حال خروجی کد را چک می‌کنیم:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a file tree with files like Client.cpp, Command.cpp, Constant.h, Server.h, Usr.h, Client.cpp, Command.cpp, Constant.h, Server.h, Usr.h, config.json, log.txt, makefile, README.md, and Server.out. The log.txt file is open in the center editor, showing a timestamped log of events from March 20, 2022. On the right, there are two terminal panes. The top terminal shows the command `ryhn@ryhnnap:~/CNca/1\$./Server.out` and its output, which includes a help menu for various commands like PWD, MKD, DELETE, LS, CWD, RENAME, RETR, HELP, and QUIT. The bottom terminal is also visible.

3.9: خارج شدن از سرور

در این قسمت تمام وضعیت‌های کاربران را به وارد نشده تغییر می‌دهیم و همچنین عدد onlineUser را به ۱- تغییر می‌دهیم که به این معناست که دیگر هیچ کاربری در این کلابینت آنلاین نیست.

```
void Command::quit()
{
    resetChannels();
    if(isLogin())
    {
        users[onlineUser] -> setUserStatus(NOT_ENTERED);
        onlineUser = -1;
        cmdChannel = USER_SUCCESSFUL_LOGOUT_A;
    }
}
```

حال خروجی کد را بررسی می‌کنیم:

The screenshot shows a terminal window with the command `ryhn@ryhnnap:~/CNca/1\$./Server.out` and its output. The output includes a help menu for the server and then shows the command `> quit` followed by the message `Command output: 221: Successful Quit.`

بخش چهار: مدیریت خطاهای

در هر بخش در صورتی که امکان وجود خطای خطا داشت چک کردن انجام شد و در صورت نیاز پیغام خطای فرستاده شد.

اما به صورت جمع بندی مواردی از آنها را نشان می‌دهیم:

The screenshot shows a terminal window titled "log.txt - 1 - Visual Studio Code" containing a log file with entries from March 20, 2022. The log entries include various user interactions and system messages. To the right of the log file, there are two terminal windows. The top window is titled "ryhn@ryhnapi:~/CNca/1\$./Server.out" and shows command-line help for the server. The bottom window is titled "ryhn@ryhnapi:~/CNca/1\$./Client.out" and shows command-line help for the client. Both windows display command inputs and their corresponding outputs.

```
log.txt - 1 - Visual Studio Code
in Terminal Help
M logfile  E log.txt M
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /hom
29 Sun Mar 20 17:57:08 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /
31 Sun Mar 20 18:25:57 2022
32 Reyhane: Has successfully downloaded file named c.txt.
33 Sun Mar 20 18:27:47 2022
34 Reyhane: Has successfully downloaded file named config.json.
35 Sun Mar 20 18:48:09 2022
36 Narges: Has successfully logged in.
37 Sun Mar 20 18:48:23 2022
38 Narges: Download failed cuz connection.
39 Sun Mar 20 18:48:44 2022
40 Narges: Permission denied on file named config.json.
41 Sun Mar 20 18:56:11 2022
42 Reyhane: Has entered username.
43 Sun Mar 20 18:56:23 2022
44 Reyhane: Has successfully logged in.
45 Sun Mar 20 18:56:31 2022
46 User disturbing another user.
47

ryhn@ryhnapi:~/CNca/1$ ./Server.out
RENAME [from] [to], Its arguments are used to specify the current and future fil
e's name. It is used to change a file's name.
RETR [name], Its argument is used to specify the file's name. It is used to down
load a file.
HELP, It is used to show instructions guide.
QUIT, It is used for current user to sign out from the server.

Data output:
> quit
Command output: 221: Successful Quit.

Data output:
> user Reyhane
Command output: 331: User name okay, need password.

Data output:
> pass 123
Command output: 230: User logged in, proceed. Logged out if appropriate.

Data output:
> user Narges
Command output: 500: Error.

Data output:
> check inval command
Command output: 500: Error.

Data output:
> []
ryhn@ryhnapi:~/CNca/1$ ./Client.out
> pwd
Command output: 332: Need account for login.

Data output:
> pwd noArgNeededButIGiveIt
Command output: 501: Syntax error in parameters or arguments.

Data output:
> []

```

ابتدا در ترمینال پایین که یوزر نداریم کامند خدمات را وارد می‌کنیم و خروجی انتظار رفته را می‌گیریم.

در ترمینال پایین دوباره کامند دستوری را وارد می‌کنیم اما به گونه‌ای که تعریف نشده و نوع تعریف و آرگومان آن ایجاد دارد و ارور مربوطه را خروجی می‌گیریم.

در ترمینال بالا کاربری را لاجین می‌کنیم و در لاجین کاربر دیگر را لاجین می‌کنیم و چون کاربر خارج نشده ارور داریم.

همین طور کامند کاملاً نامربوط تعریف می‌کنیم و ارور می‌گیریم.

بخش پنجم: مدیریت حجم کاربران

همان‌طور که در بخش دانلود توضیح داده شد با استفاده از تابع isAbleToDelete چک کردن حجم کافی برای دانلود انجام شد و بعد از دانلود

حجم فایل مربوطه از کل حجم کاربر کسر شد.

```

bool User::isAbleToDelete(int decSize)
{
    if(downloadSize - decSize > 0)
        return true;
    return false;
}

```

نتیجه‌ی این بخش دقیقا در مثال‌های قبلی گفته شده، بنابراین برای راحتی کار همان‌ها را می‌آوریم:

The screenshot shows the Visual Studio Code interface with three terminal windows and an Explorer sidebar.

- Explorer Sidebar:** Shows project files including .vscode, build, library, Client.h, Command.h, Config.h, Constant.h, Server.h, Usrch, source, client.cpp, command.cpp, Config.cpp, Server.cpp, User.cpp, c.txt, Client.out, config.json, log.txt, makefile, and README.md.
- Log.txt Content:**

```

1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully created directory named check in /home
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home
29 Sun Mar 20 17:57:08 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /
31 Sun Mar 20 18:25:57 2022
32 Reyhane: Has successfully downloaded file named c.txt.
33 Sun Mar 20 18:27:47 2022
34 Reyhane: Has successfully downloaded file named config.json.
35 Sun Mar 20 18:48:09 2022
36 Narges: Has successfully logged in.
37 Sun Mar 20 18:48:23 2022
38 Narges: Download failed cuz connection.
39 Sun Mar 20 18:48:44 2022
40 Narges: Permission denied on file named config.json.

```
- Terminal 1:** Shows a client connecting and logging in.

```

ryhn@ryhnnap:~/CNca/1$ ./Client.out
> user Narges
Command output: 331: User name okay, need password.
Data output:
> pass 321
Command output: 230: User logged in, proceed. Logged out if appropriate.
Data output:
> retr c.txt
Command output: 425: Can't open data connection.
Data output:
> retr config.json
Command output: 550: File unavailable.
Data output:
> 

```
- Terminal 2:** Shows a user attempting to download files.

```

ryhn@ryhnnap:~/CNca/1$ 

```
- Terminal 3:** Shows a user attempting to delete files.

```

ryhn@ryhnnap:~/CNca/1$ 

```

بخش شش: کلاینت

در بخش صفر قسمت کلاس کلاینت توضیح داده شد.

Logging

ما این بخش را اینگونه طراحی کردیم که اگر فایل موجود نبود، آن را ایجاد کند و اگر موجود بود در ادامه‌ی آن پیویسد زیرا که می‌خواهیم حافظه‌ی سرور ذخیره شود، و اگر در آینده به آن نیازی بود بتوانیم بررسی اش کنیم.

به کمک تایم می‌توانیم زمان هم اکنون را بیاییم. این بخش را در سرور انجام دادیم زیرا فعالیت‌ها به کمک سرور و در سرور انجام می‌شود و هنگامی که این را در سرور قرار دادیم، دیدیم که مانند بک آپ و نگه داشتن اطلاعات حیاطی سرور است به همین علت آن را همان جا نگه داشتیم.

ساخت ابتدایی فایل که پس از ساخت سرور انجام می شود اینگونه است که در سازندهی سرور تابع `creatLogFile` صدا زده می شود و در اینجا ما فایل را در صورت نبود می سازیم و در صورت بودن از آن استفاده می کنیم و در ادامهی آن می نویسیم.

در طول انجام عملیات و فعالیت سرور همان چند دستور خاص گفته شده(که واردشدن، ساختن و پاک کردن و دانلود کردن فایل بود) در صورت پروژه را به کمک تابع `addToFile` در فایل `log.txt` می نویسیم، و روند آن اینگونه است که کلاس کامند `Command.cpp` که دستورات را پردازش می کند آن کامندهای خاص نام بده شده را بررسی می کند و خروجی را آماده می کند و به صورت استرینگ به سرور می دهد و سرور به کمک همین تابع در فایل می نویسد. در ابتدا به کمک توابع کتابخانه زمان را وارد می کند و سپس فعالیت را وارد می کند.

نشان دادن توابع مربوط به لاغر:

```
void Server::createLogFile()
{
    logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::app);

    if (!logfile)
    {
        cout << NO_LOG_FILE;
        logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::trunc);
        logfile.close();
    }
}

Server::Server(Configuration configuration) : config(configuration)
{
    commandPort = configuration.getCommandPort();
    dataPort = configuration.getDataPort();
    createLogFile();
}

void Server::addToFile(string action)
{
    system_clock::time_point currentTime = system_clock::now();
    time_t cT = system_clock::to_time_t(currentTime);
    logfile.open(LOG_FILE_NAME, fstream::in | fstream::out | fstream::app);

    logfile << ctime(&cT);
    logfile << action;
    logfile.close();
}
```

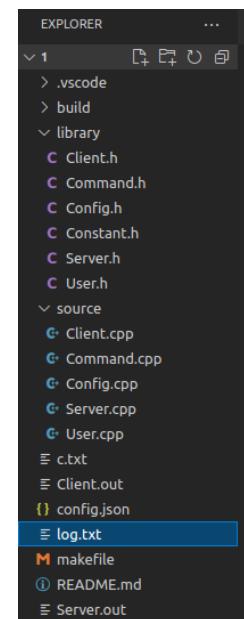
حال نتایج کامندها که در لاغر ذخیره شده را نمایش می دهیم:

```

makefile      log.txt      Server.cpp
log.txt
1 Sun Mar 20 16:24:41 2022
2 User entered wrong username.
3 Sun Mar 20 16:25:01 2022
4 User hasn't enter username.
5 Sun Mar 20 16:25:33 2022
6 Reyhane: Has entered username.
7 Sun Mar 20 16:25:48 2022
8 Reyhane: Has entered wrong password.
9 Sun Mar 20 16:26:04 2022
10 User hasn't enter username.
11 Sun Mar 20 16:33:10 2022
12 Reyhane: Has successfully logged in.
13 Sun Mar 20 17:05:34 2022
14 Narges: Has entered username.
15 Sun Mar 20 17:05:39 2022
16 Narges: Has successfully logged in.
17 Sun Mar 20 17:10:42 2022
18 Narges: Permission denied on file named config.json.
19 Sun Mar 20 17:28:57 2022
20 Reyhane: Has successfully logged in.
21 Sun Mar 20 17:29:41 2022
22 Narges: Has entered username.
23 Sun Mar 20 17:30:05 2022
24 Narges: Has successfully logged in.
25 Sun Mar 20 17:42:41 2022
26 Reyhane: Has successfully create directory named check in /home/ryhn/CNca/1.
27 Sun Mar 20 17:49:53 2022
28 Reyhane: Has successfully deleted directory named check in /home/ryhn/CNca/1..
29 Sun Mar 20 17:57:08 2022
30 Reyhane: Has successfully deleted file named notneeded.txt in /home/ryhn/CNca/1..
31 Sun Mar 20 18:25:57 2022
32 Reyhane: Has successfully downloaded file named c.txt.
33 Sun Mar 20 18:27:47 2022
34 Reyhane: Has successfully downloaded file named config.json.
35 Sun Mar 20 18:48:09 2022
36 Narges: Has successfully logged in.
37 Sun Mar 20 18:48:23 2022
38 Narges: Download failed cuz connection.
39 Sun Mar 20 18:48:44 2022
40 Narges: Permission denied on file named config.json.
41 Sun Mar 20 18:56:11 2022
42 Reyhane: Has entered username.
43 Sun Mar 20 18:56:23 2022
44 Reyhane: Has successfully logged in.
45 Sun Mar 20 18:56:31 2022
46 User disturbing another user.
47

```

حال لوکیشن فایل را در فolder پروژه نشان می دهیم:



حال خروجی استرینگی فانکشن‌های مربوطه را نشان می‌دهیم:

```
36 > string Command::loginUser(int socket) ...
65
66 > string Command::checkPass(int socket) ...
106
107 > void Command::getCurrentDirectory() ...
114
115 > string Command::makeNewDirectory() ...
140
141 > string Command::delDirectory() ...
188
189 > string toStringConverter(string fileN) ...
201
202 > void Command::getFileList() ...
222
223 > void Command::changeDirectory(int isOrg) ...
251
252 > void Command::renameFile() ...
271
272 > bool Command::fileIsSecure(string fileName) ...
281
282 > string Command::downloadFile() ...
323
324 > void Command::help() ...
343
344 > void Command::quit() ...
```

خطهای ۳۶ و ۴۱ و ۵۶ و ۲۸۲ و ۱۴۱ دستوراتی هستند که نیاز داریم نتیجه‌ی آن را در لاگر ذخیره کیم. به همین علت از نوع استرینگی هستند که از طریق کلاس کامند به سرور برسد و از طریق تابع نام برده شده در فایل لاگر بنویسد.

حال قسمتی که از کامند داده به تابع لاجر منتقل می‌شود را نشان می‌دهیم:

```
source > C:\Server.cpp > ...
158     void Server::distinguishCommand(int socketFD, char* cmd)
159     {
160         vector<string> wordsOfCmd = splitter(cmd);
161         commands[socketFD]->setWordOfCommand(wordsOfCmd);
162         CMDtype order = getCMDType(wordsOfCmd);
163
164         switch (order)
165         {
166             case USER:
167                 addToLog(commands[socketFD]->loginUser());
168                 break;
169             case PASS:
170                 addToLog(commands[socketFD]->checkPass());
171                 break;
172             case PWD:
173                 commands[socketFD]->getCurrentDirectory();
174                 break;
175             case MKD:
176                 addToLog(commands[socketFD]->makeNewDirectory());
177                 break;
178             case DELE:
179                 addToLog(commands[socketFD]->delDirectory());
180                 break;
181             case LS:
182                 commands[socketFD]->getFileList();
183                 break;
184             case CWD:
185                 commands[socketFD]->changeDirectory(wordsOfCmd.size());
186                 break;
187             case RENAME:
188                 commands[socketFD]->renameFile();
189                 break;
190             case RETR:
191                 addToLog(commands[socketFD]->downloadFile());
192                 break;
193             case HELP:
194                 commands[socketFD]->help();
195                 break;
196             case QUIT:
197                 commands[socketFD]->quit();
198                 break;
199             case ARGPAR_ERR:
200                 commands[socketFD]->setInvalActivity(PARA_SYNTAX_ERROR_A);
201                 break;
202             case NOT_EXIST:
203                 commands[socketFD]->setInvalActivity(TOTAL_ERRORS_A);
204                 break;
```