Gebze Technical University

Departman of Computer Engineering

Spring 2023 - CSE 344

Final Project Report

GONCA EZGİ ÇAKIR

151044054

1. Solution Approach

In summary, this program is implemented with a basic server client architecture to provide a solution similar as a Dropbox. Communication between client and server provided by *sockets*. Client processes are handled by *threads* on the server side by communicating through *sockets* and the synchronization between threads are provided by *counting semaphores and mutexes.*

2. System Working Mechanism

- First of all, server creates the socket and threads according to thread pool size.
- Threads are waiting on a condition that if a client request received it is readed through socket and if there is avaliablity it is added to the queue; otherwise connection refused on the cleint side.
- After connection is accepted the **server** choose one of the idle threads in order to handle request.
- Thread gets client from queue, then starts following steps:

  - Sends the server file content to client side.

  - Waits for client difference file content.

  - After receiving difference content, sends the missing file contents to client side.

  - When client sync server sync starts.

  - Gtes the client file content from client side.

  - Compares contents and sends the difference file content.

  - After sending difference content, gets its missing file contents from client side.

  - Server syncs.

- On the client side **client** sends its request info in a struct queue, then starts following steps:

  - Gtes the server file content from client side.

  - Compares contents and sends the difference file content.

  - After sending difference content, gets its missing file contents from server side.

  - Client syncs.

  - Sends the client file content to server side.

- Waits for server difference file content.
- After receiving difference content, sends the missing file contents to server side.
- When server sync client sync starts.

- The process mainly works as for also client.
- Those loops iterates continously till a server or client receives a signal.
- This process provides adding and modifying on both server and client side file contents.

Note: In order to provide better CPU usage I have added a sleep(1) at the end of the server worker thread. It is sent in a comment line but you can remove the comment in order to see a better CPU usage.

Semaphore and Mutex Usage

- Counting semaphore *empty* is set to poolSize amount at the beginning; called *sem_wait()* each time a client connected and called *sem_post()* when a client disconnected.
- Counting semaphore *full* is set to 0 amount at the beginning; called *sem_post()* each time a client connected and called *sem_wait()* when a client disconnected.
- *Mutex queue_mutex* is locked each time before a queue is manupulated and unlocked after.
- *Mutex queue_logfile i*s locked each time writing a and unlocked after finish writing.

Thread Pooling and Connection Refusing

This program creates the thread pool at the beginning of the server and handles client's requests with those threads. When a a client quits that thread which is responsible from that client's request becomes available and can be assign to new client or wait for further client connections. If there is more client than a thread pool size those clients are rejected.

Missing Properties

- Deleting functionality doesn't work properly since the program is continously checking server and client contents; the case of deleting from a server mostly doesn't effect the cliet sides.
- Program only works for local machines, due to hardware restriction I wasn't able to check this fucntionality unfourtunately.

- Signals can be handled only from the server side, server sends signal when it receives a signal. But if a client receives a signal it may cause an exit program case.

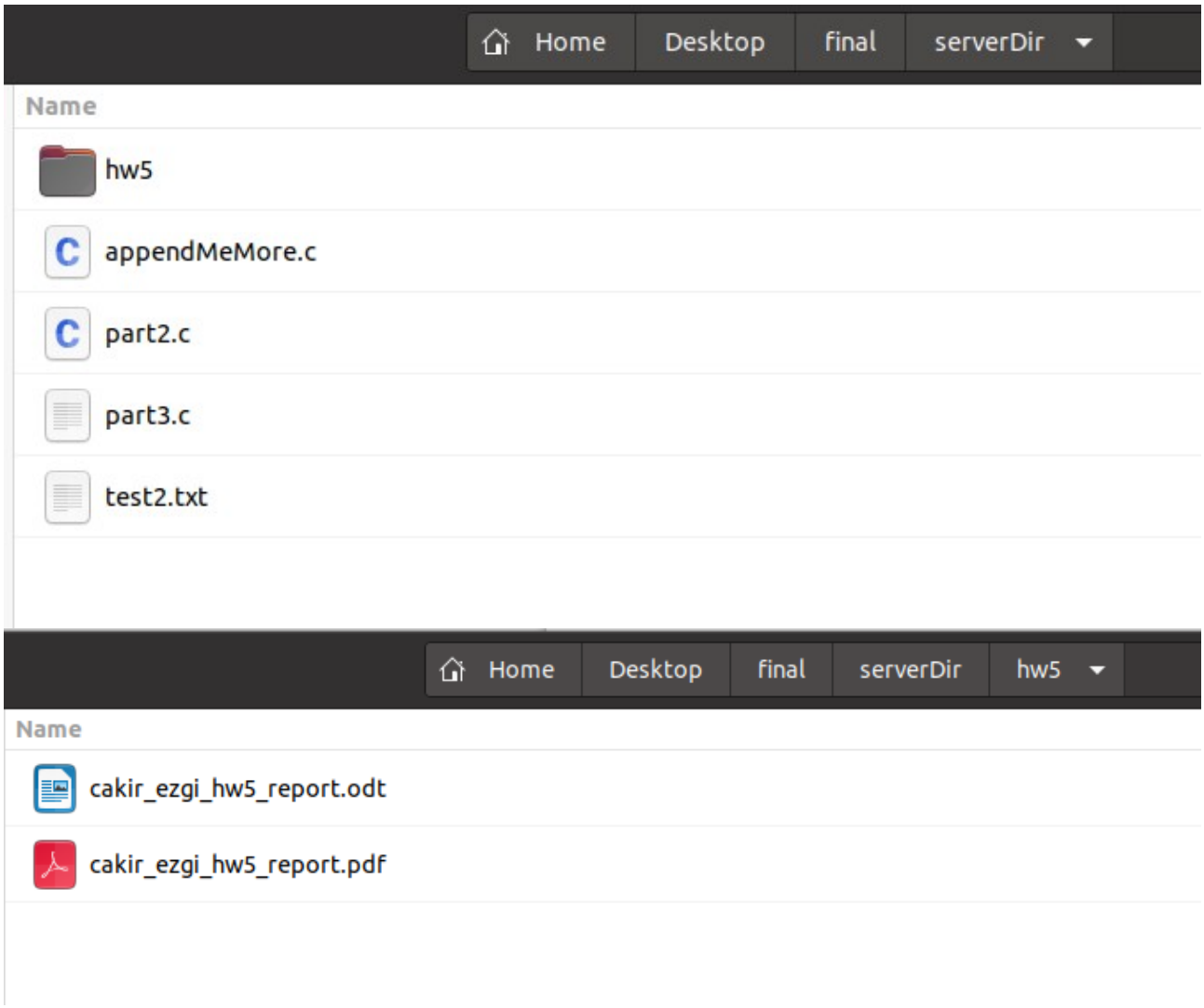
Error Handling

- If there is more or less argument than for the required server or client program prints an error message on screen and terminates.

3. Scenarios

Multiple test case senarios applied and screenshots are added. Below you will see some program input result.

Server Directory Content



Server Terminal

```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$ ./BibakBOXServer /home/ezgi/Desktop/final/serverDir 2 8094
SERVER SIDE
Thread pool is creating with size 2.
Server is waiting for clients.....

Thread0 is created.
Thread1 is created.
Client 44475 (PID) request received.
Thread0 is handling client 44475 (PID).
Client 44477 (PID) request received.
Thread1 is handling client 44477 (PID).
Client 44515 (PID) is refused, all threads are occupied.
^C
------SIGINT signal is catched-------
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$
```

Client-1 Termianal

```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$ ./BibakBOXClient /home/ezg
i/Desktop/final/test2 8094
CLIENT SIDE
Logging...

------SIGINT signal is catched------
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$
```

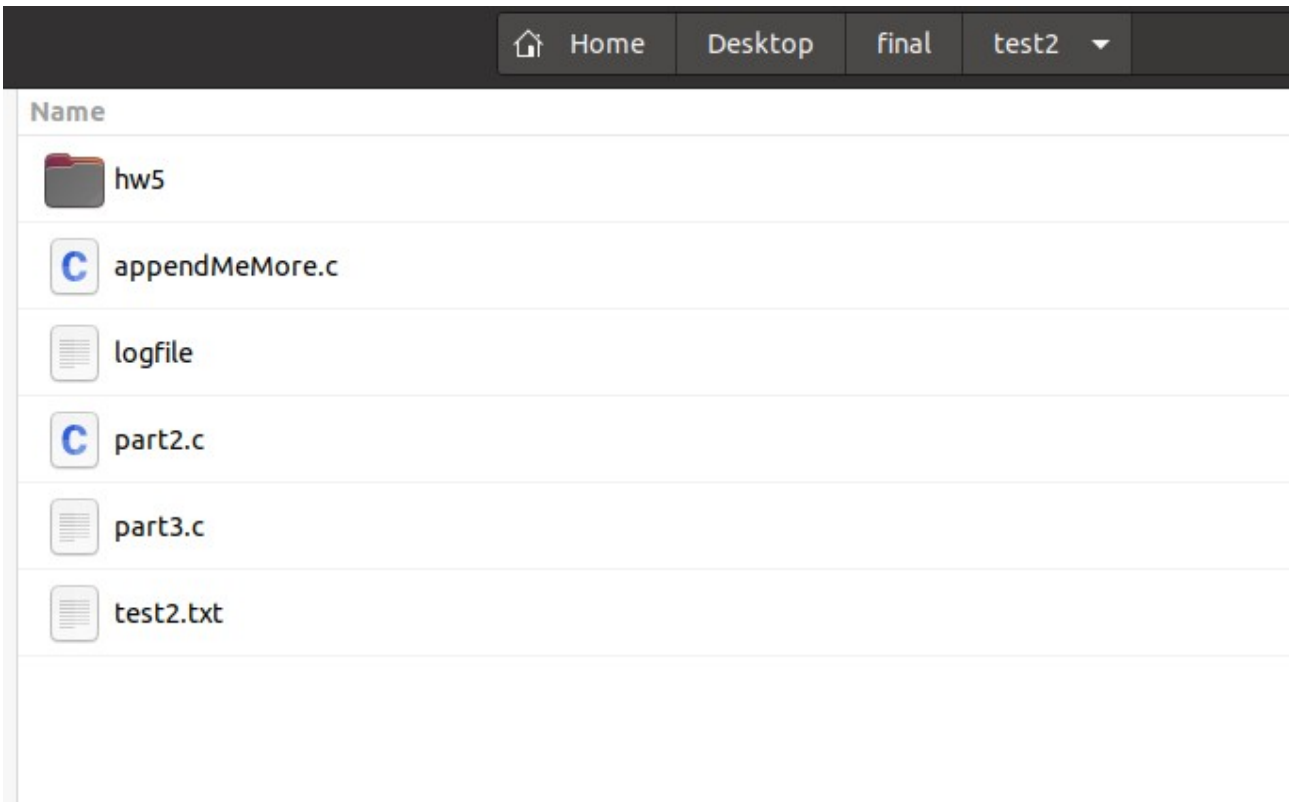
Client-2 Terminal

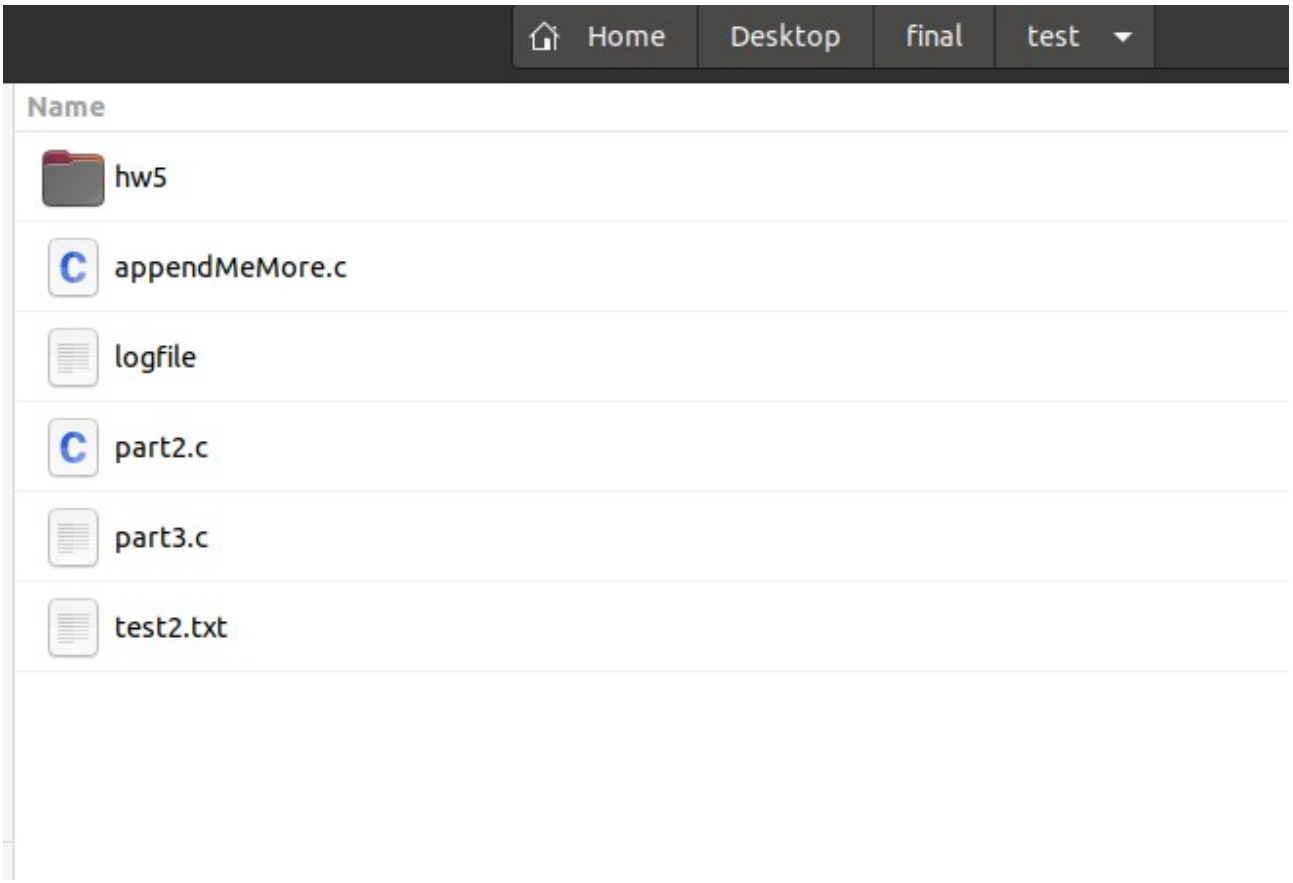
```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$ ./BibakBOXClient /home/ezg
i/Desktop/final/test2 8094
CLIENT SIDE
Logging...

------SIGINT signal is catched------
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$
```

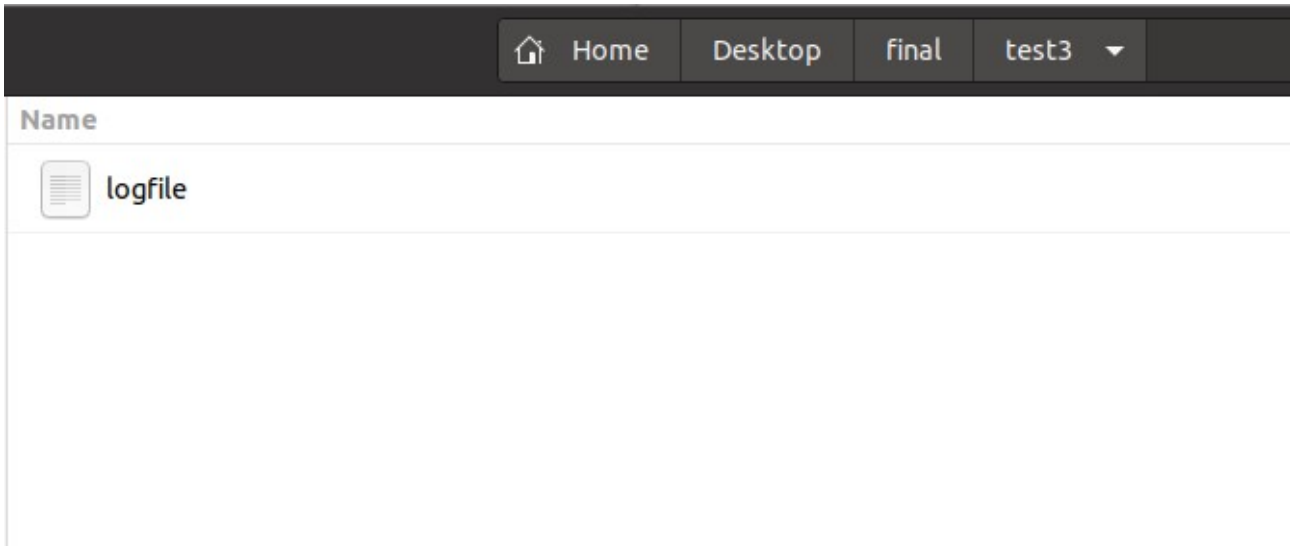
Client-3 Termianal

```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$ ./BibakBOXClient /home/ezgi/Desktop/final/test3 8094
CLIENT SIDE
Logging...
Killed
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/final$
```

Test Directory Contents After Execution







Log File Content

logfile

```
Dir hw5 is added/modified by server.
File cakir_ezgi_hw5_report.odt is added/modified by server.
File .~lock.cakir_ezgi_hw5_report.odt# is added/modified by server.
File cakir_ezgi_hw5_report.pdf is added/modified by server.
File appendMeMore.c is added/modified by server.
File part3.c is added/modified by server.
File test2.txt is added/modified by server.
File part2.c is added/modified by server.
```

Usage Information