GIT Department of Computer Engineering

CSE 344 - Spring 2023

Final Report

Mehmet Mete Şamlıoğlu

200104004093

SERVER SIDE

In server side my main goal is to set up the server for listening the client connections. I create a new thread for each client and handle the requests concurrently. I finish it when all client threads are finished and executed. I will explain my code detailly now.

I start by checking if the argc commands are correct and provided rightfully. If the user did not run by a port number, I print an error message and terminate by exit.

I declare a variable for holding server database bay pointer on server files. The number of elements are decided by MAX_SERVER_FILES.

I initialize a condition variable named serverSignal to be used in pthread_cond_init function. I do this to use for synchronization between threads. Then I also initialize a semaphore (semEmpty) using sem_init for using in access control to a shared resource and it is initially 0.

I create a socket and hold the servers address information. I call bind function that binds the socket to the servers address. I listen for incoming client connections and wait for a client to connect by calling accept function. The accept function blocks until a client establishes a connection. When a connection is made, a new socket descriptor (newsockfd) is created for communication with that particular client. The client's address information is stored in the cli_addr structure.

I create new threads that executes client function. If the thread is successfully created, I increase the num_of_threads. The main thread waits for the client finish execution by pthread_join. If the thread is joined, num_of_threads decrements. I close the sockets and free necessary serverfiles and destroy the semaphores and finish the program.

Now we have explained the main function, I will explain the other helper functions for server side.

Client function:

The client function handles the communication between the server and a specific client. It sends the list of server files to the client, receives files from the client, and enters an infinite loop to receive commands from the client and perform corresponding actions.

I initialize the necessary variables and arrays for storing commands and messages and the names of server files. I call updateserverfiles function to get the list of server files and update them. I copy the file names to send_files in a loop and send the files to the client using newsockfd socket. The function receives a signal from the client indicating that it has received the server file names. If the received signal is "done," the function calls receiveClientFiles to receive files from the client and then sends the "done" signal back to the client. If the received signal is not "done," the function returns without further processing. Then it enters an infinite loop to handle command coming from the client. The server receives a command from the client using the recv function, which is stored in the command array. The loop repeats, and the process continues until the thread is terminated. The memory allocated for the args parameter is freed using the free function.

The receiveClientFiles function receives the number of files from the client, receives the file names, stores them in a 2D array, adds the received files to the server's file storage, and updates the server's file information.

The addToServer function receives data from the client, writes it to the destination file in the server's file storage, handles termination and end-of-file signals, and keeps track of the total number of bytes read.

The sendFileToClient function sends the number of files and their names to the client. It then iterates through the file names array to send each file name individually. After that, it calls a helper function to send the file contents. Finally, it returns the success status of the file transfer.

The sendFileToClientHelper function is a helper function that reads each file from the server-side, sends its contents to the client, and signals the end of each file transfer.

The updateServerFiles function scans the current directory, skips unwanted files/directories, and stores the names of valid files in the ServerFiles array. It returns a success status indicating whether the update was performed successfully.

CLIENT SIDE

In client side, I start by declaring variables and initializing necessary structures, also allocate memory for the clientFiles array and initialize a semaphore.

Next, it checks the command-line arguments to ensure that the hostname and port are provided. It then creates a socket using socket and retrieves the server's information using gethostbyname.

I set up the server address and connect to the server using connect. I also initialize and clear the buffer.

After establishing the connection, the client updates its list of client files and receives the list of server files from the server.

It sends a "done" signal to the server indicating that the file exchange is complete, and then proceeds to send files to the server using sendFilesToServer.

The main function receives a response from the server and enters a loop to handle commands from the server. It receives commands, checks if there are any edits on the client side using isClientEdited, and sends appropriate responses to the server.

The loop continues with a delay of 3 seconds between iterations so we would know if there is any changes to the folders and files. Finally, the main function closes the socket, destroys the semaphore, and frees the allocated memory.

The function sendFilesToServer is responsible for sending files from the client to the server.

The function starts by declaring variables and initializing them. It also initializes a send_bytes variable to track the number of bytes sent.

It sends the number_of_files variable to the server using send to indicate how many files will be delivered. If the send_bytes value is less than 0, an error message is printed indicating a failure to send the signal. Otherwise, the function enters a loop to send each file to the server. It uses send to transmit the file name stored in the files array to the server. A message indicating the file name is printed for reference. After each file is sent, there is a small delay using usleep to avoid overwhelming the server.

Once all files are sent, the function calls sendFileToServerHelper to handle the actual file transfer. The return value of this helper function is assigned to isSuccesfull. Finally, the function returns the value of isSuccesfull, which indicates the success status of the file transfer.

sendFileToServerHelper handles the actual file transfer from the client to the server.

I start by initializing variables such as isSent and counter to track the transfer progress, and fd_client to store the file descriptor of the client-side file. Inside a loop, the function opens the client-side file using open.

Within another loop, the function reads data from the client-side file using read into the buffer. The read data is then sent to the server using send.

I keep track of the total number of bytes read from the file in the num_bytes_read variable. After each file is read and sent, the function increments the counter, closes the client-side file, and resets the buffer.

A small delay is introduced using usleep to avoid overwhelming the server. If there are more files to send, the function sends an "ENDFILE_SIGNAL_DETECTED_0x24367131" message to the server using send. If all files have been sent, a "TERMINATE_SIGNAL_DETECTED_0x24367131" message is sent instead.

Finally I print the number of bytes written (num_bytes_read) and return the value of isSent, indicating the success status of the file transfer.

receiveServerFiles handles the process of receiving file information from the server.

Using recv, the function receives the value of number_of_server_files from the server, which represents the total number of files to be received. An array called files stores the names of the server files.

Inside a loop, the function uses recv to receive the file_name from the server. The received file name is then copied into the corresponding row of the files array using strcpy. After all file names have been received and stored in the files array, the function calls addToClient to handle the process of adding the received files to the client's file system.

I call updateClientFiles to update the client's file information. Finally, a success message is printed for that the server's data has been copied successfully.

The function addToClient is responsible for receiving files from the server and saving them on the client-side. It iteratively receives data from the server, handles specific signals, writes file content to the appropriate destination file, keeps track of the number of bytes read.

The function updateClientFiles is responsible for updating the list of files on the client-side. It scans the current directory, skips certain files and directories, and stores the names of the remaining files in the clientFiles array, updating the file_size.

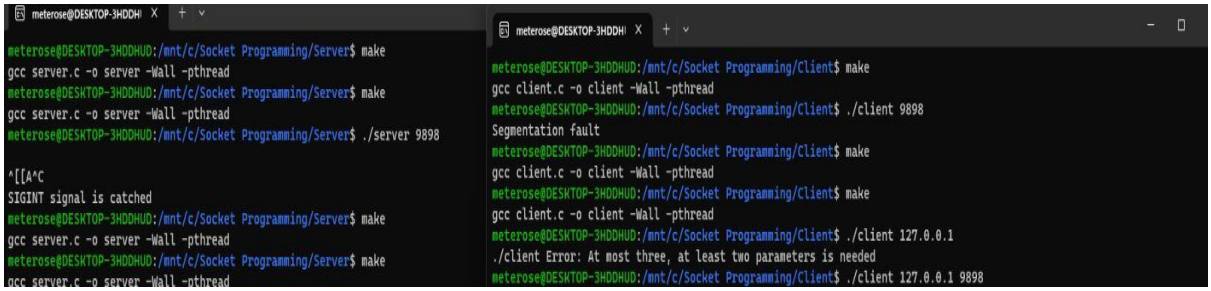
The function isClientEdited determines whether the client's files have been edited. After processing all directory entries, the function compares number_of_files_directory with file_size (which represents the number of files on the client). If they are equal, it means no changes have been made, and value_return is set to 0.

If number_of_files_directory is greater than file_size, it indicates that a new file has been added, and value_return is set to 1.

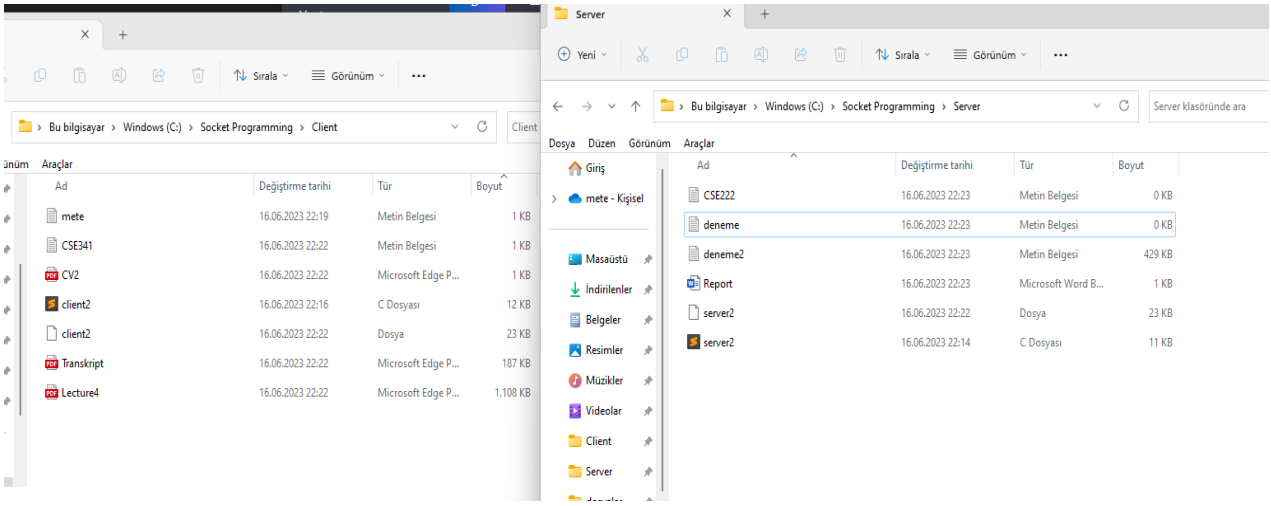
If number_of_files_directory is less than file_size, it suggests that files have been removed, and value_return is set to -1.

Finally, the function returns value_return, which represents the state of client file modifications: 0 for no changes, 1 for a new file added, and -1 for files removed.

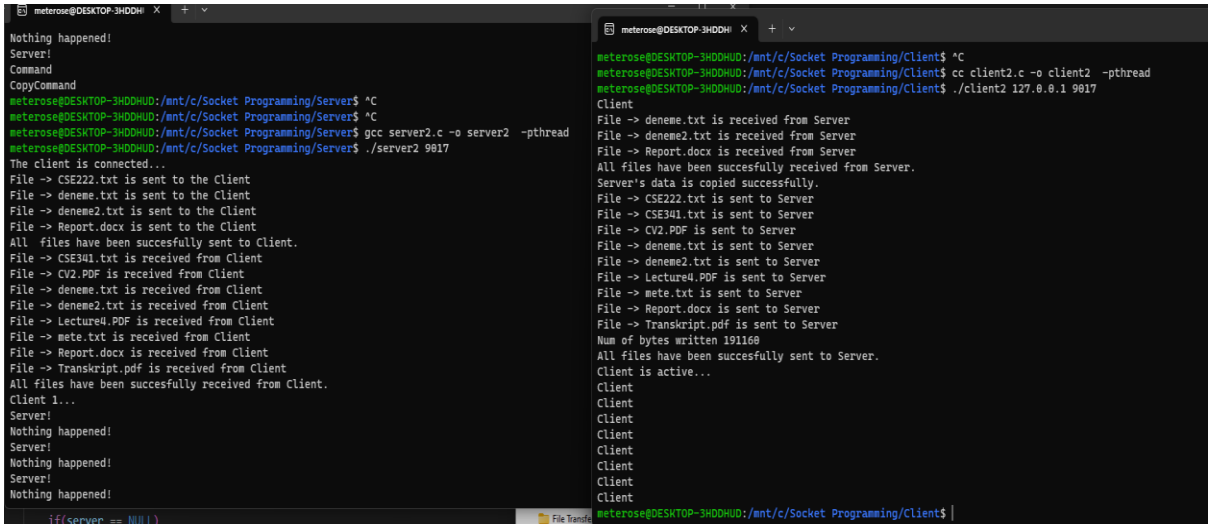
TESTS AND RESULTS



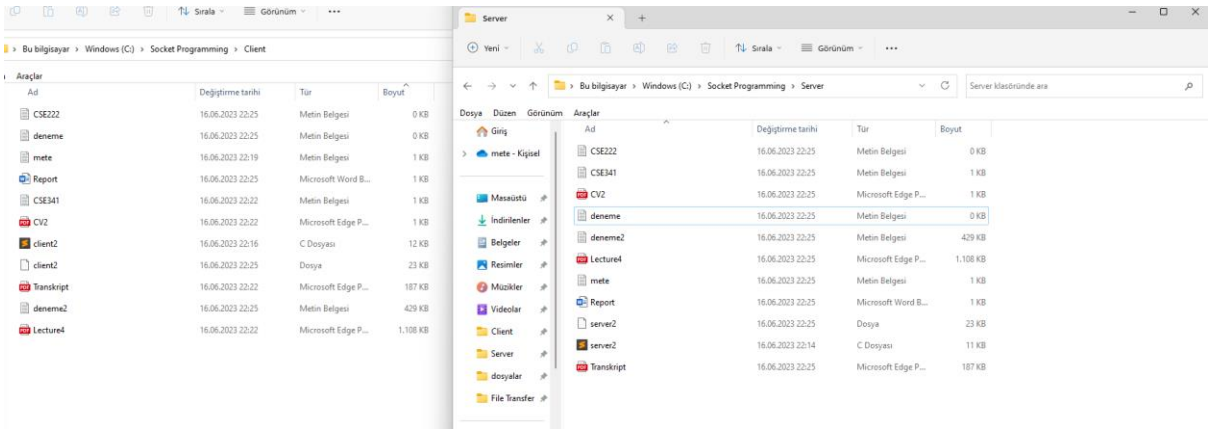
This is how to run my program.



The directory contents are different at first.



We run the program and connect the client. Then it starts to match the directory contents.



Now the contents are the same.

```
Server!
Nothing happened!
^C
SIGINT signal is catched
Server!
Failed to send command
: Bad file descriptor
meterose@DESKTOP-3HDDHUD:/mnt/c/Socket Programming/Server$
```

The SIGINT works when called.