14. Esraa Mohamed Hashish
26. Rewan Alaa El Din
72. Mennatollah Salah El Din El Kammah

Networks Assignment 1

Overall Organization:

The server is run first. Clients connect to it. Clients start sending requests to the server. For GET requests from the clients, the server handles it by sending the required data to the client. Or POST requests, the server first sends an acknowledgment “ok 200” and then waits for the following request containing data to the server. When the client does not need the connection any more, it closes it, thus the servers also will know that and close that connection and after a while, this thread will terminate and will be reaped (reap the dead threads).
For persistency, the connection from the client remains open while dealing with the server with more than one request.

Client Side:

Functions and Data Structures:

No major data structures are used, any data structure is only locally used in a function and will be described below in its specific function.

void post_server(char* data, int sockfd);
It takes two parameters: a pointer to the char array of data to send and the socket number to send that data respectively.
This function is called whenever there is a message of size less than or equal to MAXDATASIZE required to be sent. Generally the whole data to be sent is divided first to chunks less than or equal to that MAXDATASIZE and then each chunk is sent on the corresponding required socket using this function. Thus this function can be called more than one time for large data.After finishing sending the whole data, an end of data specified by "\r\n" is sent meaning the end of data and so the next message will be sent to the server will be a new POST or GET request.

void read_commands(int sockfd, char *buf);

This function reads the POST or GET requests from a file called "example.txt" in order to generate and send POST or GET requests from the clients to the server. It opens the "example.txt" file, start reading it line by line and sends that request to the server. If the command is GET, then it receives the data from the server and starts writing it in a new file it (the client) creates, then the client closes that file. If the request was POST, then it waits for the server acknowledging "ok 200" and then starts sending the data in chunks of maximum size MAXDATASIZE using the "post_server" function explained above.

void read_file(string path, int sockfd);

This function is used in POST requests to send data to the server from the client. It divides the data into chunks of maximum size equals to MAXDATASIZE and then calls the "post_server" function to send each chunk. It takes a parameter the file name with its extension, checks whether it is a text file or html file or an image. The other parameter is the socket number to send data to. The file to be sent is read character by character or byte by byte till the "MAXDATASIZE - 2" in order to manually write the end of buffer "\0" to end it. For th las chunk, the buffer "buf" used to store and sent data is freed.

vector<string> split(char *str,string sep);

It takes two parameters: a pointer to a string (str) to split on the second parameter (sep) to split the string on it. It returns a vector containing the result of slitting. This function uses "strtok" which keeps on splitting the string whenever finding the separating substring to split on. It implementation internally automatically advances a pointer to that. This function will be used mainly in reading the commands file.

void *get_in_addr(struct sockaddr *sa);

It checks the the socket address to whether use IPv4 or IPv6 and return it as a void pointer.

int main(int argc, char *argv[]);

It takes parameter the host name (server address) of the client, in the implementation that IP will not be used as it will use the IP of the current device of the client, this occurs with the line "hints.ai_flags = AI_PASSIVE;". Then the client starts connecting with the server and then prints "client: connecting to x" where x is the address of the server. After that, the function "readcommands()" is called which also calls and handles calling the other functions to send requests or data as shown above.

Server Side

Functions and Data Structures:

No major data structures are used, any data structure is only locally used in a function and will be described below in its specific function.

void get_server(char* data, int sockfd);

It takes two parameters: a pointer to the char array of data to send and the socket number to send that data respectively.
This function is called whenever there is a message of size less than or equal to MAXDATASIZE required to be sent from server to client (responses to GET requests from clients). Generally the whole data to be sent is divided first to chunks less than or equal to that MAXDATASIZE and then each chunk is sent on the corresponding required socket using this function. Thus this function can be called more than one time for large data.After finishing sending the whole data, an end of data specified by "\r\n" is sent meaning the end of data and so the next message will be sent to the server will be a new POST or GET request (accept and handle new requests from the client).

void read_file(string path, int sockfd);

This function is used in GET responses to the client to deliver data from the server to the client to send data to the server from the client. It divides the data into chunks of maximum size equals to MAXDATASIZE and then calls the "get_server" function to send each chunk. It takes a parameter the file name with its extension, checks whether it is a text file or html file or an image. The other parameter is the socket number to send data to. The file o be sent is read character by character or byte by byte till the "MAXDATASIZE - 2" in order to manually write the end of buffer "\0" to end it. For th las chunk, the buffer "buf" used to store and sent data is freed.

int send_image(int socket);

This function is called whenever an image is to be sent from the server to the client. The picture file is opened, if it is not found at the server, then it sends (replies with a response) with "404 not found", else, the picture is opened, read byte by byte till the MAXDATASIZE into a buffer "send_buffer", this buffer then is sent by reference to another function "get_server" in order to start sending it to the client as described in the "get_server" function. After that, the picture file is closed.

vector<string> split(char *str,string sep);

It takes two parameters: a pointer to a string (str) to split on the second parameter (sep) to split the string on it. It returns a vector containing the result of slitting. This function uses "strtok"

which keeps on splitting the string whenever finding the separating substring to split on. It implementation internally automatically advances a pointer to that. This function will be used mainly in reading the commands file.

void *get_in_addr(struct sockaddr *sa);
It checks the the socket address to whether use IPv4 or IPv6 and return it as a void pointer.

int main(void);
It is the main function of the server. Also uses the IP of the current device using “hints.ai_flags = AI_PASSIVE;”. Then it starts getting connections from clients and their information. For each accepted connection a new thread is created with a specified socket number for that connection. In the children, they do not need the main listener so close it. Use the “select()” function to select data from a connection. If data was received, then check the number of bytes and get that data into a buffer “buf”. If the last 4 characters in the buffer were “\r\n” then that is an end of sent data. After that, check if that data was request or data from client using POST request. If it is a GET request from the client, then handle i and send data to the client, else if it is POST request, send “ok 200” acknowledgment and start receiving and writing data from client message to the buffer then to a file if needed. Otherwise, if that was completion of data for a POST request, then mark this as data and continue reading data (requests or chunks of data if large data) into that same file for the whole data, till “\r\n”, where handling new requests will take place.

Threads or processes !

The implementation uses threads in order to use a shared variable for the timeout between all of the threads, each thread has a connection. Threads will work in parallel for handling each connection.

Handling timeout:

Using the threads, timeout is implemented in a dynamic way not static, it waits for specific amount of time, counted from the number of connections currently in the system, after that it considers the connection as time out without sending any data or requests. That number of connections is counted on creating a new thread with a new connection, also each time a connection is closed and a thread terminates. Lock are used to update that shared variable (count of current connections) among all the threads.

Handling persistence:

Persistence is that all the requests and responses between client and server are in only one connection, thus the connection will not close or terminate after only one request, but it will continue be open waiting for other requests on that same connection till the client closes it.