

Networks Assignment 1

Abdelrhman Abdelftah Kassem
18010948

Problem Statement

Create a client and a server application which can send and receive files to and from each other using HTTP requests with persistent connection.

Overall Organization

Code is divided into two parts client and server, client part has 4 files 2 header files called client.h and request.h, and two cpp files called client.cpp and request.cpp

For client side, Header files are for the include statements and function declarations, while the client.cpp file is responsible for connecting the client to the server and reading the commands.txt files and calling a function to handle the requests, request.cpp is where requests are send and response are received so it has all the main functions.

While server side also has two header files for the same purpose and two cpp files, server.cpp who is responsible for creating the server socket, listening for incoming connections and handling these connections by creating execution threads for them, also it handles the persistent time out feature so that it disconnects the client and end his thread in case of inactivity or client closing the connection, the other file is response.cpp which receives the user requests and handles them by sending the appropriate response and saving / reading the required files.

Important Functions:

Persistent_timeout: responsible for detecting client inactivity

```
/*sets the socket timeout option on receive to a value that takes into
account the number of active clients*/
void persistent_timeout(int client_descriptor)
{
    struct timeval t;
    int timeout = (MAX_ALLOWED_CONNECTIONS / current_clients) * 2;
    t.tv_sec = timeout;
    t.tv_usec = 0;
    setsockopt(client_descriptor, SOL_SOCKET, SO_RCVTIMEO, &t, sizeof t);
    printf("timeout set to: %d for client : %d\n", timeout,
client_descriptor);
}
```

Connection handler is the client thread function which services the client's requests or end his thread of execution.

```
void connection_handler(int client_descriptor)
{
    while (true)
    {
        persistent_timeout(client_descriptor);
        if (request_handler(client_descriptor) < 0)
            break;
    }
    printf("connection with client %d was terminated\n",
client_descriptor);
    current_clients--;
    int x = current_clients;
    printf("number of current clients = %d\n", x);
    return;
}
```

server -side request receiving:

```
//receives a request from the client and calls the right method base on
the request
void receive_request(char request [], int client_descriptor)
{
    if (strncmp(request, "GET", 3) == 0)
    {
        printf("GET request recognised\n");
        receive_GET_request(client_descriptor);
        printf("GET request complete\n\n\n");
    }
}
```

```
}  
else if (strncmp(request, "POST", 4) == 0)  
{  
    printf("POST request recognised\n");  
    receive_POST_request(client_descriptor);  
    printf("POST request complete\n\n\n");  
}  
else  
    printf("Unrecognised request\n");  
}
```

Parsing the commands file

```
Command parse_command(string command)  
{  
    vector<string> tokens = tokenize(command);  
    Command res;  
    res.request_type = tokens[0];  
    res.file_path = tokens[1];  
    res.host_name = tokens[2];  
    if (tokens.size() == 4)  
        res.port_number = tokens[3];  
    return res;  
}
```

Waiting for server to send file received confirmation after POST request

```
// wait for server to confirm file is received
void wait_for_POST_response(int client_descriptor, string file_path)
{
    struct timeval t;
    t.tv_sec = 5;
    t.tv_usec = 0;

    int res = setsockopt(client_descriptor, SOL_SOCKET, SO_RCVTIMEO, &t,
sizeof t);

    char buffer[1024];
    int bytes_read = recv(client_descriptor, buffer, 1024, 0);

    if (errno == EWOULDBLOCK || errno == EAGAIN || errno == EINPROGRESS ||
bytes_read == -1)
    {
        printf("File receive confirmation not received\n");
        return;
    }

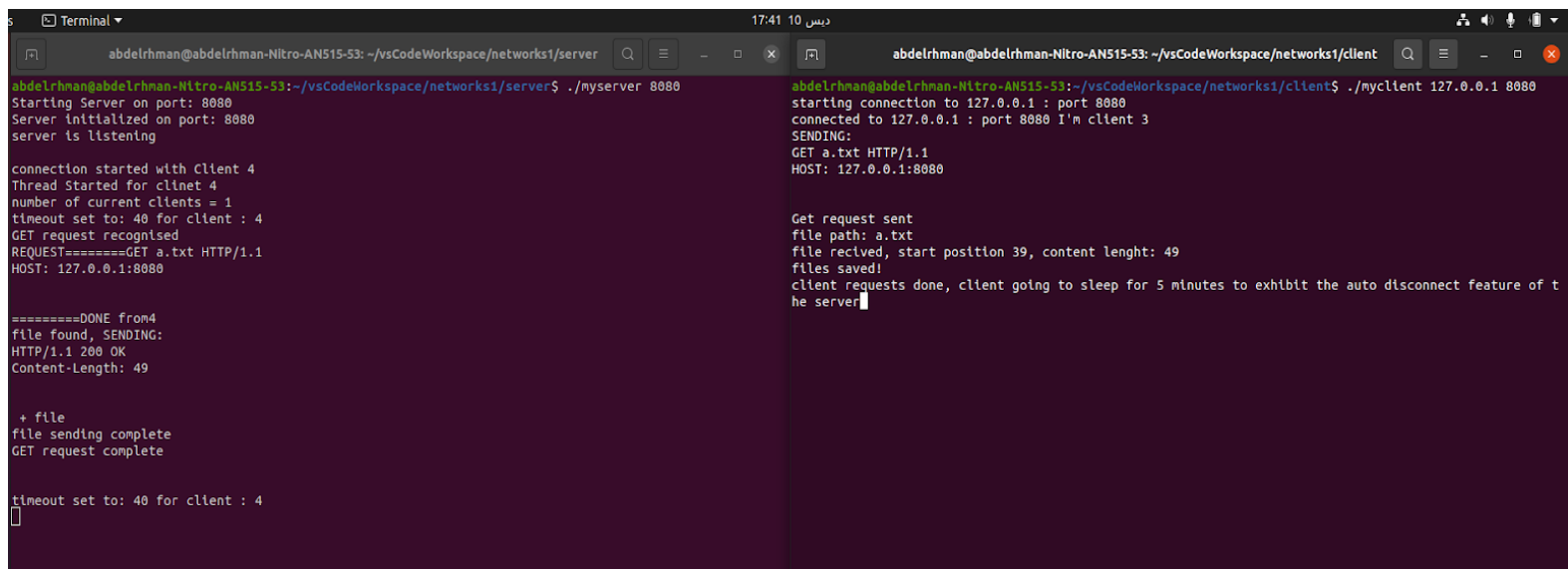
    string response = arr_to_str(buffer, bytes_read);
    string msg = "HTTP/1.1 200 OK\r\n";
    if (strncmp(response.c_str(), msg.c_str(), msg.length() - 2) == 0)
        printf("correct message received\n");
    cout << "file received at server confirmed by : " << msg << endl;
}
```

Data Structures Used

1. C++ std::string and stringstream for reading and storing requests and files
2. Char [] to receive data from the sockets using recv()
3. Struct called Command which has 4 strings (request type, file path, host name, host port number)

Sample Run:

Simple GET request of a.txt:



```
abdelrhman@abdelrhman-Nitro-ANS15-53: ~/vsCodeWorkspace/networks1/server$ ./myserver 8080
Starting Server on port: 8080
Server initialized on port: 8080
server is listening

connection started with Client 4
Thread Started for client 4
number of current clients = 1
timeout set to: 40 for client : 4
GET request recognised
REQUEST=====GET a.txt HTTP/1.1
HOST: 127.0.0.1:8080

=====DONE from4
file found, SENDING:
HTTP/1.1 200 OK
Content-Length: 49

+ file
file sending complete
GET request complete

timeout set to: 40 for client : 4
█

abdelrhman@abdelrhman-Nitro-ANS15-53: ~/vsCodeWorkspace/networks1/client$ ./myclient 127.0.0.1 8080
starting connection to 127.0.0.1 : port 8080
connected to 127.0.0.1 : port 8080 I'm client 3
SENDING:
GET a.txt HTTP/1.1
HOST: 127.0.0.1:8080

Get request sent
file path: a.txt
file recived, start position 39, content lenght: 49
files saved!
client requests done, client going to sleep for 5 minutes to exhibit the auto disconnect feature of t
he server█
```

Same request after 40 seconds of client inactivity (Note that client is using sleep() system call so client process isn't stopped).

```

abdelrhman@abdelrhman-Nitro-ANS15-53: ~/vsCodeWorkspace/networks1/server
abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/server$ ./myserver 8080
Starting Server on port: 8080
Server initialized on port: 8080
server is listening

connection started with Client 4
Thread Started for clienet 4
number of current clients = 1
timeout set to: 40 for client : 4
GET request recognised
REQUEST=====GET a.txt HTTP/1.1
HOST: 127.0.0.1:8080

=====DONE from4
file found, SENDING:
HTTP/1.1 200 OK
Content-Length: 49

+ file
file sending complete
GET request complete

timeout set to: 40 for client : 4
client 4 stopped sending requests so his execution thread will end
connection with client 4 was terminated
number of current clients = 0

abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/client
abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/client$ ./myclient 127.0.0.1 8080
starting connection to 127.0.0.1 : port 8080
connected to 127.0.0.1 : port 8080 I'm client 3
SENDING:
GET a.txt HTTP/1.1
HOST: 127.0.0.1:8080

Get request sent
file path: a.txt
file recived, start position 39, content lenght: 49
files saved!
client requests done, client going to sleep for 5 minutes to exhibit the auto disconnect feature of t
he server

```

Simple POST request of a.txt:

```

abdelrhman@abdelrhman-Nitro-ANS15-53: ~/vsCodeWorkspace/networks1/server
abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/server$ ./myserver 8080
Starting Server on port: 8080
Server initialized on port: 8080
server is listening

connection started with Client 4
Thread Started for clienet 4
number of current clients = 1
timeout set to: 40 for client : 4
POST request recognised
POST request of file: a.txt
start position 65, content lenght: 49
file recived, start position 65, content lenght: 49
SENDING :
HTTP/1.1 200 OK

POST request complete

timeout set to: 40 for client : 4

abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/client
abdelrhman@abdelrhman-Nitro-ANS15-53:~/vsCodeWorkspace/networks1/client$ ./myclient 127.0.0.1 8080
starting connection to 127.0.0.1 : port 8080
connected to 127.0.0.1 : port 8080 I'm client 3
SENDING :
POST a.txt HTTP/1.1
HOST: 127.0.0.1:8080
Content-Length: 49

file sent
correct message received
file received at server confirmed by : HTTP/1.1 200 OK

client requests done, client going to sleep for 5 minutes to exhibit the auto disconnect feature of t
he server

```


Assumptions:

For simplicity, the client always reads requests from a file called `commands.txt` next to the executable.

When the client or server receives a file it is saved next to the executable that is receiving the file.

How to use:

Source code is divided into two folders, `server` and `client`, each folder has a file named `compile.txt` which has the correct compile command of the project so both commands must be run

This will create two executables (`myserver`, `myclient`)

To run the server use the command → `./myserver 8080` (8080) can be changed to any port number or even omitted.

To run the client use the command → `./myclient 127.0.0.1 8080`

127.0.0.1 is the host name of the server and 8080 is the port number used to create the server.

This will prompt the client to read from `commands.txt` and send the request to the server.