CSE 344 Final Report
Mehmet Acar
1801042095

server.c

Data Structures:

- `File`: Structure to hold information about a file, including its name and content.
- `Directory`: Structure to represent a directory, containing its name, files, subdirectories, and their counts.
- `ClientInfo`: Structure to hold information about a client, including its socket and address.
- `file`: Structure to store information about a file, including its name, mode, content, size, modified time, and deletion status.
- `FileEntry`: Structure to store information about a file entry, including its name and content.
- `DirEntry`: Structure to represent a directory entry, containing its name, files, subdirectories, and their counts.
- `Info`: Structure to hold information about a file or directory, including its name, deletion status, type, content, and modified time.
- `client_t`: Structure to store client information, including its address, socket, UID, and name.

Global Variables:

- `pathName`: A character array to store the path name.
- `threadPoolSize`: An integer to store the size of the thread pool.
- `threadPool`: An array of pthread_t to hold thread IDs.
- `server_socket`: Socket descriptor for the server.
- `done`: A flag to indicate if the program execution is completed.
- `clientInfo`: An array of Info structures to store client-side information.
- `serverInfo`: An array of Info structures to store server-side information.
- `inf` and `inf2`: Arrays of Info structures.
- `count`, `count2`, `counter`: Counters used in the program.
- `f` and `f2`: Arrays of file structures.
- `MAX_CLIENTS` and `BUFFER_SZ`: Constants for maximum clients and buffer size.
- `cli_count`: An atomic unsigned integer to store the client count.
- `uid`: An integer to store the user ID for clients.
- `clients`: An array of client_t pointers to store client information.
- `clients_mutex`: Mutex for client-related operations.

1. It defines constants and macros such as `BUFFER_SIZE`, `MAX_FILE_CONTENT`, `MAX_CONTENT_LENGTH`, and `MAX_PATH_LENGTH` to specify buffer sizes and limits for file paths and contents.
2. It declares and initializes several global variables, including mutexes (`pthread_mutex_t`), structures (`File`, `Directory`, `ClientInfo`, `file`, `FileEntry`, `DirEntry`, `Info`), and arrays (`pathName`, `clientInfo`, `serverInfo`, `inf`, `inf2`, `f`, `f2`).
3. It includes a thread pool implementation using `pthread_t` and `pthread_mutex_t` to manage multiple client connections.
4. The code defines a function `readFile2` that reads the contents of a file and stores it in the `serverInfo` structure.
5. It defines a function `copyFile` that copies a file's content to another file.
6. There is a function `updateServer` that updates the server's file information based on the client's file information.
7. The code includes a function `traverseServer` that traverses the server's directory and populates the `serverInfo` structure with file information.
8. The `send_message` and `send_message2` functions send messages or data to connected clients.
9. It defines a function `print_client_addr` that prints the IP address of a client.
10. The code includes functions related to maintaining a queue of clients, adding and removing clients from the queue, and sending messages to clients.
11. There is a `main` function that serves as the entry point of the program. It initializes variables, creates a server socket, listens for client connections, and creates threads to handle client requests.

client.c

1. It defines several constants, such as `BUFFER_SIZE`, `MAX_PATH_LENGTH`, `MAX_FILE_CONTENT`, and `MAX_CONTENT_LENGTH`.
2. It defines multiple structures, including `File`, `Directory`, `file`, `FileEntry`, `DirEntry`, and `Info`. These structures represent different entities related to files, directories, and synchronization information.
3. It initializes several mutex variables using `pthread_mutex_t`, which are used for thread synchronization.
4. The code defines global variables and a signal handler function `catch_ctrl_c_and_exit()`.

5. The `writeLogTime()` function is defined to write log timestamps to a log file.
6. The `writeLog()` function is defined to write log messages to a log file.
7. The `readFile2()` function reads the content of a file specified by the `fileName` parameter and stores it in the `clientInfo` array.
8. The `copyFile()` function is used to copy a file specified by the `info` parameter.
9. The `updateClient()` function updates the client's file information based on the server's information.
10. The `traverseClient()` function traverses the client's directory and populates the `clientInfo` array with file information.
11. The `send_msg_handler()` function continuously sends the `clientInfo` array to the server.
12. The `recv_msg_handler()` function continuously receives the `serverInfo` array from the server.
13. The `main()` function initializes the client socket, sets up the server address and port, and establishes a connection with the server. It also creates two threads: one for sending messages and one for receiving messages.
14. The main() function starts by setting up a client-server connection using sockets. The client connects to the server using the provided server IP address and port number. The server IP address and port number are passed as command-line arguments to the program. The server listens for incoming client connections on the specified port.
15. The main function of the program consists of two threads: one for sending messages from the client to the server and another for receiving messages from the server. The send thread continuously reads file and directory information from the client's local directory, stores it in the clientInfo structure, and sends it to the server. The receive thread receives file and directory information from the server, updates the client's local directory based on the received information, and logs any changes made.

Which Requirements I met?

1-Any new file created on the server should reflect the same changes on the client side,and vice versa.

2-Maintain a logfile under the respective client's directory.

3-Handle SIGINT signal on both the server and client sides.

4-Server prompt a message when a client connection is accepted (with the address of connection) to the screen.

Test Cases

1- Connecting to the server from client on same machine

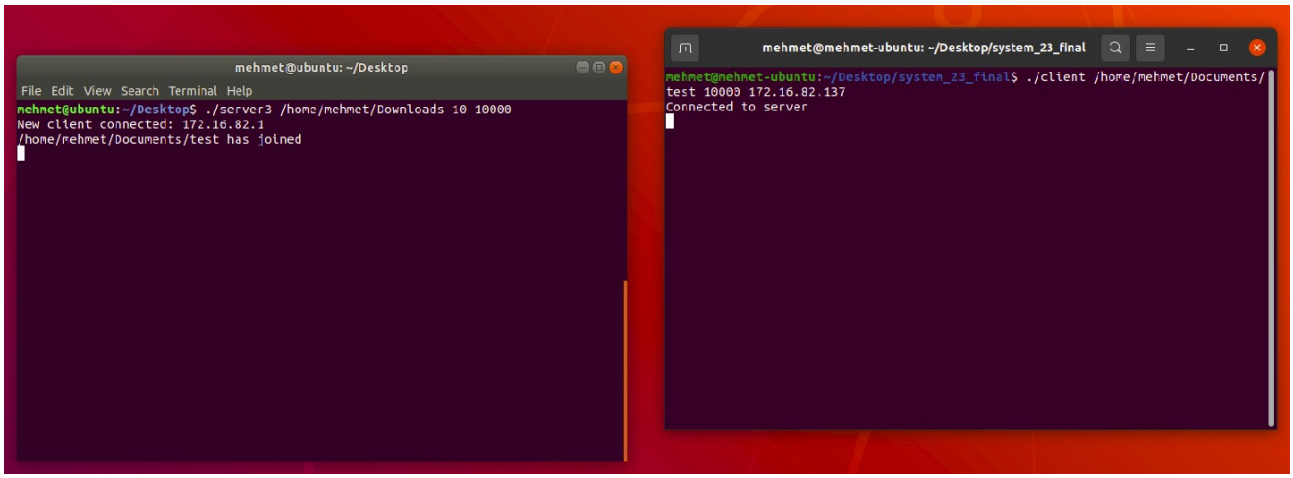
client Terminal

```
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$ ./client /home/mehmet/Documents/
test 10000
Connected to server
```

server Terminal

```
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$ ./server /home/mehmet/Documents/
deneme 10 10000
New client connected: 127.0.0.1
/home/mehmet/Documents/test has joined
```

2- Connecting to the server from client on different machines



2- Enter Ctrl C from client

Client Terminal

```
^C
Bye
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$
```

Server Terminal

```
/home/mehmet/Documents/test has left
```

3-Enter Ctrl C from server

Server Terminal

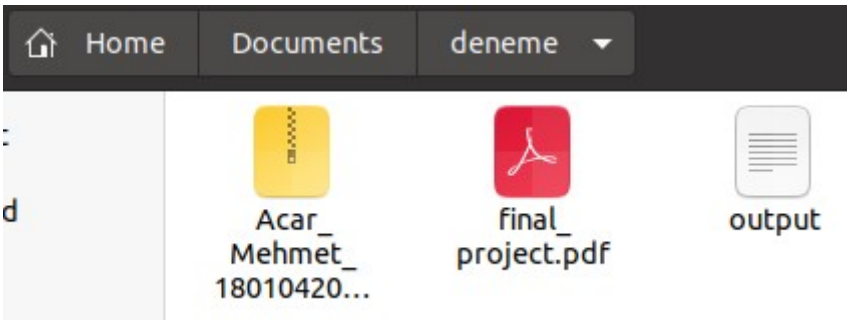
```
^C
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$
```

Client Terminal

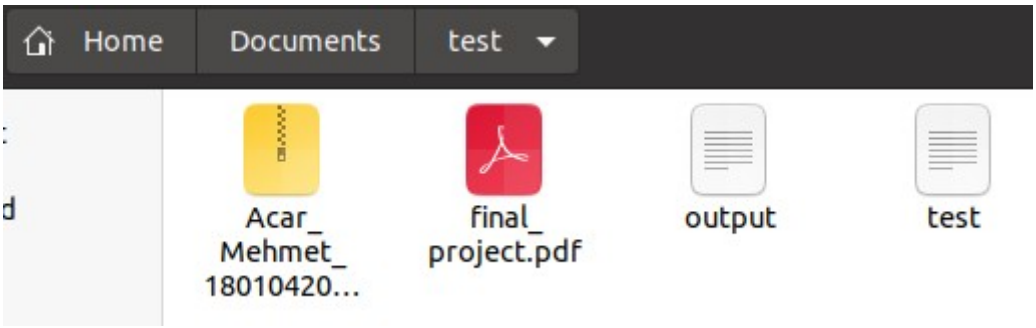
```
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$ ./client /home/mehmet/Documents/
test2 10000
Connected to server
mehmet@mehmet-ubuntu:~/Desktop/system_23_final$
```

4- Checking server and one client path files at the end of the program in order to see two path is same except client log file

Server Path



Client Path



In the client path, test is a log file and except this file, other files are same with server path.