CSE 344 SYSTEM PROGRAMMING MIDTERM PROJECT

Melih Sapmaz

220104004130

Description Of The Project

The system architecture is based on a client-server model. The server is a multi-threaded application that can handle multiple clients concurrently. The server listens for connections and creates new threads for each client that connects. This allows the server to handle multiple clients simultaneously.

The server uses a socket communication mechanism to interact with the clients. The server creates a socket, binds it to a specific port (using the server's PID as the port number), and then listens for incoming connections on that socket. When a client connects, the server accepts the connection and creates a new socket for communication with that client.

The server uses the “pthreads” library for multi-threading. Each client is handled in a separate thread, allowing the server to handle multiple clients concurrently. The server maintains an array of client sockets, an array of client PIDs, and an array of threads, one for each client.

The server uses a mutex (`file_mutex`) to synchronize access to files. This is necessary because multiple threads could try to read from or write to the same file at the same time, leading to race conditions. The mutex ensures that only one thread can access a file at a time.

The server supports several commands from clients, including listing the contents of the server's directory, reading from a file, writing to a file, uploading a file, downloading a file, archiving files, shutting down the server, and terminating the client.

The server uses the `fopen`, `fread`, `fwrite`, and `fclose` functions for file I/O. The `read_text_file` and `write_text_file` functions are used for reading from and writing to text files, while the `read_binary_file` and `write_binary_file` functions are used for reading from and writing to binary files.

The server uses the `fork` and `execvp` functions to create a child process and execute the `tar` command to archive files. The server waits for the child process to finish before continuing.

The server handles SIGINT signals (which are sent when the user presses Ctrl+C) by shutting down the server, closing all client sockets, sending a kill signal to all clients, and canceling all threads.

The server code is organized into two files: `Server.c`, which contains the main server logic, and `file_operations.c`, which contains the file I/O functions. The `file_operations.h` header file contains the function prototypes for the file I/O functions.

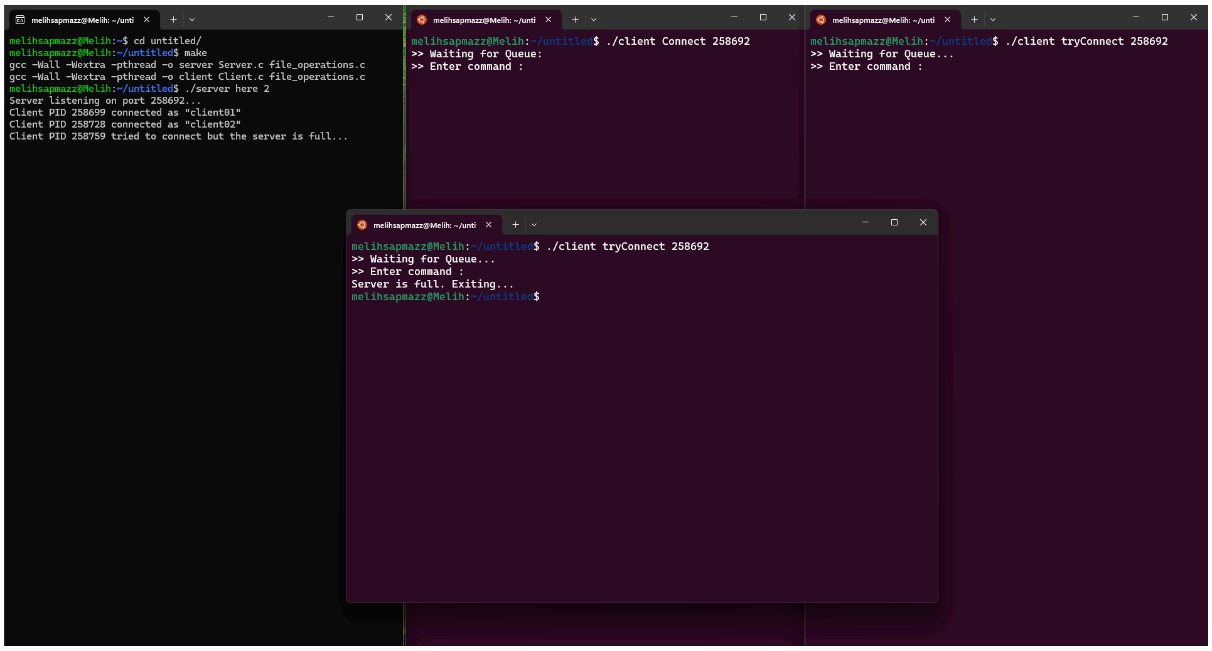
The `Client.c` file is a client-side application that communicates with a server over a network using sockets. The client can send various commands to the server, such as listing the server's files, reading a file, writing to a file, uploading a file, downloading a file, archiving server files, and terminating the server.

The client application uses a command-line interface. The user enters commands, which are then parsed and handled by the appropriate function. The commands and their corresponding handler functions are stored in an array of `Command` structures. The `command_handler` function is used to find the appropriate handler for a given command.

The client application supports two modes of operation: `Connect` and `tryConnect`. In `Connect` mode, the client attempts to connect to the server and waits if the server is full. In `tryConnect` mode, the client attempts to connect to the server and exits if the server is full.

The client application uses the `send` and `recv` functions to send commands to the server and receive responses. The `send_command` function is used to send a command to the server.

Example Outputs From the Program

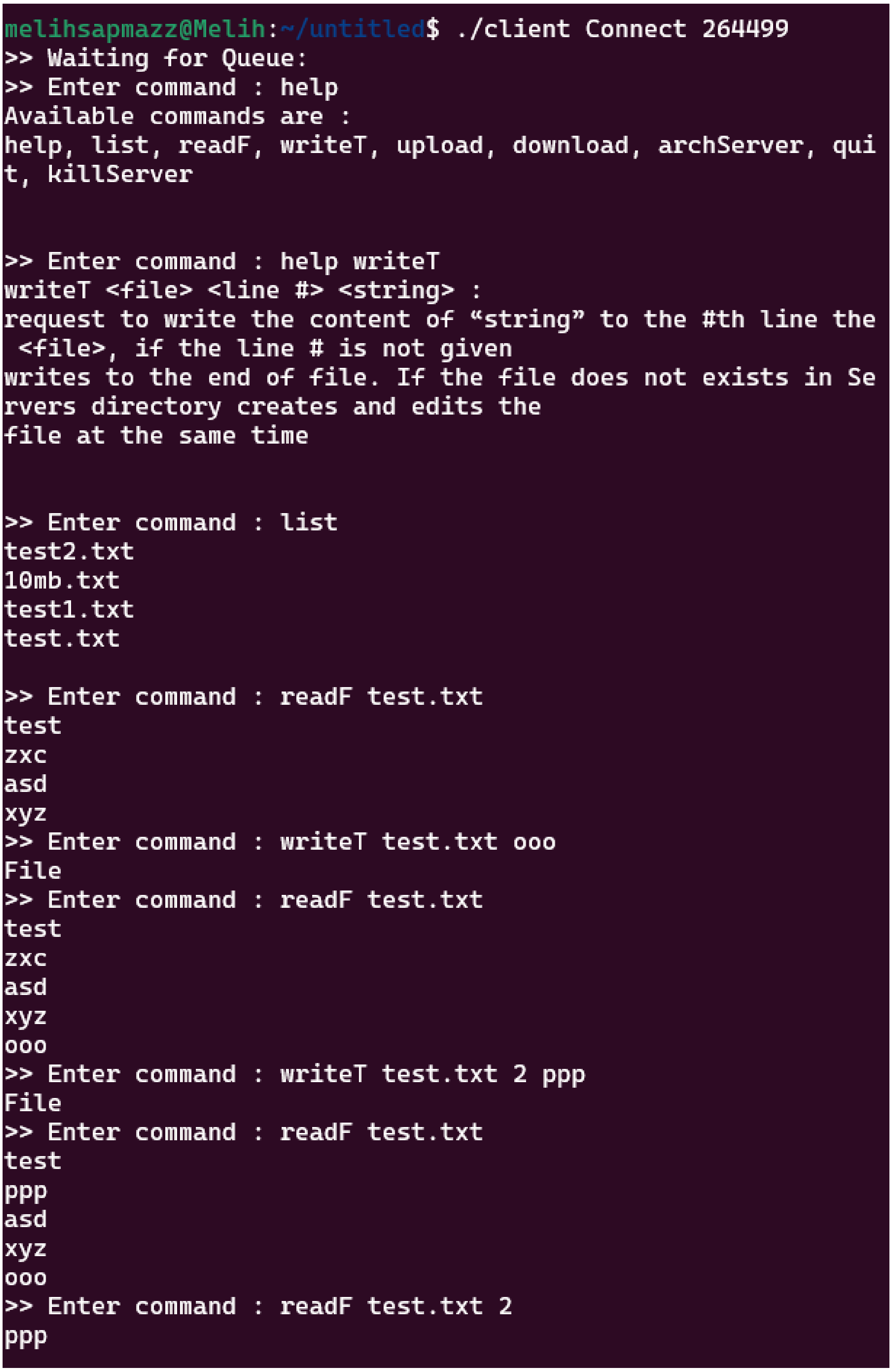

melihsapmazz@Melih: ~/unti
melihsapmazz@Melih: ~/unti
melihsapmazz@Melih: ~/unti
melihsapmazz@Melih: ~/unti

```
melihsapmazz@Melih:~$ cd untitled/
melihsapmazz@Melih:~/untitled$ make
gcc -Wall -Wextra -pthread -o server Server.c file_operations.c
gcc -Wall -Wextra -pthread -o client Client.c file_operations.c
melihsapmazz@Melih:~/untitled$ ./server here 2
Server listening on port 258692...
Client PID 258699 connected as "client01"
Client PID 258728 connected as "client02"
Client PID 258759 tried to connect but the server is full...
```

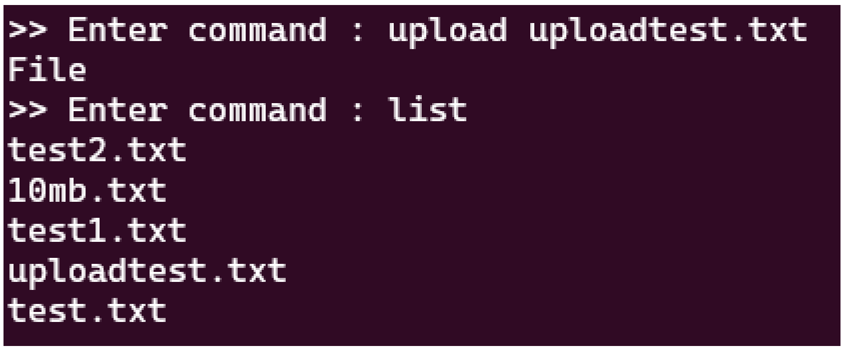
```
melihsapmazz@Melih:~/untitled$ ./client Connect 258692
>> Waiting for Queue:
>> Enter command :
```

```
melihsapmazz@Melih:~/untitled$ ./client tryConnect 258692
>> Waiting for Queue...
>> Enter command :
```

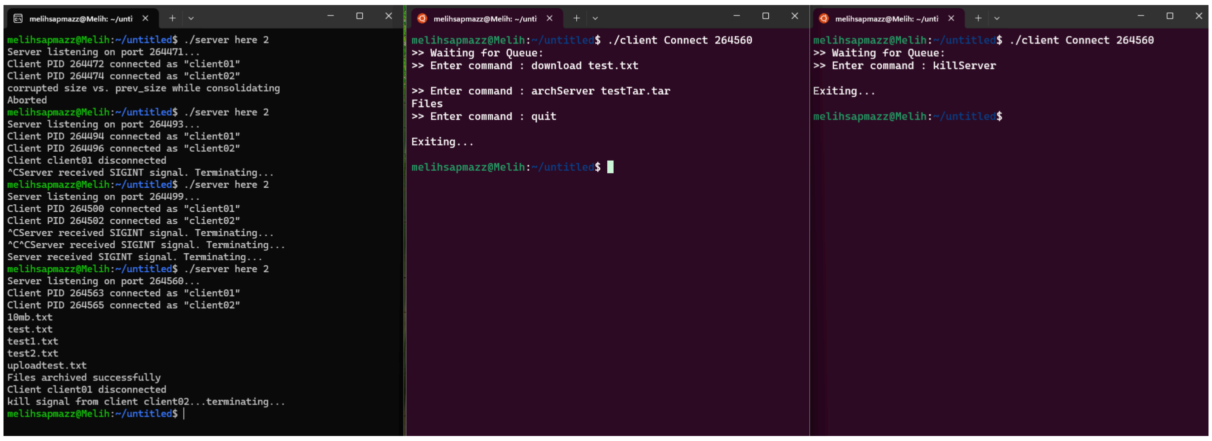
```
melihsapmazz@Melih:~/untitled$ ./client tryConnect 258692
>> Waiting for Queue...
>> Enter command :
Server is full. Exiting...
melihsapmazz@Melih:~/untitled$
```



```
melihsapmazz@Melih:~/untitled$ ./client Connect 264499
>> Waiting for Queue:
>> Enter command : help
Available commands are :
help, list, readF, writeT, upload, download, archServer, qui
t, killServer


>> Enter command : help writeT
writeT <file> <line #> <string> :
request to write the content of “string” to the #th line the
 <file>, if the line # is not given
writes to the end of file. If the file does not exists in Se
rvers directory creates and edits the
file at the same time


>> Enter command : list
test2.txt
10mb.txt
test1.txt
test.txt

>> Enter command : readF test.txt
test
zxc
asd
xyz
>> Enter command : writeT test.txt ooo
File
>> Enter command : readF test.txt
test
zxc
asd
xyz
ooo
>> Enter command : writeT test.txt 2 ppp
File
>> Enter command : readF test.txt
test
ppp
asd
xyz
ooo
>> Enter command : readF test.txt 2
ppp
```



```
>> Enter command : upload uploadtest.txt
File
>> Enter command : list
test2.txt
10mb.txt
test1.txt
uploadtest.txt
test.txt
```



melihsapmazz@Melih: ~/unti
melihsapmazz@Melih: ~/unti
melihsapmazz@Melih: ~/unti

```
melihsapmazz@Melih:~/untitled$ ./server here 2
Server listening on port 264471...
Client PID 264472 connected as "client01"
Client PID 264474 connected as "client02"
corrupted size vs. prev_size while consolidating
Aborted
melihsapmazz@Melih:~/untitled$ ./server here 2
Server listening on port 264493...
Client PID 264494 connected as "client01"
Client PID 264496 connected as "client02"
Client client01 disconnected
^CServer received SIGINT signal. Terminating...
melihsapmazz@Melih:~/untitled$ ./server here 2
Server listening on port 264499...
Client PID 264500 connected as "client01"
Client PID 264502 connected as "client02"
^CServer received SIGINT signal. Terminating...
^C^CServer received SIGINT signal. Terminating...
Server received SIGINT signal. Terminating...
melihsapmazz@Melih:~/untitled$ ./server here 2
Server listening on port 264560...
Client PID 264563 connected as "client01"
Client PID 264565 connected as "client02"
10mb.txt
test.txt
test1.txt
test2.txt
uploadtest.txt
Files archived successfully
Client client01 disconnected
kill signal from client client02...terminating...
melihsapmazz@Melih:~/untitled$
```

```
melihsapmazz@Melih:~/untitled$ ./client Connect 264560
>> Waiting for Queue:
>> Enter command : download test.txt

>> Enter command : archServer testTar.tar
Files
>> Enter command : quit

Exiting...

melihsapmazz@Melih:~/untitled$
```

```
melihsapmazz@Melih:~/untitled$ ./client Connect 264560
>> Waiting for Queue:
>> Enter command : killServer

Exiting...

melihsapmazz@Melih:~/untitled$
```