**الاسم / محمود ابراهيم جاد ابوالوفا 19016532**

**الاسم / عبدالرحمن أحمد يسري النعناعي 19015889**

**Networks Assigment1 report**

**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 100 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**
    - **close the connection with the server**

**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);  
 }  
  
 printf("The received command is %s", command);  
 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 \*response) {  
 char \*token;  
 char \*rest = buf;  
 int i = 0;  
  
 while ((token = strtok\_r(rest, " ", &rest))) {  
 strcpy(parsed[i], token);  
 i++;  
 }  
  
 if (strcmp(parsed[0], "GET") == 0) {  
 handle\_get(new\_fd, parsed[1], response);  
 } else if (strcmp(parsed[0], "POST") == 0) {  
 write\_file(parsed[1], parsed[3]);  
 strcpy(response, Ok);  
 } else if (strcmp(parsed[0], "CLOSE") == 0) {  
 strcpy(response, "CLOSE");  
 } else {  
 strcpy(response, "UNKNOWN REQUEST");  
 }  
}

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

void read\_file(int new\_fd, char \*path, char \*response) {  
 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);  
 //fseek(fileptr, 0, SEEK\_SET);  
 char send\_buffer[100000]; // no link between BUFSIZE and the file size  
  
 /\*if (!feof(fileptr)){  
  
 }\*/  
 while (!feof(fileptr)) {  
 printf("\n Iam here");  
 fflush(stdout);  
 int nb = fread(send\_buffer, 1, sizeof(send\_buffer), fileptr);  
 printf("\nsent buffer is %s ", send\_buffer);  
 printf("\n nb now is %d", nb);  
 fflush(stdout);  
 write(new\_fd, send\_buffer, nb);  
 //nb = fread(send\_buffer, 1, sizeof(send\_buffer), fileptr);  
 // no need to bzero  
 }  
}

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

void write\_file(char path[1024], char data[1024]) {  
 FILE \*fileptr;  
 fileptr = fopen(path, "w");  
  
 if (fileptr == NULL) {  
 printf("unable to create file ");  
 exit(EXIT\_FAILURE);  
 }  
  
 fputs(data, fileptr);  
 fclose(fileptr);  
  
}

**Main loop of the server program its function illustrated above**

while (1) { // main accept() loop  
 sin\_size = sizeof their\_addr;  
 new\_fd = accept(sockfd, (struct sockaddr\*) &their\_addr, &sin\_size);  
 if (new\_fd == -1) {  
 perror("accept");  
 continue;  
 }  
 inet\_ntop(their\_addr.ss\_family,  
 get\_in\_addr((struct sockaddr\*) &their\_addr), s, sizeof s);  
 printf("server: got connection from %s\n", s);  
  
 if (!fork()) { // this is the child process  
 close(sockfd); // child doesn't need the listener  
 while (1) {  
 fd\_set readfds;  
 struct timeval tv;  
  
 FD\_ZERO(&readfds);  
  
 FD\_SET(new\_fd, &readfds);  
 tv.tv\_sec = 1000;  
  
 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! No data after 100 seconds.\n");  
 break;  
 }  
  
 char buffer[2048];  
 char response[1000000];  
 memset(response, '\0', 1000000 \* sizeof(char));  
 get\_command(new\_fd, buffer);  
 printf("Server: received '%s'\n", buffer);  
 parse\_command(new\_fd, buffer, parsed, response);  
 printf("The sent response is %s", response);  
 if (send(new\_fd, response, strlen(response), 0) == -1)  
 perror("send");  
 if (strcmp(response, "CLOSE") == 0) {  
 break;  
 }  
 }  
 printf("Closing...\n");  
 fflush(stdout);  
 close(new\_fd);  
 exit(0);  
 }  
 close(new\_fd); // parent doesn't need this  
}

**terminates the child processed after forking it**

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;  
}

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 get\_file\_data(int sockfd, char \*buf) {  
 printf("Reading Picture Byte Array\n");  
 FILE \*fileptr;  
 fileptr = fopen("1.png", "w");  
  
 if (fileptr == NULL) {  
 printf("unable to create file ");  
 exit(EXIT\_FAILURE);  
 }  
  
 int size = 10000;  
 char p\_array[size];  
 char \*current = p\_array;  
 int nb = read(sockfd, current, size);  
 printf("\nrecived buffer is %s ", current);  
 fflush(stdout);  
 fputs(current, fileptr);  
 while (nb >= 0) {  
 current = current + nb;  
 nb = read(sockfd, current, size);  
 printf("%s ", current);  
 fflush(stdout);  
 fputs(current, fileptr);  
 }  
 fclose(fileptr);  
}

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

while (1) {  
 char input[2048];  
 memset(input, '\0', 2048 \* sizeof(char));  
 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);  
  
 printf("Completed input\n");  
 if (send(sockfd, input, strlen(input), 0) == -1) {  
 perror("send");  
 break;  
 }  
 printf("Sent request\n");  
  
 if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {  
 perror("recv");  
 break;  
 }  
 buf[numbytes] = '\0';  
 get\_file\_data(sockfd, buf);  
 printf("client: received '%s'\n", buf);  
}

**Main loop of client program its function is illustrated above**

**Get the socket address**

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);  
}

**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**