



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
درس شبکه های کامپیوتری
دکتر یزدانی

پروژه اول

نام و نام خانوادگی	دانیال سعیدی (810198571) محمد قره حسنلو (810198461)
تاریخ ارسال گزارش	یکشنبه - ۱۴ فروردین ۱۴۰۱

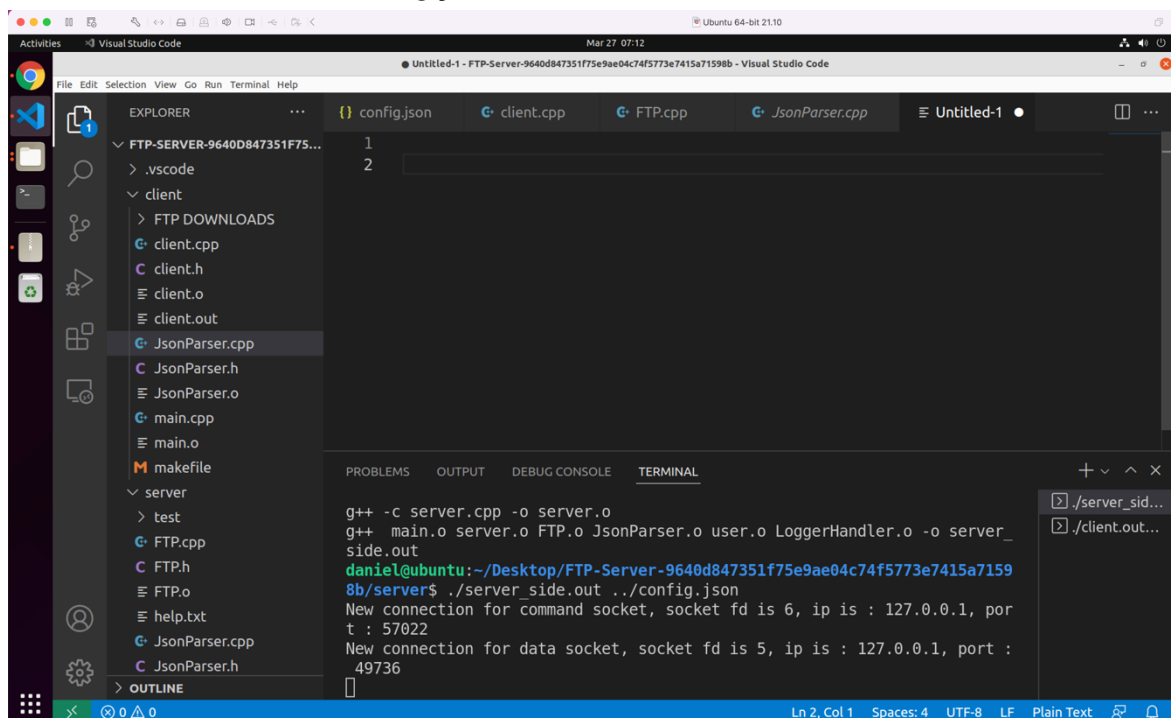
۱. نحوه اجرای برنامه

برای اجرا برنامه باید دستور های زیر به ترتیب در **Command Line** وارد شوند:
اجرا سرور:

- **cd server**
- **make**
- **./server_side.out ../config.json**

اجرای کلاینت:

- **cd client**
- **make**
- **./client.out ../config.json**



```
g++ -c server.cpp -o server.o
g++ main.o server.o FTP.o JsonParser.o user.o LoggerHandler.o -o server_side.out
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/server$ ./server_side.out ../config.json
New connection for command socket, socket fd is 6, ip is : 127.0.0.1, port : 57022
New connection for data socket, socket fd is 5, ip is : 127.0.0.1, port : 49736
```

خروجی سرور هنگام اتصال به کلاینت Figure 1

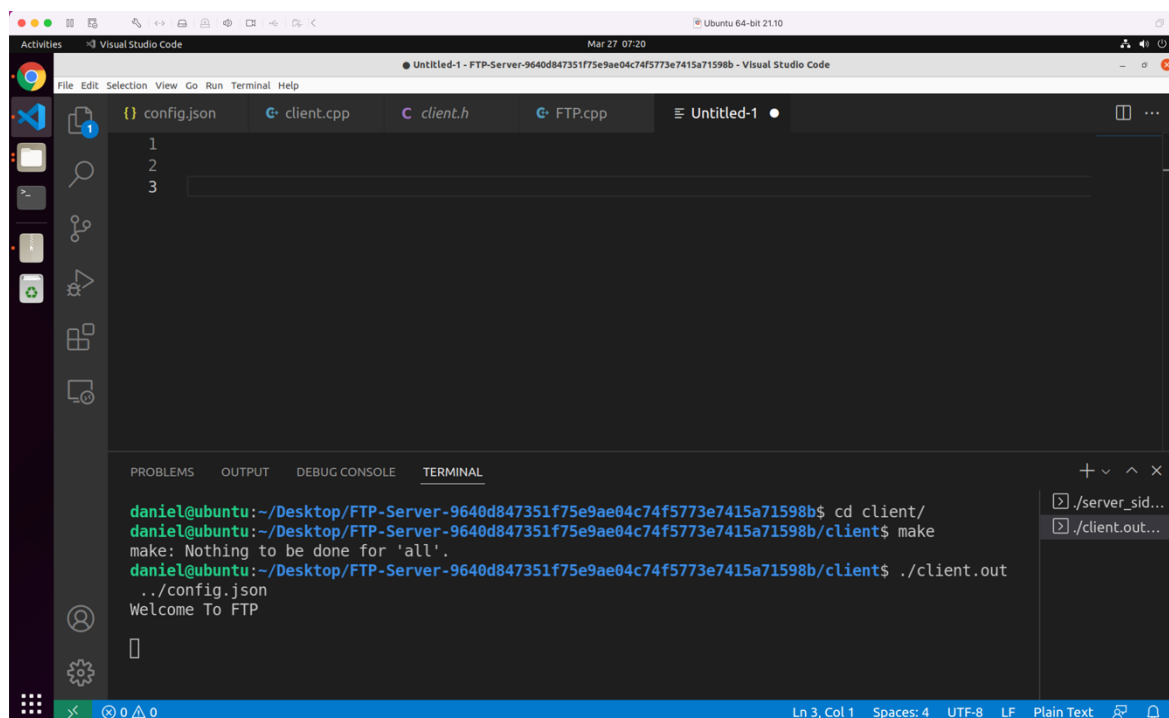


Figure 2 خروجی کلاینت هنگام اتصال به سرور

۲. نحوه خواندن فایل Json

قبل از پرداختن به بخش های client و server، باید فایل config.json را بخوانیم و اطلاعات زیر را از آن استخراج کنیم:

- خواندن لیست کاربران و نوع دسترسی آنان
- پورت دستورات (commandChannelPort)
- پورت کانال داده (dataChannelPort)

برای این منظور، کلاس JsonParser را تعریف کردیم. این کلاس دارای سه متد خواندن فایل از دیسک، پیدا کردن یک مقدار و جداسازی آرایه است.

```
class JsonParser{
public:
    string find_val_json(string json, string key);
    vector<string> split_array(string array);
    string read_json_file(string path);
};
```

قسمتی از پیاده سازی این قسمت از [اینجا](#) کمک گرفته شده است.

۳. بخش Server

برای این بخش کلاس FTP را تعریف کردیم. این کلاس وظیفه برقراری ارتباط، ارسال و دریافت اطلاعات را از کلاینت انجام می دهد.

```
class Server
{
public:
    // Server's Constructor
    Server(string config_file_path);
    vector<string> split_into_packets(string data, int packet_size);
    void print_info();
    void run();
private
    FTP system;
    vector<User*> read_users_config(string config);
    vector<string> read_files_config(string config);
    map<int, int> cmd_data_from_clients;
    vector<int> clients;
    int command_socket;
    int data_socket;
    void init_port(string config);
    void bind_sockets();
    void sockets_listen();
    void create_new_sockets();
    void set_sockets(int& max_socket_descriptor, fd_set& readfds);
    void accept_connections();
    void clients_req_handler(fd_set& readfds);
    void recieve_send_handler(int valread, char buffer[], int sd);
    int port_cmd_channel;
    int port_dchannel;
};
```

- در constructor این کلاس، JsonParser ساخته میشود و اطلاعات config.json خوانده شده و constructor کلاس FTP صدا زده میشود.
- Split_into_packets: این متد data را به packet هایی با اندازه packet_size تقسیم می کند.
- Print_info: اطلاعات سرور را چاپ می کند.
- Run: با اجرای این متد، سرور شروع به اجرا میشود.
- Create_new_sockets: این متد سوکت جدید را ایجاد میکند.
- Accept_connections: این متد accept کردن کلاینت هایی که میخواهد به سرور متصل شوند را به عهده دارد.
- Clients_req_handler: این متد وظیفه انجام دستورات از سمت کلاینت را بر عهده دارد.

- Receive_send_handler: این تابع وظیفه ارسال و دریافت به کلاینت را بر عهده دارد.

۴. کلاس Users

این کلاس برای ذخیره سازی اطلاعات کاربر استفاده می شود. متد و متغیر های این کلاس واضح هستند و احتیاجی به توضیح خاصی ندارند.

```
class User{
public:
    User(string _username, string _password, long _size, bool _is_admin);
    int get_size();
    bool get_is_admin();
    string get_username();
    string get_password();
    void decrease_user_size(int decration_size);
private:
    string username;
    string password;
    long size;
    bool is_admin;
};
```

۵. توابع Logger

تابع زیر زمان و تاریخ را صورت string بر می گرداند:

```
string get_date_time()
{
    auto now = chrono::system_clock::now();
    auto in_time_t = chrono::system_clock::to_time_t(now);
    stringstream ss;
    ss << put_time(localtime(&in_time_t), "%Y-%m-%d %X");
    return ss.str();
}
```

تابع زیر اطلاعات را در فایل log.txt مینویسد.

```
void write_log(string message)
{
    ofstream ofs;
    ofs.open (LOG_DIR, ofstream::out | ofstream::app);
    ofs << get_date_time() + " :: " + message + "\n";
    ofs.close();
}
```

۶. کلاس FTP

این کلاس شامل لیست کاربران و فایل های ادمین و کلاینت های آنلاین است. همچنین این کلاس دستور های مختلف کاربران اعم از login، mkd، quit و ... را اجرا می کند.

```
class FTP
{
public:
    FTP(vector<User*> users_list, vector<string> admin_files_list);
    FTP() = default;

    string get_user_data(int client_sd);
    vector<User*> get_users();
    vector<string> get_admin_files();
    void remove_user(int client_sd);
    string cmd_handler(char command[], int client_sd);
    bool has_user_data(int client_sd);
private:
    vector<User*> users_vec;
    vector<string> admin_files;
    string default_dir;
    map<int, client_ftp*> connected_users;

    // Command Handlers
    string password_command(vector<string> args, int client_sd);
    string mkd_command(string path, int client_sd);
    string help_command(int client_sd);
    string quit_command(int client_sd);
    string dele_command(string type, string path, int client_sd);
    string cwd_command(string path, int client_sd);
    string rename_command(string old_name, string new_name, int client_sd);
    string download_command(string file_name, int client_sd);
    string ls_command(int client_sd);
    string pwd_command(int client_sd);
    string user_command(vector<string> args, int client_sd);
    // Helper Functions
    User* find_user(string username);
    bool file_exists(string file_name, string directory);
    bool does_file_belong_to_admin(string file_name);
    client_ftp* create_user(User* user);
};
```

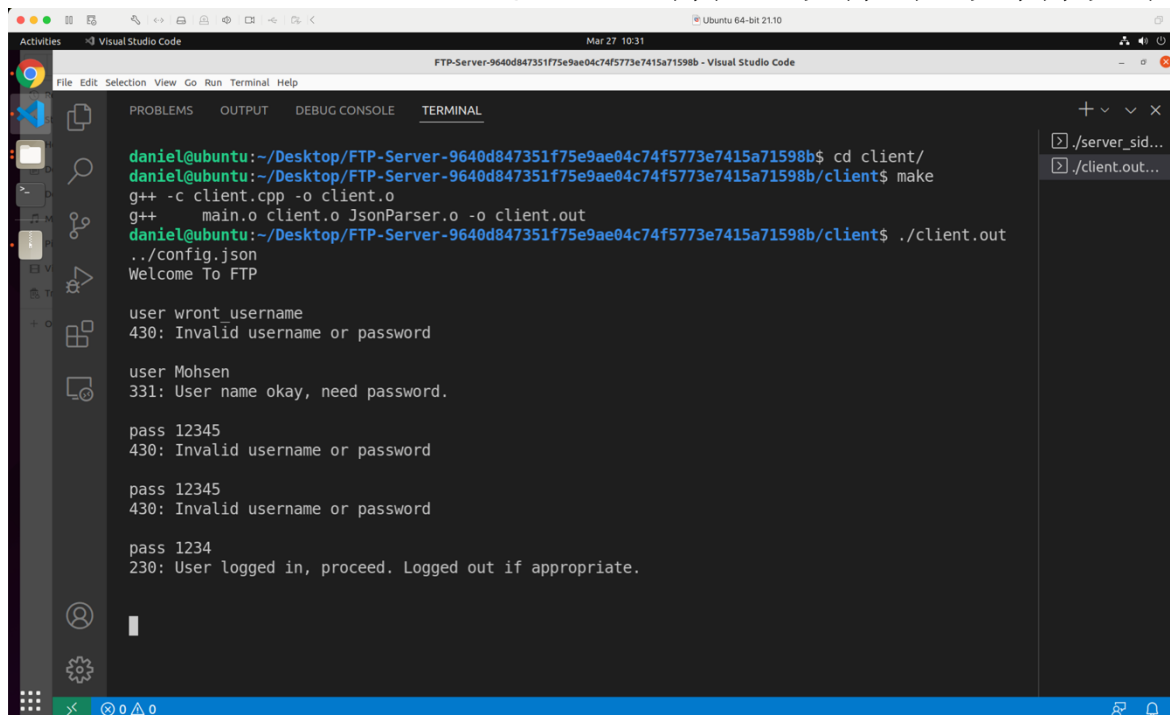
استراکت client_ftp به صورت زیر تعریف شده است:

```
struct client_ftp
{
    bool has_data;
    User* user_info;
    string current_dir;
    string data;
    bool is_authorized;
```

```
}
```

۶. احراز هویت و مدیریت دسترسی

در تصویر زیر نمونه از احراز هویت کاربر را مشاهده می کنید:



```
daniel@ubuntu:~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b$ cd client/
daniel@ubuntu:~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/client$ make
g++ -c client.cpp -o client.o
g++ main.o client.o JsonParser.o -o client.out
daniel@ubuntu:~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/client$ ./client.out
../config.json
Welcome To FTP

user wront username
430: Invalid username or password

user Mohsen
331: User name okay, need password.

pass 12345
430: Invalid username or password

pass 12345
430: Invalid username or password

pass 1234
230: User logged in, proceed. Logged out if appropriate.
```

۷. دایرکتوری فعلی

```
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a7159
8b$ cd client/
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a7159
8b/client$ make
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a7159
8b/client$ ./client.out ../config.json
Welcome To FTP

user Ali
331: User name okay, need password.

pass 1234
230: User logged in, proceed. Logged out if appropriate.

pwd
257: /home/daniel/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71
598b/server
```

۸. ساخت دایرکتوری جدید

```
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a7159
8b/client$ ./client.out ../config.json
Welcome To FTP

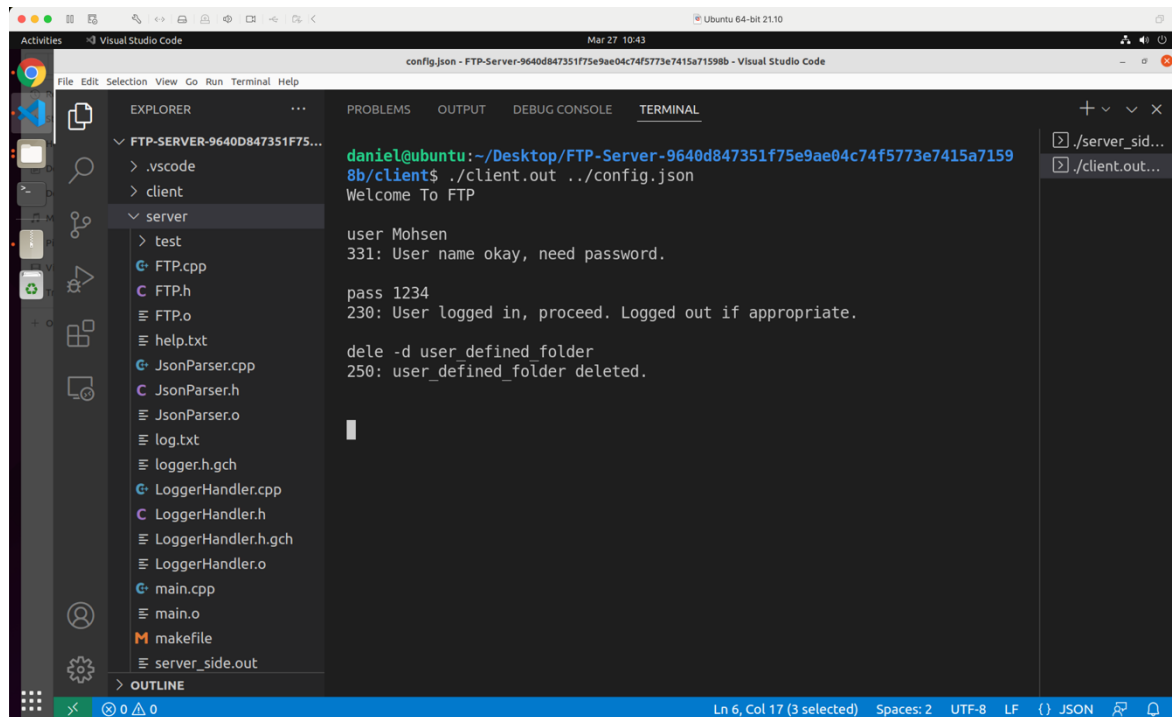
user Mohsen
331: User name okay, need password.

pass 1234
230: User logged in, proceed. Logged out if appropriate.

pwd
257: /home/daniel/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71
598b/server

mkd user_defined_folder
257: user_defined_folder created.
```


۹. حذف دایرکتوری یا فایل



```
config.json - FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b - Visual Studio Code

EXPLORER
  FTP-SERVER-9640D847351F75...
    > .vscode
    > client
    > server
      > test
      FTP.cpp
      FTP.h
      FTP.o
      help.txt
      JsonParser.cpp
      JsonParser.h
      JsonParser.o
      log.txt
      logger.h.gch
      LoggerHandler.cpp
      LoggerHandler.h
      LoggerHandler.h.gch
      LoggerHandler.o
      main.cpp
      main.o
      makefile
      server_side.out
    > OUTLINE

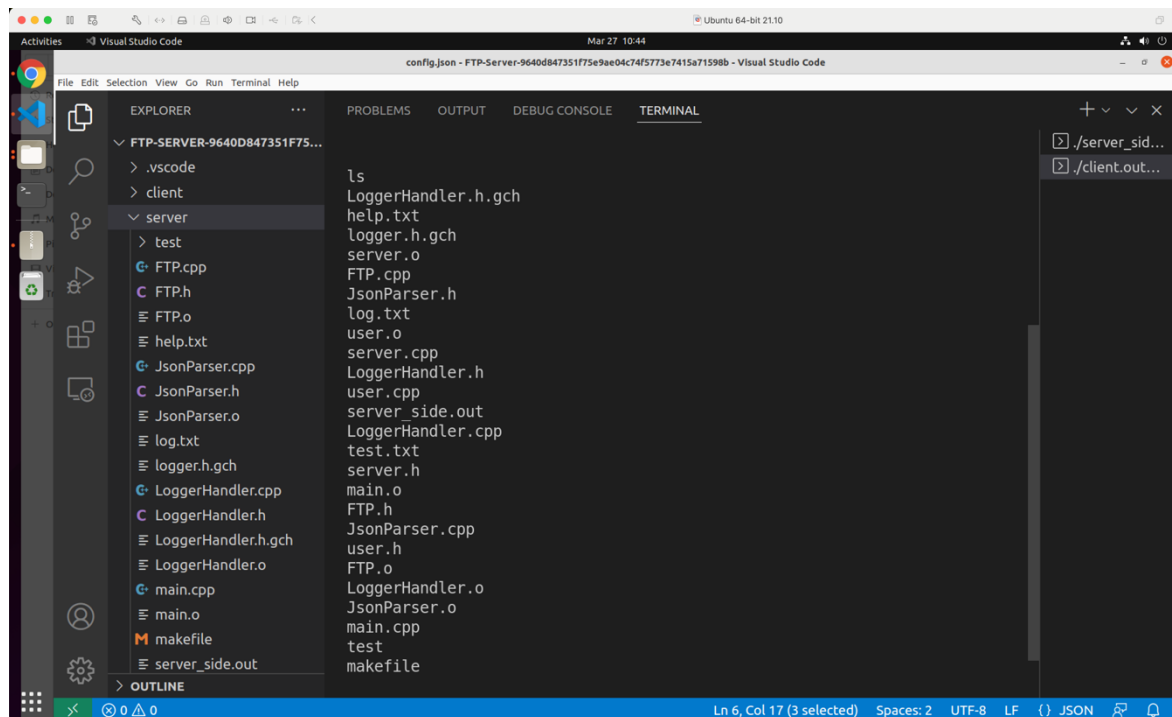
TERMINAL
daniel@ubuntu: ~/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/client$ ./client.out ../config.json
Welcome To FTP

user Mohsen
331: User name okay, need password.

pass 1234
230: User logged in, proceed. Logged out if appropriate.

dele -d user_defined_folder
250: user_defined_folder deleted.
```

۱۰. دستور ls



```
config.json - FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b - Visual Studio Code

EXPLORER
  FTP-SERVER-9640D847351F75...
    > .vscode
    > client
    > server
      > test
      FTP.cpp
      FTP.h
      FTP.o
      help.txt
      JsonParser.cpp
      JsonParser.h
      JsonParser.o
      log.txt
      logger.h.gch
      LoggerHandler.cpp
      LoggerHandler.h
      LoggerHandler.h.gch
      LoggerHandler.o
      main.cpp
      main.o
      makefile
      server_side.out
    > OUTLINE

TERMINAL
ls
LoggerHandler.h.gch
help.txt
logger.h.gch
server.o
FTP.cpp
JsonParser.h
log.txt
user.o
server.cpp
LoggerHandler.h
user.cpp
server_side.out
LoggerHandler.cpp
test.txt
server.h
main.o
FTP.h
JsonParser.cpp
user.h
FTP.o
LoggerHandler.o
JsonParser.o
main.cpp
test
makefile
```

۱۰. عوض کردن دایرکتوری

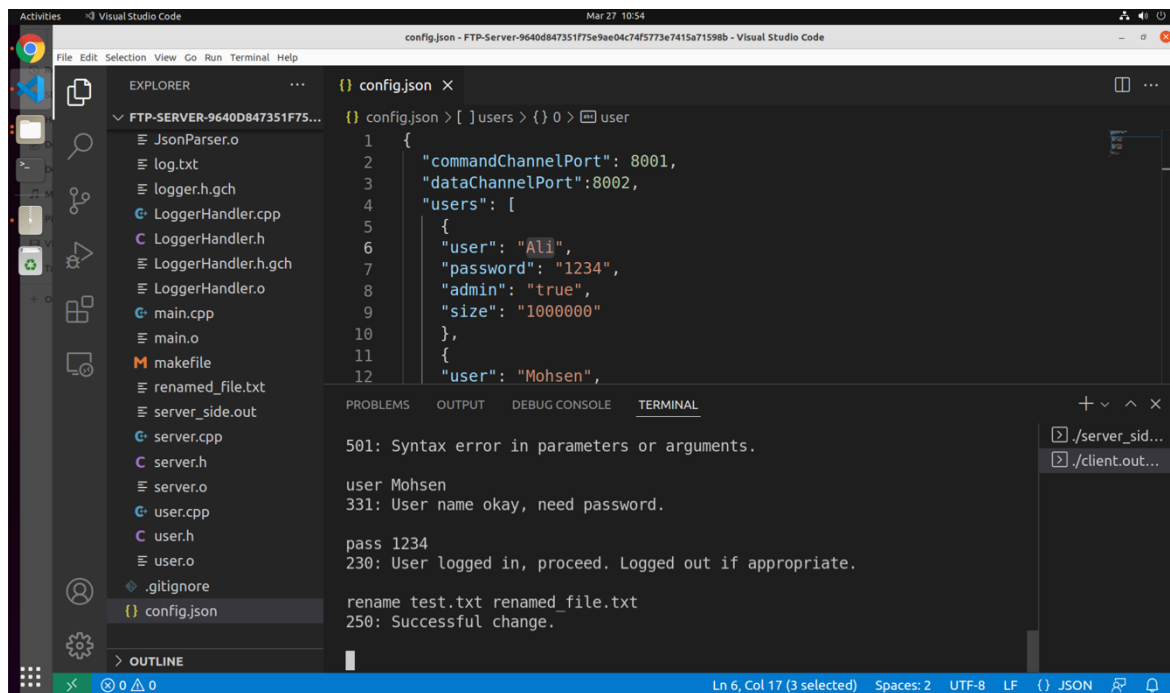
```
250: Successful change.

pwd
257: /home/daniel/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/server/test

cwd
250: Successful change.

pwd
257: /home/daniel/Desktop/FTP-Server-9640d847351f75e9ae04c74f5773e7415a71598b/server
```

۱۱. عوض کردن نام فایل



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left displays the file structure of the project, including files like log.txt, LoggerHandler.cpp, main.cpp, and config.json. The main editor window shows the config.json file with the following content:

```
{
  "commandChannelPort": 8001,
  "dataChannelPort": 8002,
  "users": [
    {
      "user": "Ali",
      "password": "1234",
      "admin": "true",
      "size": "1000000"
    },
    {
      "user": "Mohsen",

```

The TERMINAL panel at the bottom shows the output of the FTP client commands:

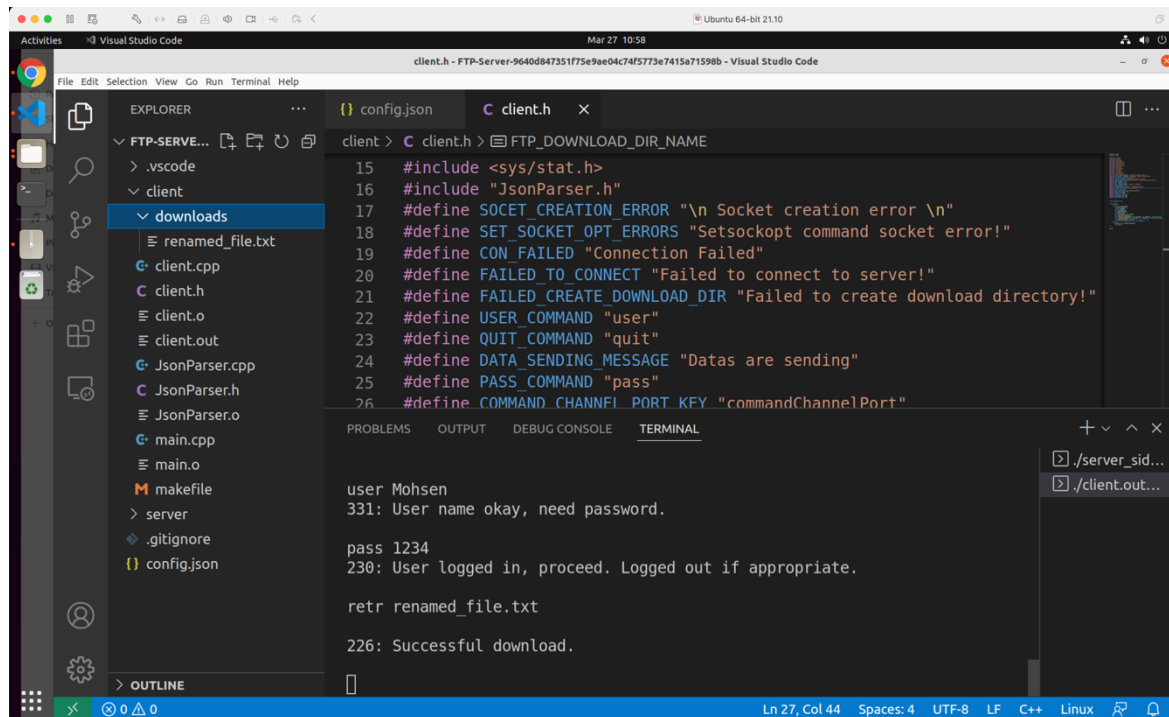
```
501: Syntax error in parameters or arguments.

user Mohsen
331: User name okay, need password.

pass 1234
230: User logged in, proceed. Logged out if appropriate.

rename test.txt renamed_file.txt
250: Successful change.
```

۱۱. دانلود فایل



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the project structure with a 'downloads' folder. The main editor displays the 'client.h' file, which contains various preprocessor directives for error handling and command definitions. The Terminal panel at the bottom shows the execution of the client program, including prompts for username, password, and file name, and the successful download of 'renamed_file.txt'.

```
15 #include <sys/stat.h>
16 #include "JsonParser.h"
17 #define SOCKET_CREATION_ERROR "\n Socket creation error \n"
18 #define SET_SOCKET_OPT_ERRORS "Setsockopt command socket error!"
19 #define CON_FAILED "Connection Failed"
20 #define FAILED_TO_CONNECT "Failed to connect to server!"
21 #define FAILED_CREATE_DOWNLOAD_DIR "Failed to create download directory!"
22 #define USER_COMMAND "user"
23 #define QUIT_COMMAND "quit"
24 #define DATA_SENDING_MESSAGE "Data are sending"
25 #define PASS_COMMAND "pass"
26 #define COMMAND_CHANNEL_PORT_KEY "commandChannelPort"
```

user Mohsen
331: User name okay, need password.

pass 1234
230: User logged in, proceed. Logged out if appropriate.

retr renamed_file.txt

226: Successful download.

۱۱. بخش Client

```
int main(int argc, char* argv[])
{
    Client client(argv[1]);
    client.run();
    return 0;
}
```

در شروع کار با سمت client تابع main فراخوانی میشود که در اینجا تابع run از کلاس Client فراخوانی میشود.

```
int Client::create_download_dir()
{
    struct stat st = {0};

    if (stat(FTP_DOWNLOAD_DIR_NAME, &st) == -1)
        return mkdir(FTP_DOWNLOAD_DIR_NAME, 0700);

    return 0;
}
```

در این تابع یک جا برای دانلود فایل از سمت سرور در پوشه Client فراهم میشود که نام آن در FTP_DOWNLOAD_DIR_NAME به صورت constant ذخیره کرده ایم و با mkdir یک directory ساختیم.

```
int Client::connect_on_port(int port){
    int sock, opt = 1;
    struct sockaddr_in serv_addr;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        cout << SOCKET_CREATION_ERROR;
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)&opt,
sizeof(opt)) < 0 )
    {
        cout << SET_SOCKET_OPT_ERRORS << endl;
        exit(EXIT_FAILURE);
    }
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) <
0)
    {
        cout << CON_FAILED << endl;
        return -1;
    }
    return sock;
}
```

در این تابع، در ابتدا یک socket تعریف میکنیم که برای ارتباط استفاده شده و یک descriptor برمیگرداند که آن را در sock ذخیره میکنیم تا در ادامه در setsockopt و connect استفاده کنیم و اگر مقدار بازگشتی کمتر از صفر باشد، در ساخت socket به ارور خورده ایم. ورودی socket به ترتیب domain، نوع(که از نوع stream است) و پروتکل است. در ادامه setsockopt را تعریف میکنیم که یک سری آپشن برای socket در نظر میگیریم که آرگومان اول همان sock است که در تابع قبل مقداردهی کردیم و در ادامه شماره سطح، شماره پروتکل مناسب را وارد میکنیم. در آخر با connect اتصال روی سوکت را مقداردهی میکنیم و به این صورت که یک ارتباط بین sock و serv_addr که با ساختار sockaddr_in ایجاد کردیم، برقرار میکنیم و در آخر sock را برمیگردانیم.

```
vector<string> Client::split(string str, char divider)
{
    stringstream ss(str);
    string word;
    vector< string> result;
```

```

while(getline(ss, word, divider))
{
    if(word != "")
        result.push_back(word);
}
return result;
}

```

این قسمت برای جدا کردن ورودی است که کاربر در ترمینال وارد کرده است تا بتوانیم در ادامه راحت تر به کاری که کاربر خواسته، پاسخ دهیم.

```

void Client::recieve_send_handler(string request, char response[],
vector<string> request_params)
{
    string data;
    int valread;

    bool is_download_req = request_params.size() > 0 && request_params[0]
== DOWNLOAD_COMMAND;

    send(command_socket , request.c_str() , strlen(request.c_str()) , 0 );

    memset(response, 0, strlen(response));
    valread = read(command_socket, response, PACKET_SIZE);

    if (strcmp(response, DATA_SENDING_MESSAGE) == 0)
    {
        data = "";

        memset(response, 0, strlen(response));
        valread = read(command_socket, response, PACKET_SIZE);
        int packet_size = atoi(response);

        float progress = 0.0;
        float step = 1.0 / (float)packet_size;
        int barWidth = 70;
        for(int i = 0; i < packet_size; i++)
        {
            memset(response, 0, strlen(response));
            valread = read(data_socket, response, PACKET_SIZE);
            string tmp = response;
            data += tmp;
            progress += step;
            if (i == packet_size - 1 && progress != 1)
                progress = 1.0;
            if (is_download_req)
            {
                cout.flush();
            }
        }
    }
}

```

```

    }
}
if (is_download_req)
{
    cout << endl;
    ofstream MyFile(string(FTP_DOWNLOAD_DIR_NAME) + "/" +
request_params[1]);
    MyFile << data;
    MyFile.close();
}
else
    cout << data << endl;
}
memset(response, 0, strlen(response));
valread = read(command_socket , response, PACKET_SIZE);
cout << response << endl;
}

```

با استفاده از send، request را به command_socket ارسال میکنیم. با استفاده از memset از 0 را به اندازه طول response در response ذخیره میکنیم. متغیری به نام data تعریف میکنیم. دستور ساکت را در ابتدا میخوانیم و سپس به اندازه packet_size، دیتا ساکت را میخوانیم که این کار با read انجام میشود و هر بار data را آپدیت میکنیم. در انتها اگر درخواست دانلود باشد، این فایل در پوشه FTP_DOWNLOAD_DIR_NAME دانلود میشود.

```

void Client::username_update(short req_type, char response[],
vector<string> request_params)
{
    vector<string> response splitted = split(response, ':');

    if (req_type == 1)
        username = stoi(response splitted[0]) == SUCCESSFUL_USER ?
request_params[1] : username;
    else if (req_type == 2)
        is_logged_in = stoi(response splitted[0]) == SUCCESSFUL_PASS ? true
: false;
    else if (req_type == 3)
    {
        if (stoi(response splitted[0]) == SUCCESSFUL_QUIT)
        {
            username = "";
            is_logged_in = false;
        }
    }
}
}

```

با استفاده از این تابع، با توجه به تایپ دستوری که از ورودی گرفتیم، username یا password را آپدیت میکنیم یا عملیات logout را انجام میدهیم.

```

void Client::run()

```

```

{
    if (create_download_dir() != 0)
    {
        cout << FAILED_CREATE_DOWNLOAD_DIR << endl;
        exit(EXIT_FAILURE);
    }

    data_socket = connect_on_port(port_dchannel1);
    command_socket = connect_on_port(port_cmd_channel);
    if (data_socket <= 0 || command_socket <= 0)
    {
        cout << FAILED_TO_CONNECT << endl;
        exit(EXIT_FAILURE);
    }

    char response[PACKET_SIZE] = {0};
    string request;

    int valread = read(command_socket , response, PACKET_SIZE);
    cout << response << endl;
    while(1)
    {
        getline(cin, request);
        vector<string> request_params = split(request);
        bool is_user_req = request_params.size() > 0 && request_params[0]
== USER_COMMAND;
        bool is_pass_req = request_params.size() > 0 && request_params[0]
== PASS_COMMAND;
        bool is_quit_req = request_params.size() > 0 && request_params[0]
== QUIT_COMMAND;

        recieve_send_handler(request, response, request_params);

        if (is_user_req || is_pass_req || is_quit_req)
        {
            short type = is_user_req ? 1 : is_pass_req ? 2 : 3;
            username_update(type, response, request_params);
        }
    }
}

```

در اینجا با استفاده از تابع هایی که بالاتر ساختیم، پروسه run به صورت بالا پیاده سازی میشود که در ابتدا یک پوشه برای جای دانلود میسازیم، سپس برای دیتا و دستور، به پورت وصل میشویم و وارد یک لوپ میشویم که دستورات را در آنجا وارد میکنیم و با توجه به ورودی که در request ذخیره میشود، دستورات مختلف مانند user، pass و... اجرا میشوند و اگر تغییری در user یا pass ایجاد شود یا logout کنیم، عملیات لازم در آخر لوپ انجام میشود.