به نام خدا

شبکه های کامپیوتری

تمرین برنامه نویسی شماره ۰۱

(FTP-Server)

اعضای گروه :

محمد سعادتی	**۸۱۰۱۹۸۴۱۰**
محمد عراقی	**۸۱۰۱۹۸۴۳۶**

مقدمه

هدف از این پروژه آشنایی با مفاهیم socket programming و ارتباط از طریق socket ها بوده است . کلاینت ها با دانستن پورت سرور به سرور وصل شده و ارتباط برقرار می کنند و بسته به نوع command ها ، اطلاعات و پیام های متناسب میان سرور و هر کلاینت رد و بدل می شود.

از آنجایی که قرار است بر روی دو پورت اطلاعات را منتقل کنیم ، دو سوکت در نظر میگیریم. یکی را روی پورت command channel ، bind می کنیم و دیگر پورت را روی data channel ، bind می کنیم.

نحوه اجرا

در دایرکتوری های Client و Server دستور make را برای ساخت پروژه توسط g++ اجرا کنید.

در دایرکتوری Server اجرا کنید: ./server.out config.json

در دایرکتوری Client اجرا کنید ./client.out config.json

لاگ های مربوط به اطلاعات اجرا در یک فایل با نام log.txt در پوشهServer نوشته می شوند.

کلاس ها و متد ها

۱) CommandHandler

وظیفه ی این کلاس تشخیص دستور ارسال شده برای سرور و پاسخ مناسب به آن است و بخش بیشتر پروژه به عهده ی آن است .

این کلاس دارای وکتوری ازنام فایل ها (فایل های مجاز کاربران) و همچنین تابع get_command است.

تابع get_command با هربار دریافت داده توسط سرور صدا زده میشود و پیام ارسالی را تحلیل میکند و پاسخ مناسب را آماده ی ارسال میکند . ورودی این تابع ، بافر سرور (پیام ارسال شده توسط کلاینت) و file descriptor کلاینت فرستنده ی پیام است.

```
class CommandHandler
{
public :
    std::vector<std::string> get_command(char buf[MAX_BUFFER_SIZE] , int fd);
    const std::string current_date_time();
    void write_log(std::string filetext);
    double get_file_size(std::string filename);
    std::string exec(const char* cmd);
    bool is_exist(std::string name);

    static inline std::vector<std::string> files;
};
```

دیگر توابع :

cuttent_date_time:

برای به دست آوردن زمان کنونی بر حسب سال و ماه و روز و ...

```
const string CommandHandler::current_date_time()
{
    time_t     now = time(0);
    struct tm  tstruct;
    char       buf[80];
    tstruct = *localtime(&now);
    strftime(buf, sizeof(buf), "%Y-%m-%d.%X", &tstruct);
    return buf;
}
```

:write_log

برای نوشتن لاگ در فایل مربوطه برای نگهداری اطلاعات ورود و خروج و ...

```
void CommandHandler::write_log(string filetext)
{
    fstream myfile;
    myfile.open ("log.txt", ios::app);
    myfile << filetext;
    myfile.close();
}
```

:get_file_size

برای پیدا کردن سایز فایل با اسم داده شده در مسیر کنونی برای استفاده در دانلود های یوزر

```
double CommandHandler::get_file_size(string filename)
{
    struct stat stat_buf;
    int rc = stat(filename.c_str(), &stat_buf);
    return rc == 0 ? stat_buf.st_size : -1;
}
```

:exec

برای اجرای کامند ها در سطح کرنل به وسیله ی پایپ و بازگرداندن نتیجه ی به دست آمده به عنوان result

```
string exec(const char* cmd) {
    array<char, 128> buffer;
    string result;
    unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);
    if (!pipe) {
        throw runtime_error("popen() failed!");
    }
    while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
        result += buffer.data();
    }
    return result;
}
```

is_exist:

چک کردن اینکه آیا فایل یا دایرکتوری وجود دارد یا خیر.

```
bool CommandHandler::is_exist(string name)
{
    struct stat buffer;
    return (stat (name.c_str(), &buffer) == 0);
}
```

۲) ConfigurationParser

اطلاعات مربوط به فایل config در این کلاس نگهداری شده و در صورت نیاز با دسترسی به آن میتوان به اطلاعاتی اعم از اطلاعات یوزرها ، فایل های مجاز ، و همچنین پورت های چنل دیتا و کامند دسترسی یافت .

```cpp
class ConfigurationParser
{
public:
    ConfigurationParser(const std::string& config_file_path);
    std::vector<User*> get_users();
    std::vector<std::string> get_files();
    int get_command_channel_port();
    int get_data_channel_port();

private:
    void parse();
    int command_channel_port;
    int data_channel_port;
    std::string config_file_path;
    std::vector<User*> users;
    std::vector<std::string> files;
};
```

Attributes:

command_channel_port: پورت چنل کامند (مورد نیاز برای کلاینت ها)

data_channel_port: پورت چنل دیتا (مورد نیاز برای کلاینت ها)

config_file_path: مسیر فایل config.json

users: وکتوری از یوزرهایمان که اطلاعات خوانده شده ی یوزر ها در آن نگه داری میشود

files: وکتوری از نام فایل های مجاز یوزرهایمان

Methods:

Parse: برای خواندن اطلاعات فایل config.json صدا زده میشود و به شکل زیر عمل میکند :

```
void ConfigurationParser::parse()
{
    json json;
    ifstream file(config_file_path);
    file >> json;
    command_channel_port = json["commandChannelPort"].get<int>();
    data_channel_port = json["dataChannelPort"].get<int>();
    for (auto& user: json["users"])
    {
        users.push_back(
            new User(
            user["user"].get<string>(),
            user["password"].get<string>(),
            user["admin"].get<string>() == "true",
            stoi(user["size"].get<string>()))
        );
    }
    for (auto& file: json["files"])
        files.push_back(file.get<string>());
}
```

که همانگونه که مشخص است اطلاعات را خوانده و به ازای هر یوزر یک یوزر ساخته و به وکتور یوزرها اضافه میکند .

۳) ConnectedUser

اطلاعات مربوط به یوزر هایی که به سرور متصل شده اند را نگه میدارد ، اعم از وضعیت login ، directory کنونی آن و ...

```
class ConnectedUser
{
public:
    ConnectedUser(int command_socket, int data_socket);
    void set_user(User* _user);
    void set_is_username_entered(bool _state);
    void set_is_password_entered(bool _state);
    void set_current_directory(std::string path);
    User* get_user();
    int get_command_socket();
    int get_data_socket();
    bool get_is_username_entered();
    bool get_is_passsword_entered();
    std::string get_current_directory();

private:
    User* user;
    int command_channel_socket_fd;
    int data_channel_socket_fd;
    bool is_username_entered;
    bool is_passsword_entered;
    std::string current_directory;
};
```

Attributes:

User: یوزر مربوطه در وکتور users در ConfigurationParser

fd: file descriptor یوزر وصل شده به سرور

command_channel_socket_fd و data_channel_socket_fd:

fd های کانال دیتا و کامند

is_user_name_entered و is_password_entered:

بزای بررسی وضعیت لاگین و اینکه در کدام مرحله ی لاگین است . (username و password وارد شده اند یا خیر).

current_directory: directory کنونی کلاینت مان

Methods:

decrease_available_size: برای کاهش حجم کنونی یوزر پس از دانلود فایل

is_able_to_download: برای بررسی ملزومات کافی برای دانلود فایل

is_admin_user: برای بررسی admin بودن یا یوزر عادی بودن

۴) Server

برای ایجاد و بایند کردن سرورمان.

به دلیل بلاکینگ بودن سیستم کال ها و برای اینکه سرور بتواند همزمان به چند کلاینت پاسخ دهد ، از selectاستفاده شده است .

```
class Server
{
public:
    void run_server();
    int run_socket(int port);

    static inline std::vector<std::string> files;
    static inline int command_channel_port;
    static inline int data_channel_port;
};
```

Methods:

run_server: برای برقراری سرور

run_socket: برای بایند کردن سرور به سوکت مناسب و بقیه کارهای اجرایی که به شکل زیر است:

```
int Server::run_socket(int port)
{
    struct sockaddr_in server_sin;
    int server_socket_fd;
    server_sin.sin_port = htons(port);
    server_sin.sin_addr.s_addr = inet_addr("127.0.0.1");;
    server_sin.sin_family = AF_INET;
    int opt = 1;

    if ((server_socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        cout << "Failed to create a socket" << endl;
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_socket_fd, SOL_SOCKET, SO_REUSEADDR, (char*)&opt, sizeof(int)) < 0)
    {
        cout << "Failed to set socket option" << endl;
        exit(EXIT_FAILURE);
    }

    if (bind(server_socket_fd, (struct sockaddr*)& server_sin, sizeof(server_sin)) < 0)
    {
        cout << "Failed to bind a socket" << endl;
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket_fd, MAX_CONNECTIONS) < 0)
    {
        cout << "Failed to listen" << endl;
        exit(EXIT_FAILURE);
    }

    return server_socket_fd;
}
```

و طبق گفته های کلاس پیاده سازی شده است .

۵) UserManager

تمام کارهای مربوط به مدیریت یوزرها ، اعم از اطلاعات اولیه یوزرها و وضعیت کنونی آن ها ، پیدا کردن یوزر ها بر اساس پارامتر های مناسب ، حذف و اضافه یوزرها و ... به عهده این کلاس است و به users و connected_users دسترسی دارد .

```
class UserManager
{
public:

    static User* find_user_by_username(std::string user_name);
    static ConnectedUser* get_connected_user_by_fd(int socket);
    static void add_connected_user(int command_socket, int data_socket);
    static void remove_connected_user(int socket);

    static std::vector<User*> users;
    static std::vector<ConnectedUser*> connected_users;
};
```

۶) User

همانند connected_user ، با این تفاوت که اطلاعات اولیه یوزرها را که از فایل کانفیگ خوانده شده شامل است و به وضعیت کنونی یوزرها ارتباطی ندارد.

```
class User
{
public:
    User( const std::string& username, const std::string& password, const bool& is_admin, double available_size);
    std::string get_username();
    std::string get_password();
    bool is_admin_user();
    double get_available_size();
    void decrease_available_size(double file_size);

private:
    std::string username;
    std::string password;
    bool is_admin;
    double available_size;

};
```

Attributes:

is_admin : برای بررسی یوزر عادی یا ادمین بودن

available_size: سایز دانلود باقی مانده برای یوزر

fd: file descriptor یوزر متصل شده

Methods:

Is_matched_with : برای بررسی تطابق یوزرنیم و پسورد

Decrease_available_size: برای کاهش سایز باقی مانده یوزر پس از دانلود

۷) Client

کلاس کلاینت مان که اجرا شده و با داشتن پورت های لازم (پورت کانال کامند و دیتا) به سرور متصل شده و شروع به رد و بدل اطلاعات میکند .

```
class Client
{
public:
    void run_client();
    void write_downloaded_data_to_file(char buffer[MESSAGE_BUFFER_SIZE]);

    static inline int command_channel_port;
    static inline int data_channel_port;
    static inline int number_of_downloaded_file = 0;
};
```

run_client: برای اجرای کلاینت و ساختن کلاینت جدید به شکل زیر:

```cpp
void Client::run_client()
{
    int client_socket_fd, client_data_socket_fd;
    if ((client_socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        cout << "Failed to create a socket." << endl;
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_command_address;
    server_command_address.sin_family = AF_INET;
    server_command_address.sin_port = htons(command_channel_port);
    if (inet_pton(AF_INET, "127.0.0.1", &server_command_address.sin_addr) <= 0) {
        cout << "The address is invalid or unsupported." << endl;
        exit(EXIT_FAILURE);
    }
    cout<<"command channel oppened!"<<endl;

    if (connect(client_socket_fd,(struct sockaddr*)&server_command_address, sizeof(server_command_address)) < 0)
    {
        cout << "Failed to establish a connection with the server." << endl;
        exit(EXIT_FAILURE);
    }
    cout<<"connected to server!"<<endl;

    if ((client_data_socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        cout << "Failed to create a data socket." << endl;
        exit(EXIT_FAILURE);
    }
```