**Networks Assignment 1 report**

**الاسم / عبدالرحمن أحمد يسري النعناعي 19015889**

**الاسم / محمود ابراهيم جاد ابوالوفا 19016532**

**The overall organization of the program :**

* **The program is divided in to two subprograms**
  + **A program simulating the server**
  + **A program simulating the client**
* **The server program is organized through running a main function which executes a while loop which**
  + - **First : listen for new connections**
    - **Accept new connection for incoming client**
    - **Parse the http1.1 request**
    - **Check if request is for GET or POST**
    - **if GET**
      * **Determine if file exists if so transmit its content**

**With success response http1.1 200 ok \r\n**

* + - * **if not return file not found error**

**With file not found response http1.1 404 \r\n**

* + - **if POST** 
      * **write the data sent to a file with specified path**

**With success response http1.1 200 ok \r\n**

* + - **wait for new connections**
    - **close if time out**

**time out is sent according to number of connections to the server as the number of connections is increasing, the time out limit is decreasing**

**as the number of connections is decreasing , the time out is increasing up to a limit 128 seconds**

* **The client program is organized through running a main function which executes a while loop which :**
  + - **Get the http request line by line then send it to the server when request is terminated (\r\n is entered)**
    - **if GET request :** 
      * **Receive the response from the server until the response is terminated**
    - **if POST request :**
      * **send the data to the server until data is terminated**
      * **receive the response from the server**
    - **if CLOSE request the connection with the server will be safely closed;**

**Major functions :**

**Server program :**

void get\_command(int new\_fd, char buffer[2048]) {

int numbytes;

int i = 0;

char command[2048];

memset(command, '\0', 2048 \* sizeof(char));

if ((numbytes = recv(new\_fd, command, 2048 - 1, 0)) == -1) {

perror("recv");

exit(1);

}

strcpy(buffer, command);

command[numbytes] = '\0';

printf("received request\n");

fflush(stdout);

}

**Used to receive command from client**

**used to parse the command sent from client and determine whether it's get or post or other**

void parse\_command(int new\_fd, char \*buf, char (\*parsed)[1024]) {

char \*token;

char \*rest = buf;

int i = 0;

while ((token = strtok\_r(rest, " ", &rest))) {

strcpy(parsed[i], token);

i++;

}

}

void handle\_get(int new\_fd, char \*path, char \*response) {  
 if (access(path, F\_OK) == 0) {  
 // file exists  
 read\_file(new\_fd, path, response);  
 } else {  
 strcpy(response, Notfound);  
 }  
}

**Handles the get response and if file exists then read and send it else response with file not found**

if (argc == 2) {

strcpy(PORT, argv[1]);

}

printf("%s ", PORT);

fflush(stdout);

char buf[MAXDATASIZE];

char parsed[MAX\_COMMANDS\_SPACES][1024];

int sockfd, new\_fd, numbytes; // listen on sock\_fd, new connection on new\_fd

// ftok to generate unique key

key\_t key = ftok("shmfile", 65);

// shmget returns an identifier in shmid

int shmid = shmget(key, 1024, 0666 | IPC\_CREAT);

// shmat to attach to shared memory

int \*num = (int\*) shmat(shmid, (void\*) 0, 0);

\*num = 0;

struct addrinfo hints, \*servinfo, \*p;

struct sockaddr\_storage their\_addr; // connector's address information

socklen\_t sin\_size;

struct sigaction sa;

int yes = 1;

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE; // use my IP

if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

// loop through all the results and bind to the first we can

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((sockfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol))

== -1) {

perror("server: socket");

continue;

}

if (setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(int))

== -1) {

perror("setsockopt");

exit(1);

}

const int enable = 1;

if (setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &enable, sizeof(int))

< 0)

printf("setsockopt(SO\_REUSEADDR) failed");

if (bind(sockfd, p->ai\_addr, p->ai\_addrlen) == -1) {

close(sockfd);

perror("server: bind");

continue;

}

break;

}

freeaddrinfo(servinfo); // all done with this structure

if (p == NULL) {

fprintf(stderr, "server: failed to bind\n");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) {

perror("listen");

exit(1);

}

sa.sa\_handler = sigchld\_handler; // reap all dead processes

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = SA\_RESTART;

if (sigaction(SIGCHLD, &sa, NULL) == -1) {

perror("sigaction");

exit(1);

}

printf("server: waiting for connections...\n");

**if file exists read and sent it in chunks of length 10000 to the client**

void send\_file(int new\_fd, char \*path) {

FILE \*fileptr;

long filelen;

fileptr = fopen(path, "rb"); // Open the file in binary mode

fseek(fileptr, 0, SEEK\_END); // Jump to the end of the file

filelen = ftell(fileptr); // Get the current byte offset in the file

rewind(fileptr);

char send\_buffer[10000]; // no link between BUFSIZE and the file size

int nb = fread(send\_buffer, 1, sizeof(send\_buffer), fileptr);

while (!feof(fileptr)) {

write(new\_fd, send\_buffer, nb);

nb = fread(send\_buffer, 1, 10000, fileptr);

}

write(new\_fd, send\_buffer, nb);

}

**used to write the data to a file if request is post request**

void receive\_file(int new\_fd, char \*path) {

printf("Reading Data\n");

int size = 10000;

char p\_array[size];

FILE \*fileSent = fopen(path, "wb");

int nb = read(new\_fd, p\_array, size);

while (nb > 0) {

if (strncmp(p\_array, Ok, strlen(Ok)) == 0)

break;

fflush(stdout);

fwrite(p\_array, sizeof(char), nb, fileSent);

nb = read(new\_fd, p\_array, size);

}

fclose(fileSent);

printf("Finished reading\n");

fflush(stdout);

sleep(1);

write(new\_fd, Ok, strlen(Ok));

}

**Child process handling the request coming**

if (!fork()) { // this is the child process

close(sockfd); // child doesn't need the listener

while (1) {

char buffer[2048];

memset(parsed, '\0', sizeof(parsed));

memset(buf, '\0', sizeof(buf));

memset(buffer, '\0', sizeof(buffer));

fd\_set readfds;

struct timeval tv;

FD\_ZERO(&readfds);

FD\_SET(new\_fd, &readfds);

if(\*num == 1) tv.tv\_sec = 128;

else tv.tv\_sec = 100 / log2(\*num);

printf("number of connections= %d \n", \*num);

printf("assigned timeout = %d \n", tv.tv\_sec);

rv = select(new\_fd + 1, &readfds, NULL, NULL, &tv);

if (rv == -1) {

perror("select"); // error occurred in select()

} else if (rv == 0) {

printf("Timeout occurred!\n");

break;

}

}

printf("Closing...\n");

fflush(stdout);

\*num = \*num - 1;

close(new\_fd);

exit(0);

}

get\_command(new\_fd, buffer);

printf("Server: received\n'%s'\n", buffer);

parse\_command(new\_fd, buffer, parsed);

if (strcmp(parsed[0], "GET") == 0) {

handle\_get(new\_fd, parsed[1]);

} else if (strcmp(parsed[0], "POST") == 0) {

receive\_file(new\_fd, parsed[1]);

} else if (strcmp(parsed[0], "CLOSE") == 0) {

if (send(new\_fd, Ok, strlen(Ok), 0) == -1)

perror("send");

break;

} else {

if (send(new\_fd, BadRequest, strlen(BadRequest), 0) == -1)

perror("send");

}

}

void sigchld\_handler(int s) {  
 // waitpid() might overwrite errno, so we save and restore it:  
 int saved\_errno = errno;  
  
 while (waitpid(-1, NULL, WNOHANG) > 0)  
 ;  
  
 errno = saved\_errno;  
}

**terminates the child processed after forking it**

void\* get\_in\_addr(struct sockaddr \*sa) {  
 if (sa->sa\_family == AF\_INET) {  
 return &(((struct sockaddr\_in\*) sa)->sin\_addr);  
 }  
  
 return &(((struct sockaddr\_in6\*) sa)->sin6\_addr);  
}

**Get the socket address**

**Client program :**

void receive\_file(int sockfd, char \*path) {

printf("Reading Data\n");

int size = 10000;

char p\_array[size];

char parsed[MAX\_COMMANDS\_SPACES][1024];

char \*token;

char \*rest = path;

int i = 0;

while ((token = strtok\_r(rest, "\\", &rest))) {

strcpy(parsed[i], token);

i++;

}

FILE \*recievedFile = fopen(parsed[i - 1], "wb");

int nb = read(sockfd, p\_array, size);

while (nb > 0) {

if (strncmp(p\_array, Ok, strlen(Ok)) == 0)

break;

fwrite(p\_array, sizeof(char), nb, recievedFile);

nb = read(sockfd, p\_array, size);

}

fclose(recievedFile);

printf("Finished reading\n");

fflush(stdout);

}

**Receive data from server in case of get request and write it to a file**

**Send the file to the server (post request)**

void send\_file(int soc\_fd, char \*path) {

fflush(stdout);

FILE \*fileptr;

long filelen;

char parsed[MAX\_COMMANDS\_SPACES][1024];

char \*token;

char \*rest = path;

int i = 0;

while ((token = strtok\_r(rest, "\\", &rest))) {

strcpy(parsed[i], token);

i++;

}

fileptr = fopen(parsed[i - 1], "rb"); // Open the file in binary mode

fseek(fileptr, 0, SEEK\_END); // Jump to the end of the file

filelen = ftell(fileptr); // Get the current byte offset in the file

rewind(fileptr);

char send\_buffer[10000]; // no link between BUFSIZE and the file size

int nb = fread(send\_buffer, 1, sizeof(send\_buffer), fileptr);

fflush(stdout);

while (!feof(fileptr)) {

write(soc\_fd, send\_buffer, nb);

nb = fread(send\_buffer, 1, 10000, fileptr);

}

write(soc\_fd, send\_buffer, nb);

sleep(1);

write(soc\_fd, Ok, strlen(Ok));

}

while (1) {

char input[2048];

memset(input, '\0', 2048 \* sizeof(char));

memset(parsed, '\0', sizeof(parsed));

memset(buf, '\0', sizeof(buf));

printf("Enter your command:\n");

do {

read = getline(&command, &len, stdin);

if (read == -1)

return -1;

command[read] = '\0';

strcat(input, command);

} while (strcmp(command, END) != 0);

if (send(sockfd, input, strlen(input), 0) == -1) {

perror("send");

break;

}

printf("Sent request\n");

parse\_command(input, parsed);

if (strcmp(parsed[0], "GET") == 0) {

if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {

perror("recv");

break;

}

buf[numbytes] = '\0';

printf("client: received '%s'\n", buf);

if (strcmp(buf, Ok) == 0) {

receive\_file(sockfd, parsed[1]);

}

}

} else if (strcmp(parsed[0], "POST") == 0) {

send\_file(sockfd, parsed[1]);

if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {

perror("recv");

break;

}

buf[numbytes] = '\0';

printf("client: received '%s'\n", buf);

} else if (strcmp(parsed[0], "CLOSE") == 0) {

if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {

perror("recv");

break;

}

buf[numbytes] = '\0';

printf("client: received '%s'\n", buf);

if (strcmp(buf, Ok) == 0) {

printf("GOOD BYE \n");

} else {

printf("ERROR CLOSING \n");

}

break;

} else {

if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {

perror("recv");

break;

}

buf[numbytes] = '\0';

printf("client: received '%s'\n", buf);

}

}

**Data structures :**

* **One dimensional Char array :**

**to store the (response and data) sent from server to client**

**and also to store the (request and data) sent from client to server**

* **Two dimensional Char array at the server :**

**to store the request sent from the client**

**after parsing it**

**How to use:**

GET (Path of the file on the server) HTTP/1.1\r\n

Host: www-net.cs.umass.edu\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n

Accept: text/html,application/xhtml+xml\r\n

Accept-Language: en-us,en;q=0.5\r\n

Accept-Encoding: gzip,deflate\r\n

Connection: keep-alive\r\n

\r\n

POST (Path of the file to save the file on the server) HTTP/1.1\r\n

Host: www-net.cs.umass.edu\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n

Accept: text/html,application/xhtml+xml\r\n

Accept-Language: en-us,en;q=0.5\r\n

Accept-Encoding: gzip,deflate\r\n

Connection: keep-alive\r\n

\r\n

**Lines other than command, the final \r\n can be neglected**