

GEBZE TECHNICAL UNIVERSITY

Computer Engineering Department

CSE 344 – 2024 Spring

Midterm Project Report

Serdar Genç

210104004023

1) Connection:

The initial connection of a client and the server is provided by the Server FIFO which is contained inside the “server_directory”. This FIFO obeys the name pattern “server_%d.fifo” (%d = server pid).

“server_directory” is received from command line argument, but for program to work the SERVER_DIR definition in client.c must be same with server directory name.

Client program sends a **client** into the Server FIFO and the server reads it. After reading, server creates a Client FIFO with pattern “client_%d.fifo” (%d = client pid).

```
struct client {  
    pid_t pid;  
    int tryConnect;  
};
```

1.1) tryConnect vs Connect:

tryConnect is set to 1 if connection type is “tryConnect”. If tryConnect is 1 and the server is currently full, the server sends information to the client that tells server is full via Client FIFO. If the server is full, then tryConnect client terminates without waiting queue. Else it is sent to child_process function to become an active client.

If tryConnect is set to 0 this means connection type is “Connect”. If tryConnect is 0 and the server is currently full, the server enqueues the client into wait_queue which is an instance of struct queue.

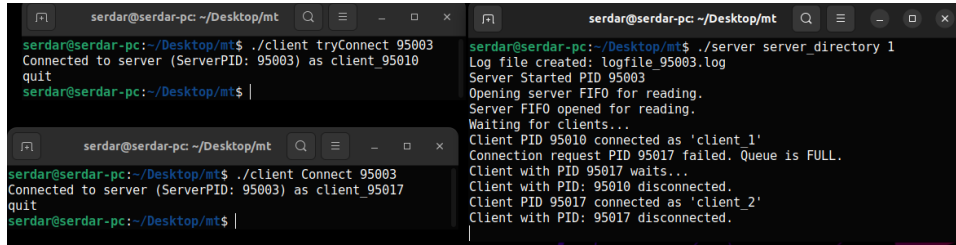
```
struct queue {  
    int front;  
    int rear;  
    int capacity;  
    int size;  
    struct client **elements;
```

```
};
```

When a client is enqueued it waits for its turn. Whenever an active client is terminated, if there is a queue, the front of the queue is dequeued and it is sent to `child_process` function to become an active client.

This is how i check this condition:

```
while (1){ if (num_clients < max_clients && wait_que.size > 0){ ..
```



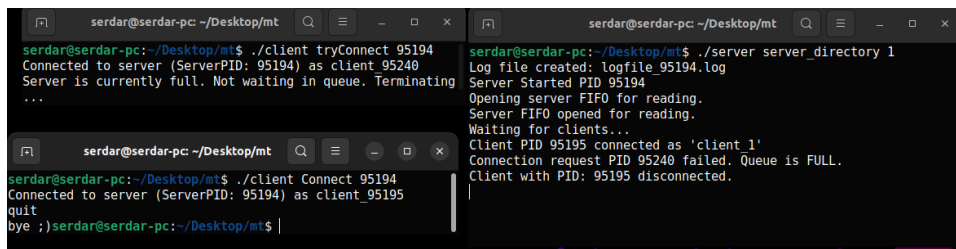
The image shows three terminal windows. The top-left window shows a client trying to connect to a server with PID 95003 and then connecting as client_95010. The bottom-left window shows a client connecting to a server with PID 95003 and then connecting as client_95017. The right window shows the server's perspective, including log file creation, server startup, and handling of client connections and disconnections.

```
serdar@serdar-pc: ~/Desktop/mt
serdar@serdar-pc:~/Desktop/mt$ ./client tryConnect 95003
Connected to server (ServerPID: 95003) as client_95010
quit
serdar@serdar-pc:~/Desktop/mt$

serdar@serdar-pc:~/Desktop/mt
serdar@serdar-pc:~/Desktop/mt$ ./client Connect 95003
Connected to server (ServerPID: 95003) as client_95017
quit
serdar@serdar-pc:~/Desktop/mt$

serdar@serdar-pc:~/Desktop/mt$ ./server server_directory 1
Log file created: logfile_95003.log
Server Started PID 95003
Opening server FIFO for reading.
Server FIFO opened for reading.
Waiting for clients...
Client PID 95010 connected as 'client_1'
Connection request PID 95017 failed. Queue is FULL.
Client with PID 95017 waits...
Client with PID: 95010 disconnected.
Client PID 95017 connected as 'client_2'
Client with PID: 95017 disconnected.
```

(left top: client_1[tryconnect], left bottom: client_2[connect], right: server terminal)



The image shows three terminal windows. The top-left window shows a client trying to connect to a server with PID 95194 and then connecting as client_95240. The bottom-left window shows a client connecting to a server with PID 95194 and then connecting as client_95195. The right window shows the server's perspective, including log file creation, server startup, and handling of client connections and disconnections.

```
serdar@serdar-pc:~/Desktop/mt
serdar@serdar-pc:~/Desktop/mt$ ./client tryConnect 95194
Connected to server (ServerPID: 95194) as client_95240
Server is currently full. Not waiting in queue. Terminating
...

serdar@serdar-pc:~/Desktop/mt
serdar@serdar-pc:~/Desktop/mt$ ./client Connect 95194
Connected to server (ServerPID: 95194) as client_95195
quit
bye ;)serdar@serdar-pc:~/Desktop/mt$

serdar@serdar-pc:~/Desktop/mt$ ./server server_directory 1
Log file created: logfile_95194.log
Server Started PID 95194
Opening server FIFO for reading.
Server FIFO opened for reading.
Waiting for clients...
Client PID 95195 connected as 'client_1'
Connection request PID 95240 failed. Queue is FULL.
Client with PID: 95195 disconnected.
```

(left top: [tryconnect], left bottom: client_1[connect, right: server terminal)

As you can see, Connect client waits for its turn but tryConnect client does not.

2) Client Requests and Server Responses:

A client sends requests in a while loop which does not stop until kill signal, stop signal or quit command. In each iteration it checks if the server is still online by checking the Server FIFO status. If Server FIFO is still existing, then it waits for user to enter command and sends the command to the server via Client FIFO. Then client opens read end of Client FIFO and waits for the server to write a response. At the same time the server receives the command and extract arguments from it. Then it executes command, creates a response and writes it back into Client FIFO. Client FIFO receives the response and prints it.

2.1) 'help' commands:

```
help
Available commands are:
help, list, readF, writeT, upload, download, archServer, killServer, quit
help readF

readF <file> <line #>
Display the #th line of the <file>, returns with an error if <file> does not exists.
help writeT

writeT <file> <line #> <string>
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 Servers directory creates and edits thefile at the same time.
help upload

upload <file>
Uploads the file from the current working directory of client to the Servers directory.
help download

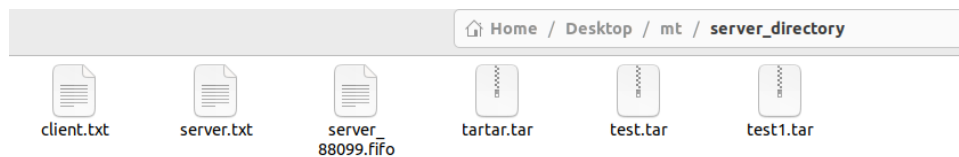
download <file>
Receive <file> from Servers directory to client side
help archServer

archServer <fileName>.tar
Using fork, exec and tar utilities create a child process that will collect all the files currently available on the the Server side and store them in the <filename>.tar archive
help killServer

killServer
Sends a kill request to the Server
```

help, help list, help readF, help writeT, help upload, help download, help archServer, help killServer commands can be used to receive information about functions and how functions work.

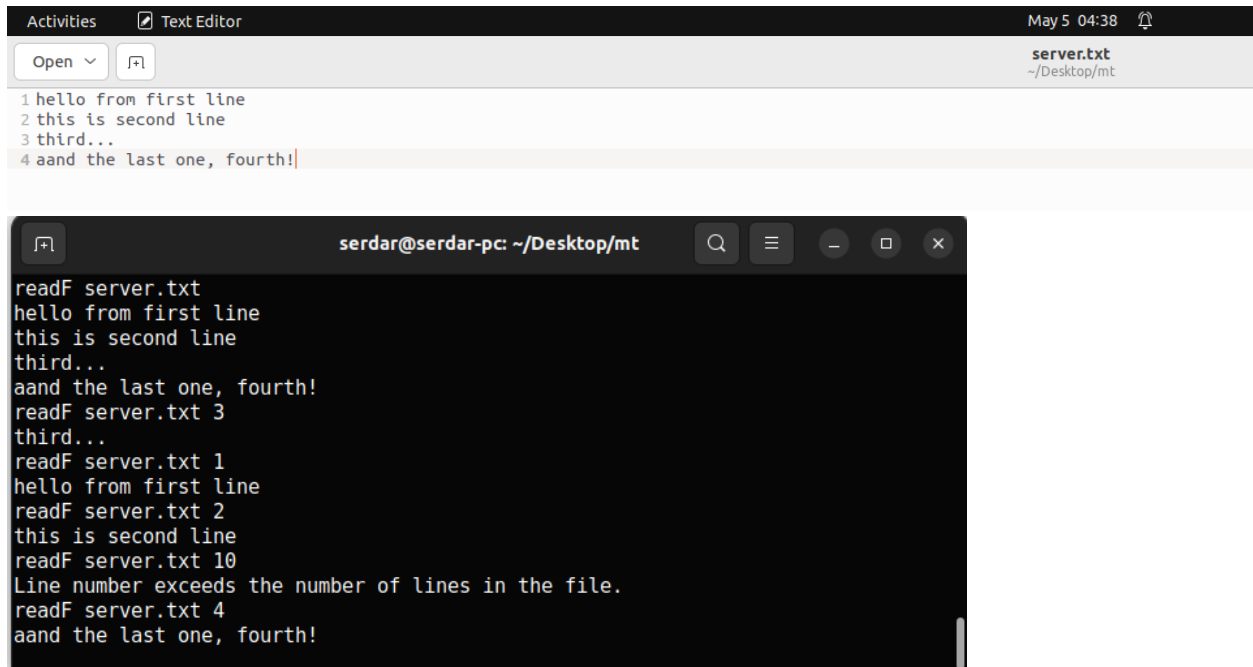
2.2) 'list' command:



```
serdar@serdar-pc: ~/Desktop/mt
list
tartar.tar
test1.tar
test.tar
server_88099.fifo
client.txt
server.txt
```

list command opens “server_directory” and reads directory using dirent struct and readdir() function (<dirent.h> library). Then sends file names as response one by one.

2.3) ‘readF’ command:



The image shows two windows. The top window is a text editor titled 'server.txt' located at '~/Desktop/mt'. It contains four lines of text: '1 hello from first line', '2 this is second line', '3 third...', and '4 aand the last one, fourth!'. The bottom window is a terminal titled 'serdar@serdar-pc: ~/Desktop/mt'. It shows the following sequence of commands and outputs: 'readF server.txt' outputs all four lines; 'readF server.txt 3' outputs 'third...'; 'readF server.txt 1' outputs 'hello from first line'; 'readF server.txt 2' outputs 'this is second line'; 'readF server.txt 10' outputs 'Line number exceeds the number of lines in the file.'; and 'readF server.txt 4' outputs 'aand the last one, fourth!'.

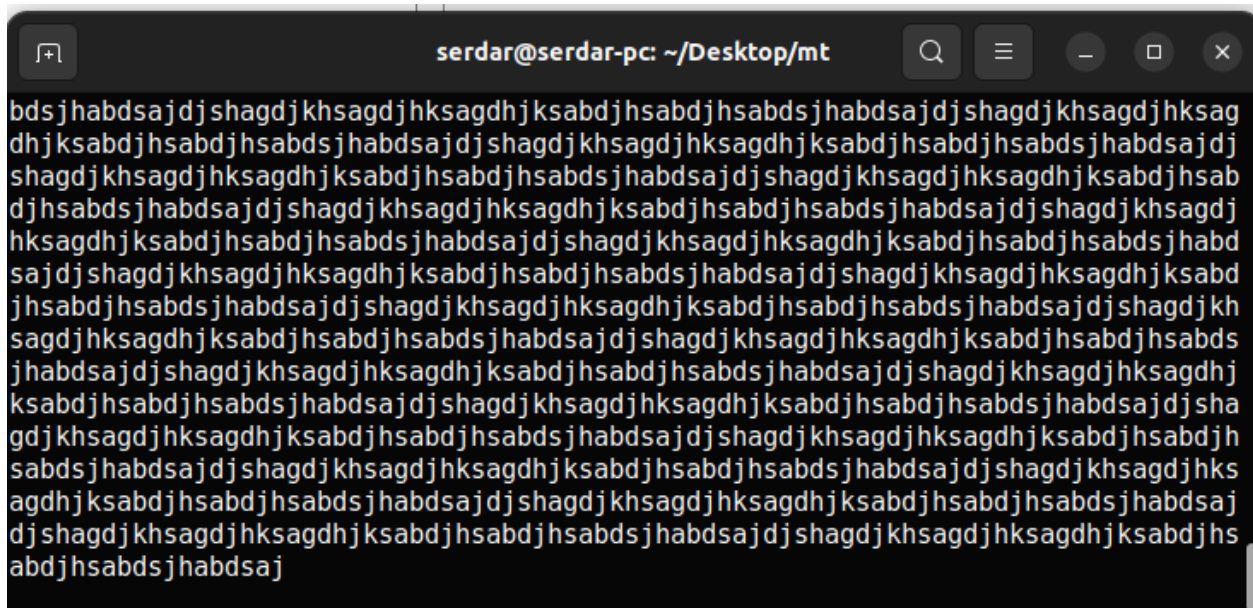
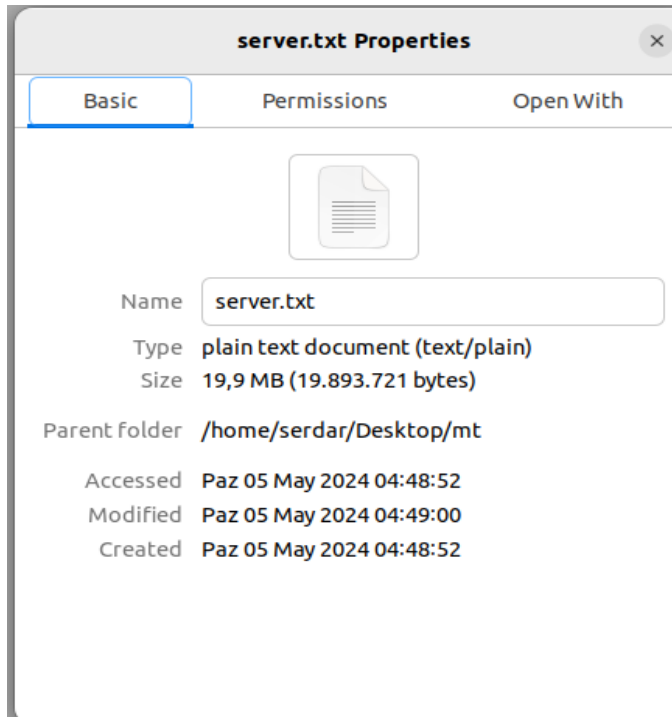
```
Activities Text Editor May 5 04:38
server.txt
~/Desktop/mt
1 hello from first line
2 this is second line
3 third...
4 aand the last one, fourth!

serdar@serdar-pc: ~/Desktop/mt
readF server.txt
hello from first line
this is second line
third...
aand the last one, fourth!
readF server.txt 3
third...
readF server.txt 1
hello from first line
readF server.txt 2
this is second line
readF server.txt 10
Line number exceeds the number of lines in the file.
readF server.txt 4
aand the last one, fourth!
```

readF <filename> <line> command reads all lines when; line argument is smaller than 1, or line argument is null. If line argument is bigger than the number of lines in the file server sends response: “Line number exceeds the number of lines in the file.” Otherwise, it reads the line from <line> of file. I show the test with 20 MB file in writeT part.

For this operation I read file char by char and increment current_line when ‘\n’ is encountered. If current_line is selected_line, I send that line as response to Client FIFO.

2.4) 'writeT' command:



(a part of output from readF for 20MB file)

```

djsahgdjksahgdjksahgdhjkhsabdjhsabdjhsabd
abdjhsabdsjhabdsaj
writeT server.txt lastline!
Successfully written into the file.

```

[illegible]

(output from readF for 20MB file after 'writeT server.txt lastline!' command)

```
readF server.txt
1
2
3
4
writeT server.txt 2 second
Successfully written into the file.
writeT server.txt 3 third
Successfully written into the file.
readF server.txt
1
second
third
4
```

writeT <filename> <line> <string> command writes <string> at the end of file when; line argument is smaller than 0, or line argument is null. If line argument is bigger than the number of lines in the file server sends response:

“Line number exceeds the number of lines in the file.” Otherwise, it writes <string> to the line from <line> of file.

For this operation I open a temp file, I write into temp until the selected line. When selected line == current line I write selected line, then I write the rest of the file. At the last part, I rewrite the file with write buffer which reads its data from temp file.

2.5) ‘upload’ and ‘download’ commands:

The image shows a file manager window and a terminal window. The file manager window displays the contents of the `server_directory` folder, which includes `server.txt` and `server_90401.fifo`. The terminal window shows the execution of several commands and their outputs:

```
serdar@serdar-pc: ~/Desktop/mt
upload server.txt
Error opening file for upload
download client.txt
File does not exist in the server's directory
upload client.txt
File uploaded successfully. Total bytes transferred: 37
download server.txt
File downloaded successfully. Total bytes transferred: 0
download server.txt
File already exists in the client's directory
upload client.txt
File already exists in the server's directory
```

The file manager window also shows the contents of the `client.txt` file, which is:

```
1 1
2 second
3 third
4 4
5 server.txt
6 -1ahaha
```


upload <filename> command creates a file in “server_directory” and copies the data from original file to newly created file.

When successful it prints: ‘File uploaded successfully. Total bytes transferred: %d’ (%d = bytes transferred).

If the original file does not exist, then it prints: ‘Error opening file for upload’.

If the file has already been uploaded to server’s directory before then it prints: ‘ File already exists in the server's directory’.

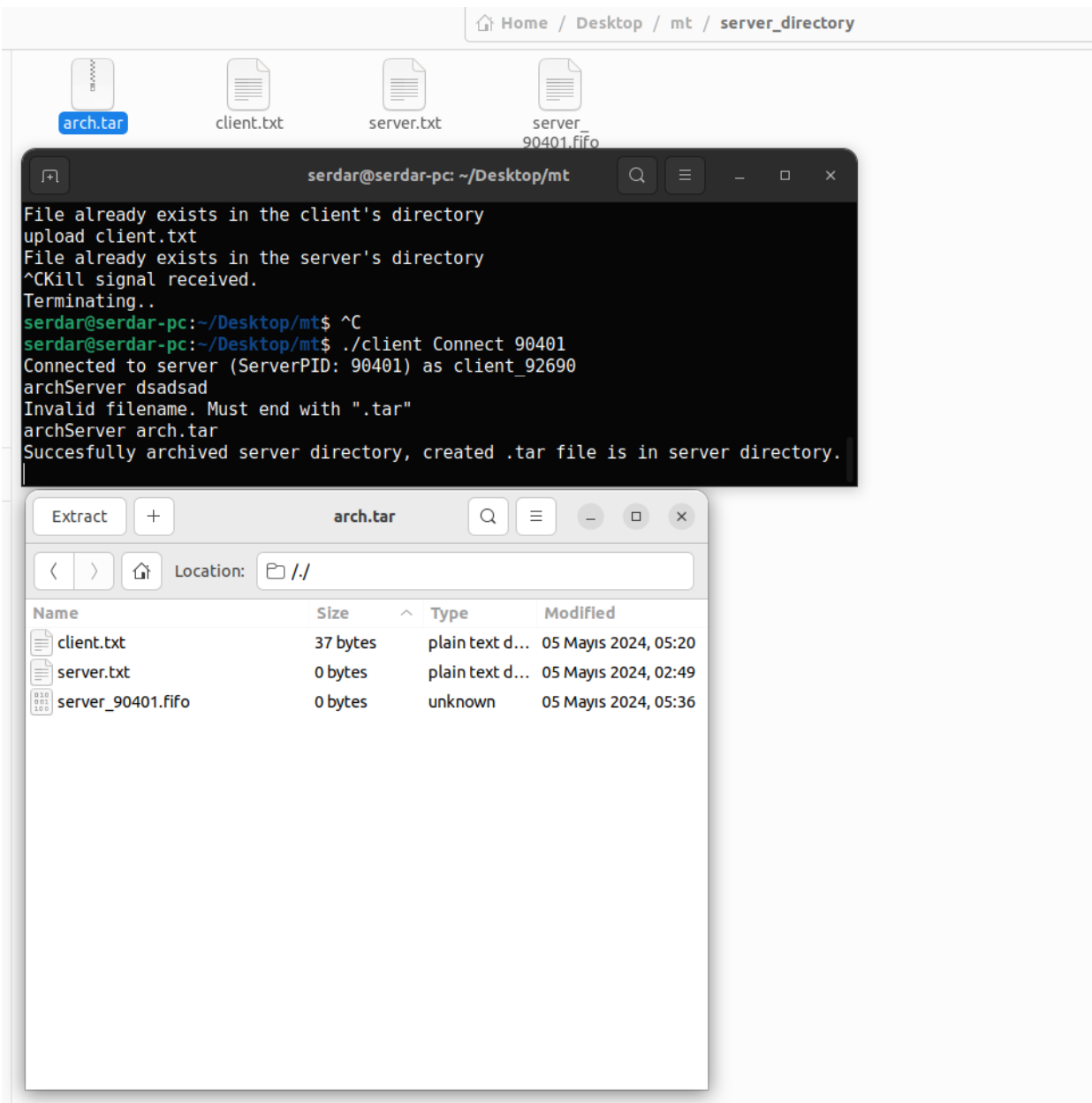
Download <filename> command creates a file in current_directory(the directory that we can writeT into files or readF from files, and the directory that contains log files and client fifos.) and copies the data from the original file (the file from server directory) to newly created file.

When successful it prints: ‘File downloaded successfully. Total bytes transferred: %d’ (%d = bytes transferred).

If the original file does not exist, then it prints: ‘File does not exist in the server's directory’.

If the file has already been uploaded to client’s directory before then it prints: ‘ File already exists in the client’s directory’.

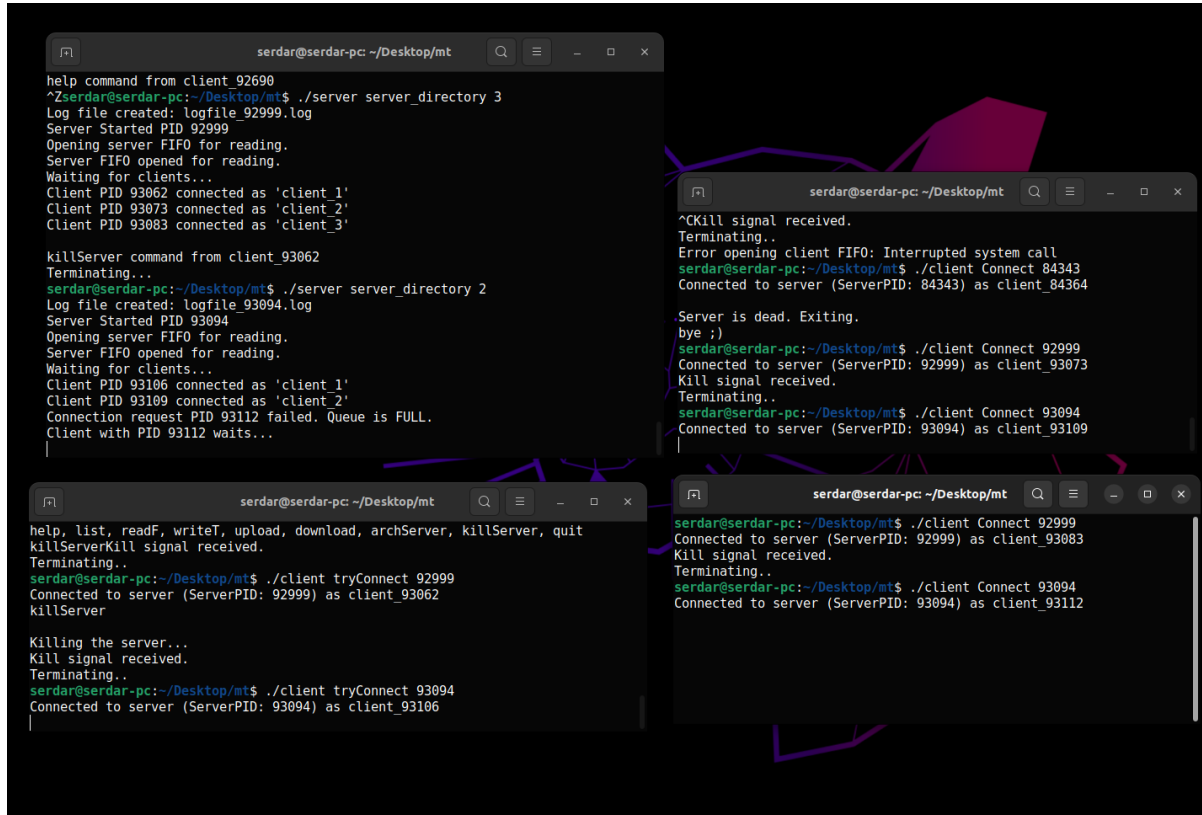
2.6) archServer command:



`archServer <archiveName>` function only accepts archive names that end with `‘.tar’`.

Then it creates a child process and in child process it opens “server_directory” using `chdir()` function. After changing directory, it calls **`exec1p("tar", "tar", "-cf", archiveName, ".", NULL);`** to create the `.tar` archive.

2.7) killServer command:



The image displays four terminal windows illustrating the process of terminating a server and its clients using the `killServer` command.

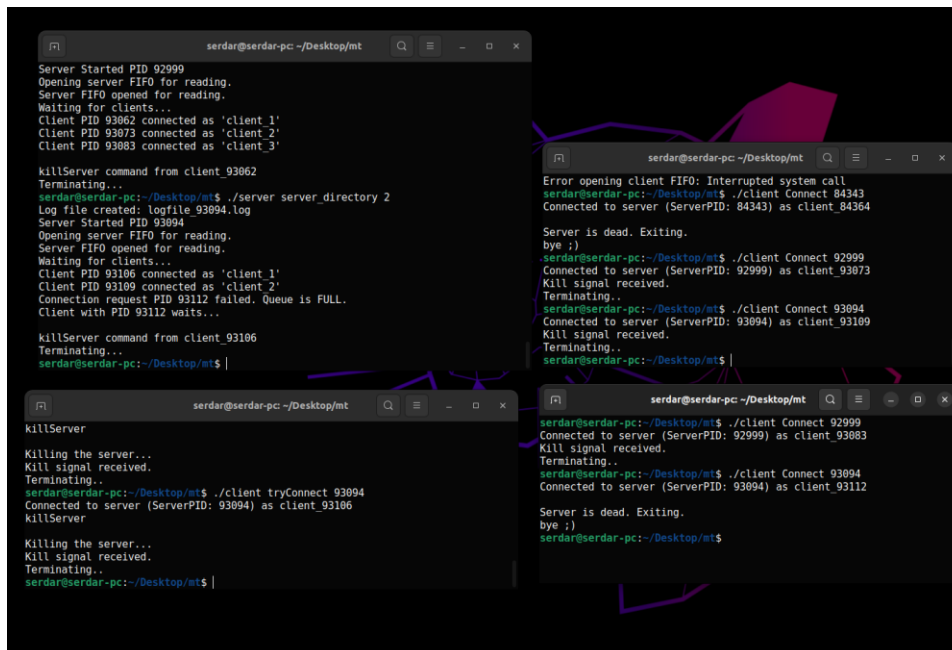
Top Left Window: Shows the server starting with PID 92999. It receives three client connections: 'client 1' (PID 93062), 'client 2' (PID 93073), and 'client 3' (PID 93083). A `killServer` command is received from client 93062, and the server begins terminating.

Top Right Window: Shows the server receiving a `^CKill signal received.` and terminating. It then receives a connection from client 84343 (ServerPID: 84343) and exits with the message "Server is dead. Exiting. bye ;)".

Bottom Left Window: Shows the server starting with PID 93094. It receives two client connections: 'client 1' (PID 93106) and 'client 2' (PID 93109). A connection request from PID 93112 fails because the queue is full. A `killServer` command is received from client 93106, and the server begins terminating.

Bottom Right Window: Shows the server receiving a `^CKill signal received.` and terminating. It then receives connections from client 92999 (ServerPID: 92999) and client 93083 (ServerPID: 93083). It then receives a `kill signal received.` and terminates.

(before `killServer`, 2 active clients and 1 client waiting in line)



The image displays four terminal windows showing the state of the server and clients after the `killServer` command has been executed.

Top Left Window: Shows the server starting with PID 92999. It receives three client connections: 'client 1' (PID 93062), 'client 2' (PID 93073), and 'client 3' (PID 93083). A `killServer` command is received from client 93062, and the server begins terminating.

Top Right Window: Shows the server receiving a `^CKill signal received.` and terminating. It then receives a connection from client 84343 (ServerPID: 84343) and exits with the message "Server is dead. Exiting. bye ;)".

Bottom Left Window: Shows the server starting with PID 93094. It receives two client connections: 'client 1' (PID 93106) and 'client 2' (PID 93109). A connection request from PID 93112 fails because the queue is full. A `killServer` command is received from client 93106, and the server begins terminating.

Bottom Right Window: Shows the server receiving a `^CKill signal received.` and terminating. It then receives connections from client 92999 (ServerPID: 92999) and client 93083 (ServerPID: 93083). It then receives a `kill signal received.` and terminates.

(after `killServer` in an active client, all clients and server are terminated)

Note: For clients that were waiting in wait_queue before termination, it is necessary to press enter in their terminal for terminating safely and unlinking their fifos.

3) Synchronization:

I have a shared memory and in that shared memory I store **Shared* sd;**

```
typedef struct shared_memory_files {
```

```
    int capacity;
```

```
    int size;
```

```
    File files[64];
```

```
} Shared;
```

```
typedef struct {
```

```
    char name[255];
```

```
    int reader_count;
```

```
    sem_t reader_mutex;
```

```
    sem_t writer_mutex;
```

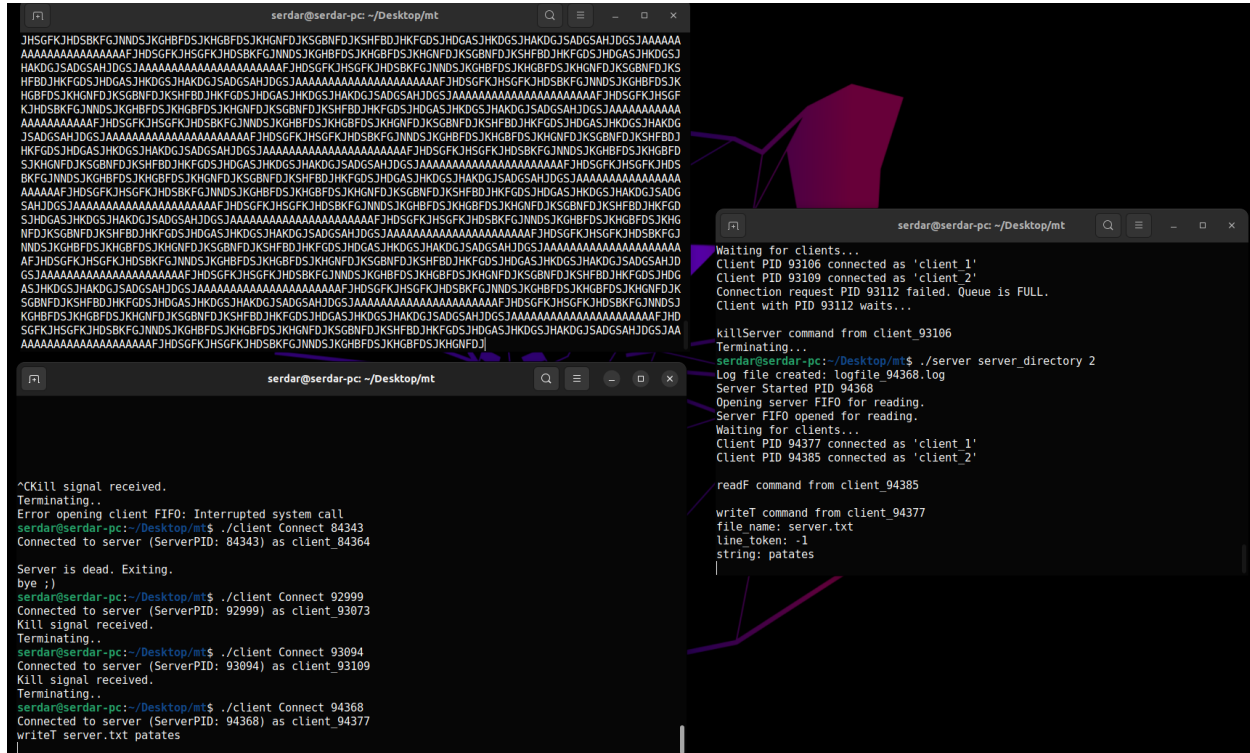
```
} File;
```

This **sd** pointer points to a struct that stores array of File structs which store file names, an integer that holds the count of readers currently reading that file and semaphores to synchronize read and write operations.

At the start of my main code, I initialize files inside my current_directory to shared memory and I always initialize newly created files inside my current_directory (the only way of initialization in runtime is calling writeT with non-existing filename).

I used semaphores to synchronize my code, I configured my semaphores to allow many amounts of readers but only one writer at the same time. If there is a writer, there can't be any more writers or readers. I defined and used

3.1) Proof of my synch working:



The client in upper terminal is currently reading a very big file (76 MB) and the client in lower terminal has sent its writeT command, on the right terminal (server terminal) you can see that writeT command is succesfully received but it is not being written yet because it waits for reading to be over.

```
serdar@serdar-pc: ~/Desktop/mt
AFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJKSGBNFDJKSHFBDJHKFGDSJHDGASJHKDGSJHAKDGJSADGSAHJD
GSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJKSGBNFDJKSHFBDJHKFGDSJHDG
ASJHKDGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJK
SGBNFDJKSHFBDJHKFGDSJHDGASJHKDGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJ
KGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
SGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
AAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
DGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJ
DKSHFBDJHKFGDSJHDGASJHKDGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHB
DSJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
KGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
AAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
AKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKSH
FBDJHKFGDSJHDGASJHKDGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKH
GBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
JHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
JHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
AAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
SADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
KFGDSJHDGASJHKDGSJHAKDGJSADGSAHJDGSJAAAAAAAAAAAAAAAAAFJHDSGFKJHSGFKJHDSBKFJNNDSJKGHBFDJJKHGNFDJ
JKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJJKHGNFDJ
patates

serdar@serdar-pc: ~/Desktop/mt

^CKill signal received.
Terminating..
Error opening client FIFO: Interrupted system call
serdar@serdar-pc: ~/Desktop/mt$ ./client Connect 84343
Connected to server (ServerPID: 84343) as client_84364

Server is dead. Exiting.
bye ;)
serdar@serdar-pc: ~/Desktop/mt$ ./client Connect 92999
Connected to server (ServerPID: 92999) as client_93073
Kill signal received.
Terminating..
serdar@serdar-pc: ~/Desktop/mt$ ./client Connect 93094
Connected to server (ServerPID: 93094) as client_93109
Kill signal received.
Terminating..
serdar@serdar-pc: ~/Desktop/mt$ ./client Connect 94368
Connected to server (ServerPID: 94368) as client_94377
writeT server.txt patates
Successfully written into the file.

serdar@serdar-pc: ~/Desktop/mt
Client PID 93109 connected as 'client_2'
Connection request PID 93112 failed. Queue is FULL.
Client with PID 93112 waits...

killServer command from client_93106
Terminating...
serdar@serdar-pc: ~/Desktop/mt$ ./server server_directory 2
Log file created: logfile_94368.log
Server Started PID 94368
Opening server FIFO for reading.
Server FIFO opened for reading.
Waiting for clients...
Client PID 94377 connected as 'client_1'
Client PID 94385 connected as 'client_2'

readF command from client_94385

writeT command from client_94377
file name: server.txt
line token: -1
string: patates

readF command from client_94385
```

Here you can see that after reading is done, writeT stops waiting and executes successfully.