به نام خدا

گزارش پروژه اول درس شبکه‌های کامپیوتری

دکتر یزدانی

پرنیان فاضل ۸۱۰۱۹۸۵۱۶ - پریا خوش‌تاب ۸۱۰۱۹۸۳۸۷

ما برای پیاده سازی این پروژه کلاس های زیر را پیاده سازی کردیم:

- کلاس Client
- کلاس Server
- کلاس UserManager
- کلاس CommandHandler
- کلاس User
- کلاس JsonParser
- کلاس Logger

کلاس Client:

این کلاس وظیفه برقراری ارتباط با سرور را بر عهده دارد. در واقع این کلاس پس از وصل شدن به سرور از طریق پورت های داده شده در فایل کانفیگ، دستورها را از طریق کانال دستور به سرور ارسال میکند و سپس پاسخ دستور را از طریق کانال دستور یا داده از سرور دریافت میکند.

```
class Client {
public:
    Client();
    void run();
    void readPorts(std::string configPath);
private:
    int commandChannelPort, dataChannelPort;
};
```

فیلدها:

- commandChannelPort: پورت کانال دستور است که از طریق فایل کانفیگ خوانده می‌شود.
- dataChannelPort: پورت کانال داده است که از طریق فایل کانفیگ خوانده می‌شود.

متدها:

- readPorts: در این تابع از طریق فایل کانفیگ، پورت های کانال داده و دستور را می‌خوانیم و در فیلدهای مربوطه ذخیره می‌کنیم.
- run: در این تابع ابتدا به ۲ کانال دستور و داده متصل می‌شویم. دستور کاربر را از طریق ترمینال دریافت کرده و با استفاده از کانال دستور به سرور ارسال می‌کنیم و پس از دریافت خروجی از کانال دستور و داده، (با تابع recv) جوابی که از سرور دریافت کردیم را نمایش می دهیم.

کلاس CommandHandler:

این کلاس وظیفه هندل کردن دستورهای ورودی کاربر و برگرداندن خروجی مناسب را دارد.

فیلدها:

- Logger: برای ثبت و ذخیره تمامی اطلاعات و دستورهای کاربر با تاریخ و ساعت وقوع استفاده میشود.
- dataFd: برای ارسال دیتا از طریق کانال دیتا (دستور retr) استفاده میشود.
- directory: دایرکتوری فعلی کلاینت در این متغیر ذخیره میشود.
- clientDirectory: دایرکتوری اولیه در این متغیر ذخیره میشود.
- userManager: برای مدیریت کارهای مربوط به ورود و خروج کاربر استفاده میشود.
- jsonParser:

متدها:

- userHandler: این متد هندلر دستور user میباشد. در این تابع ابتدا خطاهای مربوط به مدیریت کاربران بررسی میشوند و سپس عملیات چک کردن نام کاربری وارد شده و true کردن متغیر مربوطه صورت میگیرد و پیغام مناسب ریترن میشود.
- passHandler: این متد هندلر دستور pass میباشد. در این تابع ابتدا خطاهای مربوط به مدیریت کاربران بررسی میشوند و سپس عملیات چک کردن رمز عبور وارد شده و true کردن متغیر مربوطه صورت میگیرد و پیغام مناسب ریترن میشود.

- pwdHandler: این متد هندلر دستور pwd میباشد که تنها دایرکتوری فعلی(directory) را ریترن میکند.
- mkdHandler: این متد هندلر دستور mkd میباشد. در این تابع ابتدا مسیر دایرکتوری از کنار هم قراردادن دایرکتوری فعلی کاربر و مسیر وارد شده در ورودی ساخته می شود و سپس دستور mkdir توسط تابع execCommand اجرا میشود.
- deleHandler: این متد هندلر دستور dele میباشد. در این تابع ابتدا دسترسی غیر مجاز به فایل های مربوط به ادمین چک می شود و سپس مسیر دایرکتوری از کنار هم قراردادن دایرکتوری فعلی کاربر و مسیر وارد شده در ورودی ساخته می شود و دستور rm (پاک کردن فایل) یا rm -r (پاک کردن دایرکتوری) توسط تابع execCommand اجرا میشود.
- lsHandler: این متد هندلر دستور ls میباشد. در این تابع دستور ls توسط تابع execCommand اجرا میشود.
- cwdHandler: این متد هندلر دستور cwd میباشد. در این تابع ابتدا چک می شود که اگر آرگمانی وارد نشده بود، دایرکتوری اولیه ریترن شود و در غیر این صورت دستور chdir به ترکیب دایرکتوری فعلی کاربر و مسیر وارد شده در ورودی و سپس دستور pwd توسط تابع execCommand اجرا میشود.
- renameHandler: این متد هندلر دستور rename میباشد. در این تابع ابتدا دسترسی غیر مجاز به فایل های مربوط به ادمین چک می شود و سپس مسیر فایل ها از کنار هم قراردادن

دایرکتوری فعلی کاربر و مسیر های وارد شده در ورودی ساخته می شوند و دستور mv توسط تابع execCommand اجرا میشود.

- retrHandler: این متد هندلر دستور retr میباشد. در این تابع ابتدا دسترسی غیر مجاز به فایل های مربوط به ادمین چک می شود و مسیر فایل از کنار هم قراردادن دایرکتوری فعلی کاربر و مسیر وارد شده در ورودی ساخته می شود، سپس سایز فایل موردنظر بر حسب بایت با استفاده از تابع tellg محاسبه میشود و در صورت داشتن سایز کافی موجود برای دانلود، سایز این فایل از سایز موجود کاربر کاسته می شود و در نهایت محتویات فایل با استفاده از تابع sendfile از طریق کانال دیتا فرستاده میشود.
- helpHandler: این متد هندلر دستور help میباشد. در این تابع رشته حاوی دستورات موجود در سرور را به همراه راهنمای استفاده از آنها ریترن میکند.
- quitHandler: این متد هندلر دستور quit میباشد. در این تابع کاربر فعلی با ست کردن متغیرهای مربوطه به false از سیستم خارج می شود.
- run: این متد برای مدیریت دستورات وارد شده از سوی یک کاربر و فراخوانی هندلر موردنظر نوشته شده است. در این تابع ابتدا با استفاده از تابع getRequest رشته ورودی را به دستور و آرگمان های آن parse میکنیم و سپس علاوه بر مدیریت کردن خطاها، تابع هندلر مربوط به هر دستور را صدا میزنیم.

- execCommand: این تابع برای اجرای دستورات ترمینال سیستم عامل با استفاده از تابع system استفاده میشود و در نهایت رشته خروجی را برمیگرداند. برای پیاده سازی این تابع از این لینک استفاده کردیم.

```
struct Request{
    std::string command;
    std::vector<std::string> args;
};

class CommandHandler{
public:
    CommandHandler();
    CommandHandler(JsonParser jsonParser, Logger* logger, int data_fd);
    std::string run(std::string input);
private:
    Logger* logger;
    int dataFd;
    std::string directory, clientDirectory;
    UserManager userManager;
    JsonParser jsonParser;
    std::string userHandler(std::string username);
    std::string passHandler(std::string pass);
    std::string execCommand(std::string command, std::vector<std::string> arguments);
    std::string pwdHandler(std::vector<std::string> arguments);
    std::string mkdHandler(std::vector<std::string> arguments);
    std::string deleHandler(std::vector<std::string> arguments);
    std::string lsHandler();
    std::string cwdHandler(std::vector<std::string> arguments);
    std::string renameHandler(std::vector<std::string> arguments);
    void retrHandler(std::vector<std::string> arguments);
    std::string helpHandler();
    std::string quitHandler();
};
```

کلاس JsonParser:

این کلاس وظیفه parse کردن jsonهای فایل کانفیگ ورودی و ذخیره کردن اطلاعات آنها را بر عهده دارد.

متدها:

- configFilePath: مسیر فایل کانفیگ سرور در این متغیر ذخیره میشود.
- users: وکتوری از جنس کلاس User میباشد که اطلاعات مربوط به کاربران در این وکتور ذخیره میشود.
- privilegedFiles: وکتوری از جنس string میباشد که اسم فایل هایی که تنها ادمین به آنها دسترسی دارد، در این متغیر ذخیره میشود.
- commandChannelPort: پورت مربوط به کانال دستور در این متغیر ذخیره میشود.
- dataChannelPort: پورت مربوط به کانال دیتا در این متغیر ذخیره میشود.

فیلدها:

- getUsers: فیلد users را ریترن میکند.
- getPrivilegedFiles: فیلد privilegedFiles را ریترن میکند.
- getCommandChannelPort:فیلد commandChannelPort را ریترن میکند.
- getDataChannelPort فیلد dataChannelPortرا ریترن میکند.
- isPrivileged: این متد مشخص میکند که فایل ورودی در لیست فایل های متمایز ادمین هست یا خیر.

```
class JsonParser
{
public:
    JsonParser();
    JsonParser(std::string config_file_path);
    std::vector<User> getUsers();
    std::vector<std::string> getPrivilegedFiles();
    int getCommandChannelPort();
    int getDataChannelPort();
    bool isPrivileged(const std::string file);
private:
    std::string configFilePath;
    std::vector<User> users;
    std::vector<std::string> privilegedFiles;
    int commandChannelPort, dataChannelPort;
};
```

کلاس Logger:

این کلاس در زمان اجرا یک فایل log در کنار خود ایجاد میکند و تمامی اطلاعات را با تاریخ و ساعت وقوع در آن ذخیره کند.

فیلدها:

- path: مسیر فایل log در این متغیر ذخیره میشود.

متدها:

- record: این متد برای ذخیره کردن وقایع در فایل log استفاده میشود. در این متد با استفاده از یک متغیر به نام currTime از جنس time_t زمان جاری دستور و رشته مورد نظر آن در انتهای فایل append میشود.

```
class Logger {
public:
    Logger();
    Logger(std::string path);
    void record(std::string message);

private:
    std::string path;
};
```

کلاس Server:

فیلدها:

- logger: پوینتری به یک متغیر از جنس Logger میباشد که برای ثبت وقایع به commandHandlerها پاس داده میشود.
- commandHandlers: یک map از جنس int به CommandHandler میباشد که برای ذخیره کردن commandHandler مربوط به fd دستور سوکت کلاینت ها استفاده می‌شود.

توجه شود که این مپ برای این لازم است که بتوانیم درخواست کاربران مختلف را به درستی با سرور هندل کنیم.

- jsonParser: برای parse کردن و ذخیره کردن اطلاعات فایل کانفیگ سرور از این متغیر استفاده میشود.

متدها:

- setupSocket: این متد پورت را به عنوان ورودی دریافت میکند و یک سوکت را روی پورت مربوطه ایجاد میکند و سپس روی fd مربوط به این سوکت bind و listen میکند.
- start: در این متد ابتدا دو سوکت روی پورت های کانال داده و دستور ایجاد میکنیم و سپس با اسفاده از select روی fdها حلقه میزنیم تا چندین کاربر بتوانند به صورت همزمان و non-blocking با سرور ارتباط برقرار کنند. اگر تغییرات روی fd مربوط به کانال دستور باشد، یعنی یک کلاینت در حال وصل شدن به سرور است که در این حالت connection های کانال داده و دستور را accept میکنیم و در غیر این صورت، کلاینت در حال فرستادن پیام به سرور میباشد که در این حالت با استفاده از تابع send پیام را از کانال دستور به سرور ارسال میکنیم.

```cpp
class Server {
public:
    Server(std::string configPath);
    int setupSocket(int port);
    void start();

private:
    Logger* logger;
    std::map<int, CommandHandler> commandHandlers; //commandFd to commandHandler
    JsonParser jsonParser;
};
```

کلاس User:

فیلدها:

- username: نام کاربری کاربر در این متغیر ذخیره میشود.
- password: رمز عبور کاربر در این متغیر ذخیره میشود.
- availableSize: حجم مجاز کاربری کاربر در این متغیر ذخیره میشود.
- admin: یک متغیر از جنس bool است که اگر کاربر ادمین باشد، true میشود.

متدها:

- getUsername: فیلد username را ریترن میکند.
- getPassword: فیلد password را ریترن میکند.
- getAvailableSize: فیلد availableSize را ریترن میکند.
- isAdmin: فیلد admin را ریترن میکند.
- reduceAvailableSize: این متد به اندازه سایز ورودی از حجم باقیمانده کاربر کم میکند.

```cpp
class User{
private:
    std::string username;
    std::string password;
    double availableSize;
    bool admin;

public:
    User();
    User(std::string username, std::string password, double availableSize, std::string admin)
    std::string getUsername();
    std::string getPassword();
    int getAvailableSize();
    bool isAdmin();
    void reduceAvailableSize(double size);
};
```

کلاس UserManager:

فیلدها:

- user: یک متغیر از جنس User است که این کلاس برای مدیریت ورود و خروج این کاربر نوشته شده است.
- loggedIn: یک متغیر از جنس bool است که در صورتی که کاربر مورد نظر(user) دستور user را وارد کند و موفقیت آمیز باشد، این متغیر true میشود.
- foundUser: یک متغیر از جنس bool است که در صورتی که کاربر مورد نظر(user) دستور pass را وارد کند و موفقیت آمیز باشد، این متغیر true میشود.

متدها:

- checkUser: این تابع علاوه بر بررسی کردن خطاهای مربوط به دستور user، روی کاربران پیمایش میکند و در صورتی که کاربر با نام کاربری ورودی را بیابد، متغیر foundUser را true میکند و خروجی true برمیگرداند.
- login: این تابع علاوه بر بررسی کردن خطاهای مربوط به دستور pass، رمز عبور وارد شده را با رمز عبور کاربر مطابقت میدهد و در صورتی که یکسان باشند، متغیر loggedIn را true میکند و خروجی true برمیگرداند.
- quit: این تابع ابتدا بررسی میکند که اگر کاربری وارد نشده، خطا بدهد و در غیر اینصورت متغیرهای foundUser و loggedIn را false میکند و و خروجی true برمیگرداند.
- isLoggedIn: فیلد loggedIn را ریترن میکند.
- getUser: فیلد user را ریترن میکند.

```
class UserManager{
public:
    UserManager();
    std::string checkUser(std::vector<User> users,std::string username);
    bool login(std::string pass);
    bool quit();
    bool isLoggedIn();
    User getUser();
private:
    User user;
    bool loggedIn, foundUser;
};
```