سيف الدين إيهاب مصطفي إبراهيم 32

REPORT

Overall Description

Server side:

The program starts with creating a mutex lock to prevent any deadlocks or wrong values during the multithreading then declaring variables number of current clients, socket of the server, socket of the current client, address of the server, address of the client and address size.

- After taking the input from the client first thing the program searches for the port number if it is not specified in the input then the program uses port number 80 as default number.
- It continues by declaring a socket for the server which acts as a file descriptor but among the connection.
- Then it assigns some important information like memory space for the server address, socket family and socket port.
- After that it associates the socket with a port on the local machine through the bind function.
- Server socket starts listening for any client requests with max pending clients as 50.
- After that the program finally enters the infinite loop of processing requests until the connection is over.
- The server then accepts the first client storing its socket and address and start handling the client after increasing the number of current clients in locked instruction.
- The client is being handled by creating a new thread with the specified function and the client socket as its parameter by detaching the thread execution from the object independently.
- The client handling starts by creating a buffer to contain the request and entering a loop until the client is done.
- In the loop the keep alive interval which is 30 mins by default is distributed among every current client then assigning that time to the socket for its execution.
- Then the program read the incoming data from the client using the recv function once data exists without any errors the server starts executing them preparing to send the response back to the client.
- Until the client finishes all its requests the server start executing by finding the true length of the total buffer and whether it needs to be cleaned or not.
- The server then starts processing the request by finding the first token and specifying whether it is a GET or POST request and finding the path of the requested file in the request.

- If the request is GET request then the server starts by finding the file to verify that it exists and storing its size.
- If the file doesn't exist, then the server sends its response to the client as status code 404 not found.
- If the file exists, then the server prepares its response with status code 200 OK and continues by finding the type of the file whether it is HTML, an image or just a text file.
- After that it extracts the exact file name from the path of the file then adds the appropriate header to the response adding the file type, file length and file name.
- Then it starts to open the file and adding the data involved to the response.
- At the end the server sends back to the client socket its response full of all the data alongside with the size of the response.
- Now if the request was a POST request, then the server starts by finding the data associated with the content disposition header, separating it by space, taking the third token that has the path and storing the filename.
- After that it stores the data sent in the POST request then finds the type header, separate it with space then storing the data type whether it is an HTML file, an image or just plain text.
- Same with the length header and storing the data size.
- After that the server opens the file and starts to write the data received.
- Once the client is done the program removes it from the current clients and then close the connection to the client socket.

Client Side:

The client side uses many of the algorithms used in the client side except they are in a different order and data processing is opposite to the server side.

- The client starts by figuring the port number, assigning important info then starts connecting to the server socket using the server address.
- If the request is GET or POST request then it sends the correct format to the server alongside its size.
- Then the program starts the loop until the client is finished by receiving the response from the server then extracting the important information as like the file path, file name, data involved, data type and its size.
- After the client is done it closes the connection between the 2 sockets.

At any given time if an error is discovered then the program prints the error explaining it and terminates the program immediately.

Major Functions

1. getTrueLength: it finds the end of the buffer after pressing enter twice then finds the length of the main request without the headers if the request is GET or the length of the request and the whole data associated if it is a POST request.
2. getFileType: finds the type of the file by reading the extension of the file after finding it in the path of the file associated which is located after the last dot.
3. getDataType: compares the data type written the headers and accordingly returns the appropriate extension to the file.
4. sendAll: keeps track of total bytes and bytes left while sending each data through the socket TCP connection if there is any error then it returns -1.

Data Structures

1. Server address and client address are represented as sockaddr_in.
2. The lock used during the multi threading is pthread_mutex_lock.
3. The time data struct that keeps track of the time passed and distribute them among the current clients through the setsockopt function is of type struct timeval.
4. Finding the file and its size using std::experimental::filesystem.
5. Opening the file though std::ifstream.
6. Iterator used for separating strings is of type std::istreambuf_iterator.
7. Tokens from the iterator are saved as vector<string>.
8. Request sent by the client is of type pair<string, string>.