

**GEBZE**  
**TEKNİK ÜNİVERSİTESİ**



**SPRING 2023 – CSE 344 SYSTEM PROGRAMMING**

**MIDTERM PROJECT**

**MERT GÜRŞİMŞİR**

**1901042646**

## System Architecture, Design Decisions & Implementation Details

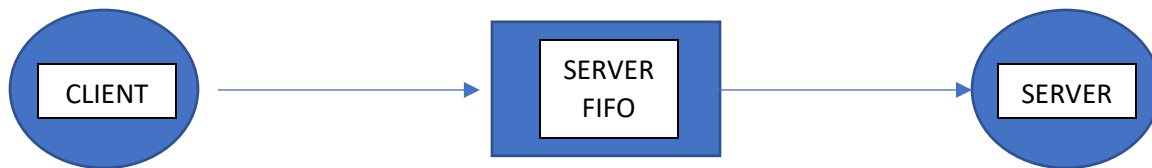
To understand the needs of this project we need to know the concepts of inter process communication (IPC), file input output, signals, processes, and synchronization.

For IPC, I have used FIFO for my system. First of all, high-level architecture has to be designed. The maximum number of client processes has been given to the system as input from the user. Semaphores are used to run at most given number of clients.

- Wait semaphore when you fork for the client process.
- Post semaphore when you done with client process.

With this design, system is able to run at most maximum number of clients at the same time.

Other important issue that is need to be talked about is design of FIFOs. To connect to the server, server FIFO is used with the same name as server's process id:

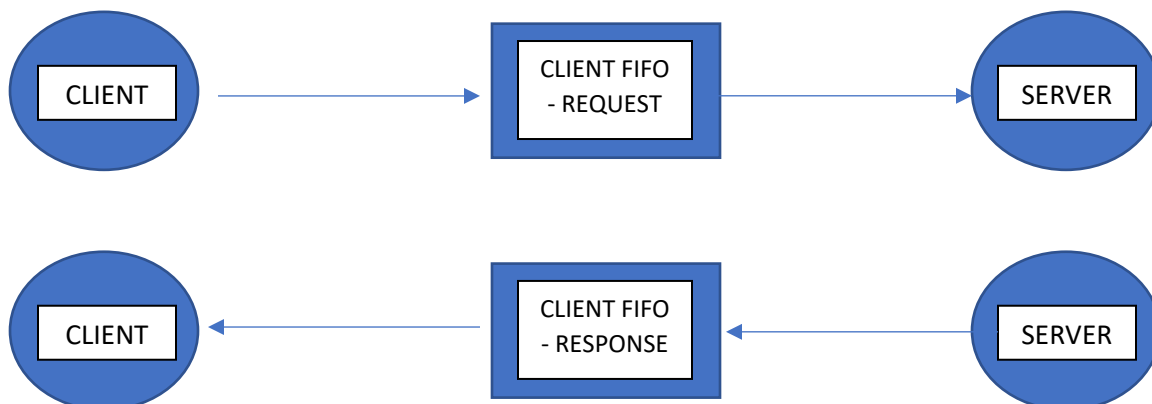


Client writes one end of this FIFO while server reads other end all the time. Whenever new request (that consists of client's current working directory, its pid, and request message) comes from the client, server immediately forks and this new child process takes care of the client.

This new child process takes care of 2 things:

- request comes to the server
- response goes to the client

For these 2 purposes, I have created 2 FIFOs with the name of the client's pid:

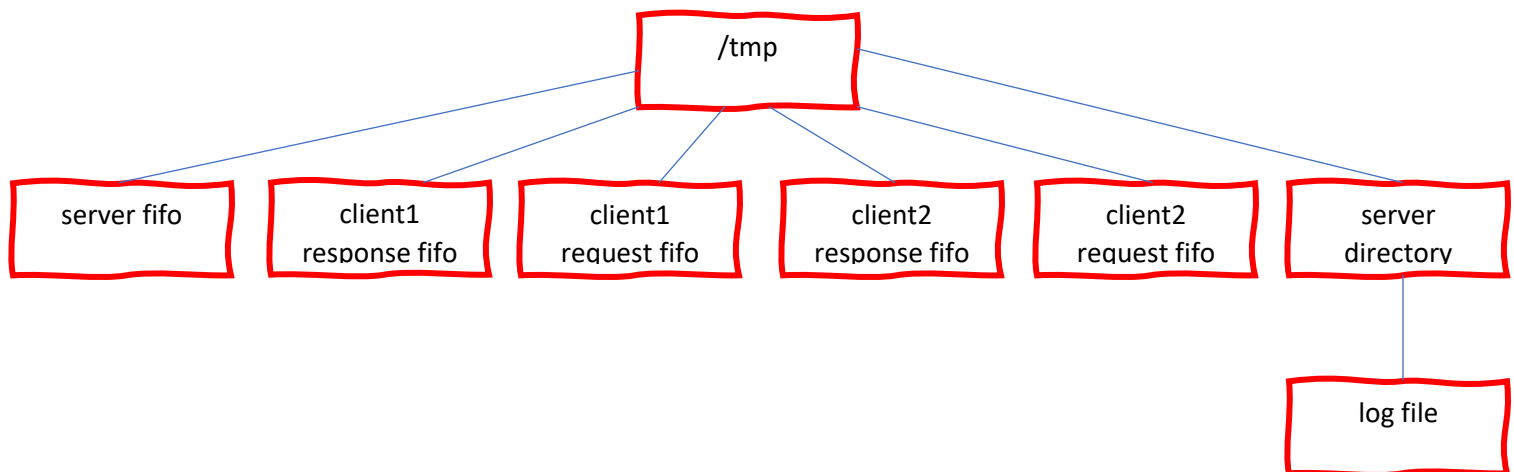


Forked process gets the request from the client, tokenize the input and then proper job is done by this process. Then result is sent back to the client from other FIFO. Connected client and its pid is written to the log file during this process. If request is quit, this process exits and also posts the semaphore.

On the other hand, client opens server FIFO with given pid and waits for the other end. If connection is successful, client program starts sending its requests through request FIFO created by itself.

For synchronization, I have used semaphores. Whenever a file job is going to be done, code waits for the binary semaphore and at the end, semaphore is posted.

Architecture is mostly based on /tmp directory. Server's directory is being created here. Server's FIFO and client's 2 FIFOs are also here. Client access to /tmp/{ServerPID} file which is FIFO to connect to the server.



Client FIFOs can be as many as possible in theory.

Client program can be executed wherever user wants. All the system are on top of /tmp directory so proper name alignments are being done for the program.

Signal handler for server is doing these:

- Writes all of the child process (clients) information to the log file and then deletes the client FIFOs.
- Closes all the file descriptors.
- Frees dynamically allocated spaces.
- Destroy semaphores.
- Unlink shared memory.
- Deletes the server FIFO.

Forked client process is another process that is independent of main server process. So semaphore value is needed to be shared between all the clients. For that purpose, shared memory is used to keep semaphore so that every client can post it or wait it.

For client requests, my implementations are goes like these:

- readF
  - If line number is not given, read the whole file and assign it to an array that is dynamically allocated.
  - If line number is given, read the file byte by byte and count the '\n' characters. If wanted line is reached, read it to buffer.
- writeT
  - If line is not given, write directly to the end by opening the file with O\_APPEND flag.
  - Get to the line if it is given by reading the file, save what comes next until end of the file, then write given string and then write again the saved part of the file.
- upload/download
  - These 2 requests simply opens the file in the corresponding directory, read it to dynamically allocated buffer, then writes it to the file that is newly created with same name to the corresponding directory.

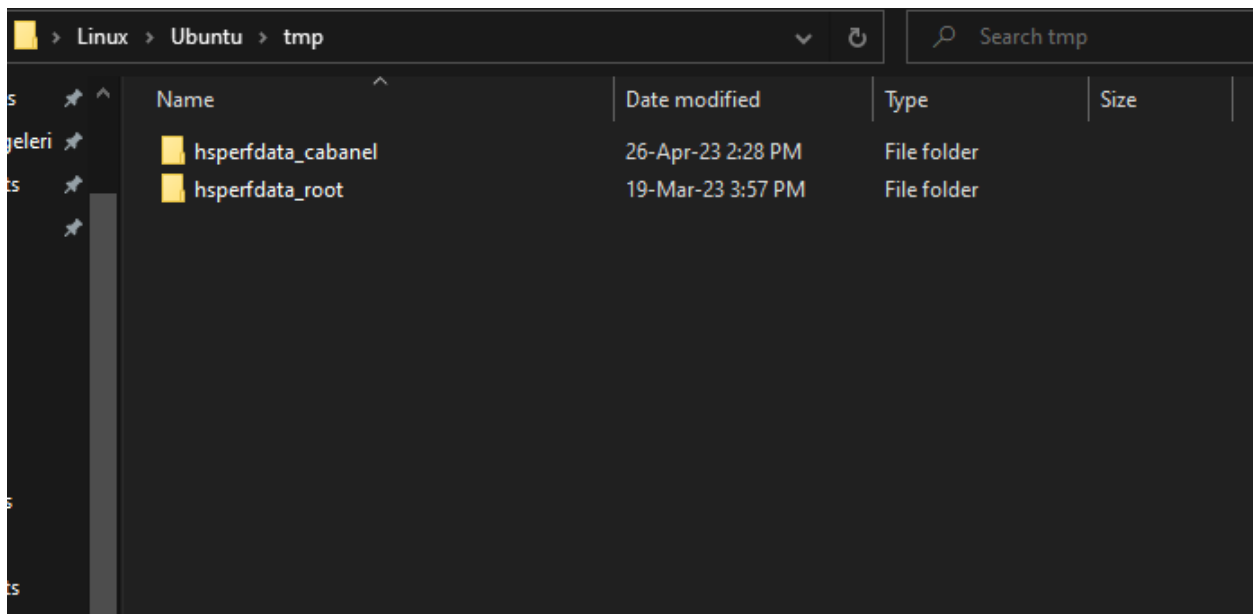
## Tests

To be sure that this system works properly, these parts should be tested:

- Can client and server communicate properly?
- Can client that exceeds the maximum client number wait for one of other processes to finish?
- Can waiting client connect to the server when there is enough place for it?
- Can server serve to many clients which are connected to itself at the same time?
- Can client display the requests that it can perform and learn the details of them?
- Can client see the list of files in server's directory?
- Can client read file that is at the server side?
- Can client write to file that is at the server side?
- Can client upload a file to the server's directory?
- Can client download a file from server's directory?
- Can log file be created and filled properly?
- Can client quit from the server that it is connected to?
- Can client sends a kill request to server?
- Can server handle the signal that tries to kill it? It needs to perform some operations?
- Can client connect with tryConnect when queue is empty?
- Does client wait when it tries to connect with tryConnect when queue is full?

### 1) Can client and server communicate properly?

/tmp directory before running the code:



/tmp directory after running the biboServer.c:

Name	Date modified	Type	Size
hsperfdata_cabanel	26-Apr-23 2:28 PM	File folder	
hsperfdata_root	19-Mar-23 3:57 PM	File folder	
mert	17-May-23 4:05 AM	File folder	
3606	17-May-23 4:05 AM	File	0 KB

3606 is the pid of the server.

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 1
Server Started PID 3606...
waiting for clients...
```

Now server is waiting for clients to connect to it. Let's connect:

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 1
Server Started PID 3606...
waiting for clients...
Client PID 3611 connected as "client1"

cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3606
>> Waiting for Que...Connection established:
>>
```

As you can see client and server communicated and client is connected to server.

## 2) Can client that exceeds the maximum client number wait for one of other processes to finish?

As you can see from the previous example, we have chosen maximum client number as 1. So next client that connects to the server will have to wait for the process with pid 3611.

```
cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 1
Server Started PID 3606...
waiting for clients...
Client PID 3611 connected as "client1"
Connection request PID 3615... Que FULL
-

cabanel@MSI: /mnt/c/Users/Mert/Desktop

>> quit
bye
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3606
>> Waiting for Que...Connection established:
>>

cabanel@MSI: /mnt/c/Users/Mert/Desktop

>> quit
bye
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3606
>> Waiting for Que...
```

New client is waiting for the queue to be empty. When process with pid 3611 finishes its execution, waiting process can communicate with server.

### 3) Can waiting client connect to the server when there is enough place for it?

```
cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 1
Server Started PID 3606...
waiting for clients...
Client PID 3611 connected as "client1"
Connection request PID 3615... Que FULL
client1 disconnected...
Client PID 3615 connected as "client2"

cabanel@MSI: /mnt/c/Users/Mert/Desktop

>> quit
bye
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3606
>> Waiting for Que...Connection established:
>> quit
bye
cabanel@MSI:/mnt/c/Users/Mert/Desktop$

cabanel@MSI: /mnt/c/Users/Mert/Desktop

>> quit
bye
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3606
>> Waiting for Que...Connection established:
>>
-
```

As it is seen, when client 1 disconnects, client 2 connects to the server.

Also this execution answers the question: "Can client quit from the server that it is connected to?". It can quit and disconnected message is shown at server side immediately. Then other clients may be served.

#### 4) Can server serve to many clients which are connected to itself at the same time?

```
cabanel@MSI: /mnt/c/Users/Mert/Desktop
Ctrl+C pressed. Exiting...
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 2
Server Started PID 3620...
waiting for clients...
Client PID 3621 connected as "client1"
Client PID 3624 connected as "client2"

cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3620
>> Waiting for Que...Connection established:
>> help

    Available commands are:

help, list, readF, writeT, upload, download, quit, killServer

>>

cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3620
>> Waiting for Que...Connection established:
>> list
.
..
LOG.txt
>>
```

Maximum number of clients is 2 now. So 2 clients can be connected to the server and server can serve to both of them.

With this execution we know the answer of questions "Can client display the requests that it can perform?" and "Can client see the list of files in server's directory?".

To be able to display details of requests, following format is used → "help {request}"



```
>> help list

list
    sends a request to display the list of files in Servers directory
    (also displays the list received from the Server)

>>
```

```
>> help upload


upload <file>
    uploads the file from the current working directory of client to the Servers directory
    (beware of the cases no file in clients current working directory and file with the same
    name on Servers side)
```

You can learn about all the details of all the requests like these.

### 5) Can client read file that is at the server side?

For this test, a txt file is copied to the server directory.

---

 test - Notepad  
File Edit Format View Help  
Awakenings  
boyhood 2014  
THE NIGHT HOUSE 2020  
CREEP 2014  
RUN 2020  
ELEPHANT  
THE PALE BLUE EYE  
marriage story  
manchester by the sea  
saving private ryan  
schindlers list

Let's read the 3<sup>rd</sup> line of this file:

```
>> readF test.txt 10
saving private ryan
>>
```

If you want, you can read the all file too:

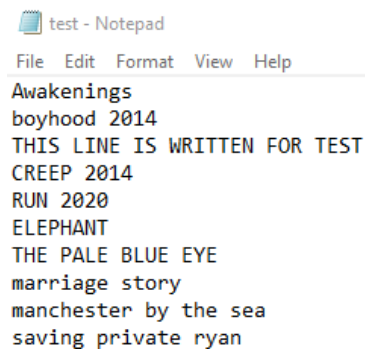
```
>> readF test.txt
Awakenings
boyhood 2014
THE NIGHT HOUSE 2020
CREEP 2014
RUN 2020
ELEPHANT
THE PALE BLUE EYE
marriage story
manchester by the sea
saving private ryan
schindlers list
marley and me
end of watch
shame steve mcqueen
pandoranın kutusu
ZEITGEIST
PASSENGERS
BARBARIAN
What Josiah Saw
TRUTH AND JUSTICE (2019) - Tanel Toom
PIYANIST
CESUR YÜREK
SCARFACE
OLD BOY
ÇİZGİLİ PİJAMALI ÇOCUK
PUBLIC ENEMIES
LIFE IS BEAUTIFUL
GODFATHER
KARA ŞÖVALYE
REVENANT
SONSUZLUK BİR GÜN
```

#### 6) Can client write to file that is at the server side?

```
>> writeT test.txt 3 THIS LINE IS WRITTEN FOR TEST
SUCCESS
>>
```

“SUCCESS” message is returned to us from server.

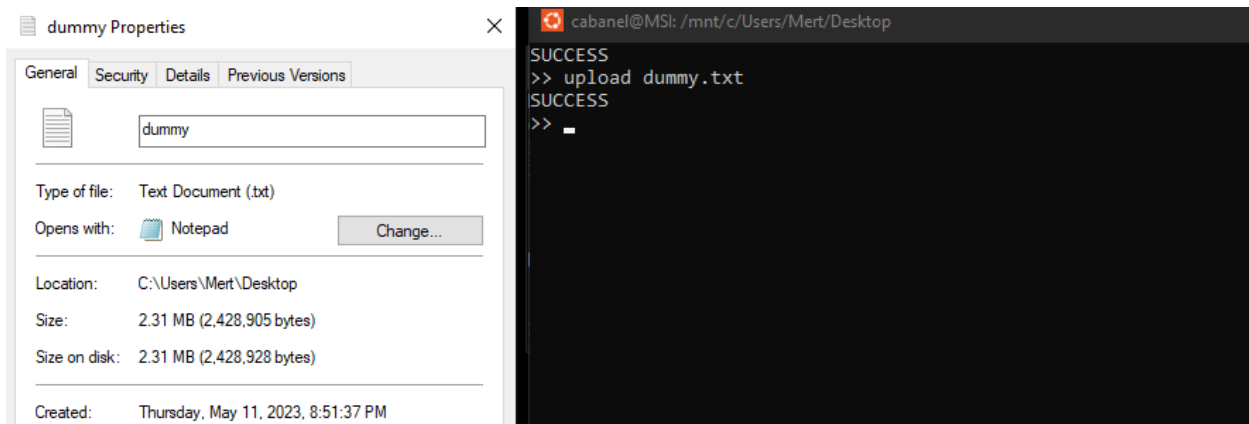
After this request file becomes this:



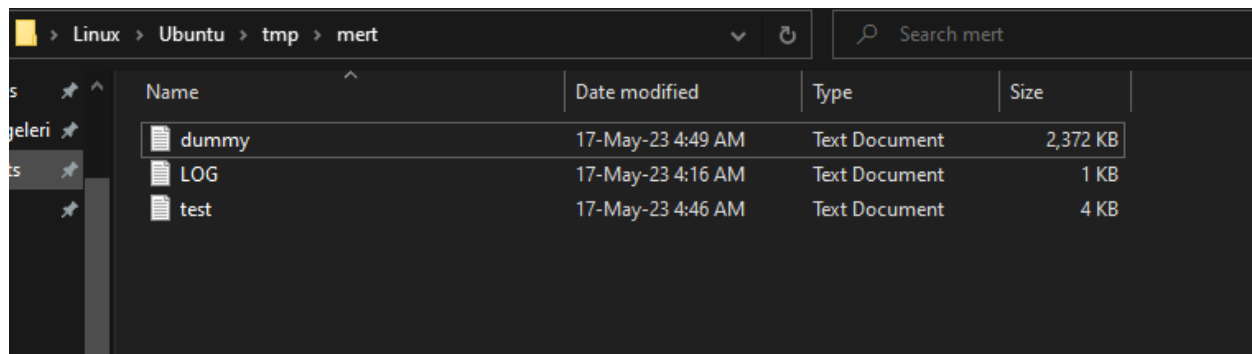
```
test - Notepad
File Edit Format View Help
Awakenings
boyhood 2014
THIS LINE IS WRITTEN FOR TEST
CREEP 2014
RUN 2020
ELEPHANT
THE PALE BLUE EYE
marriage story
manchester by the sea
saving private ryan
```

### 7) Can client upload a file to the server's directory?

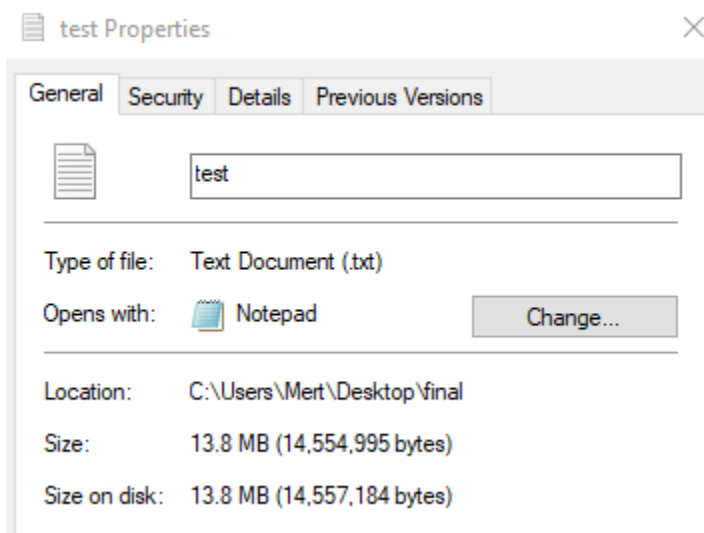
dummy.txt is the file that is at the client's side.



When we upload it, we can see it at the server's side:






We can try bigger file for uploading such as this test.txt file at client's side:



```

cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboClient tryConnect 4144
>> Waiting for Que...Connection established:
>> upload test.txt
SUCCESS
>>

```

Ubuntu > tmp > mert			
Name	Date modified	Type	Size
 dummy	17-May-23 4:49 AM	Text Document	2,372 KB
 LOG	17-May-23 6:18 AM	Text Document	1 KB
 test	17-May-23 6:23 AM	Text Document	14,214 KB

test - Notepad

File Edit Format View Help

POSIX THREADS

- Introduction to Threads
- POSIX Threads
- The Pthread API
- Threads vs Processes

Threads

Like processes, threads are a mechanism that permits an application to perform multiple tasks concurrently. A single process can contain multiple threads

All threads are independently executing the same program, and they all share the same global memory, including the initialized data, uninitialized data, and heap segments.

The threads in a process can execute concurrently. On a multiprocessor system, multiple threads can execute parallel. If one thread is blocked on I/O, other threads are still eligible to execute.

THREADS ENABLES USER TO DO CONCURRENT PROGRAMMING WITH MULTIPLE TASKS.

WITH PROCESSES WE USE SO MANY SYSTEM RESOURCES. FOR SOME OF THE APPLICATIONS, WE DON'T HAVE TO USE THAT MUCH OF RESOURCES. ALSO IPC CAUSES A LOT OF WORK.

THREADS CAN EXECUTE CONCURRENTLY ON A MULTIPROCESSOR SYSTEM.

IF ONE OF THE THREADS BLOCKED, OTHER THREADS CAN CONTINUE EXECUTION.

Ln 1, Col 14 100% Windows (CRLF) UTF-8

test - Notepad

File Edit Format View Help

POSIX THREADS

- Introduction to Threads
- POSIX Threads
- The Pthread API
- Threads vs Processes

Threads

Like processes, threads are a mechanism that permits an application to perform multiple tasks concurrently. A single process can contain multiple threads

All threads are independently executing the same program, and they all share the same global memory, including the initialized data, uninitialized data, and heap segments.

The threads in a process can execute concurrently. On a multiprocessor system, multiple threads can execute parallel. If one thread is blocked on I/O, other threads are still eligible to execute.

THREADS ENABLES USER TO DO CONCURRENT PROGRAMMING WITH MULTIPLE TASKS.

WITH PROCESSES WE USE SO MANY SYSTEM RESOURCES. FOR SOME OF THE APPLICATIONS, WE DON'T HAVE TO USE THAT MUCH OF RESOURCES. ALSO IPC CAUSE!

LOT OF WORK.

THREADS CAN EXECUTE CONCURRENTLY ON A MULTIPROCESSOR SYSTEM.

IF ONE OF THE THREADS BLOCKED, OTHER THREADS CAN CONTINUE EXECUTION.

Contents are the same.

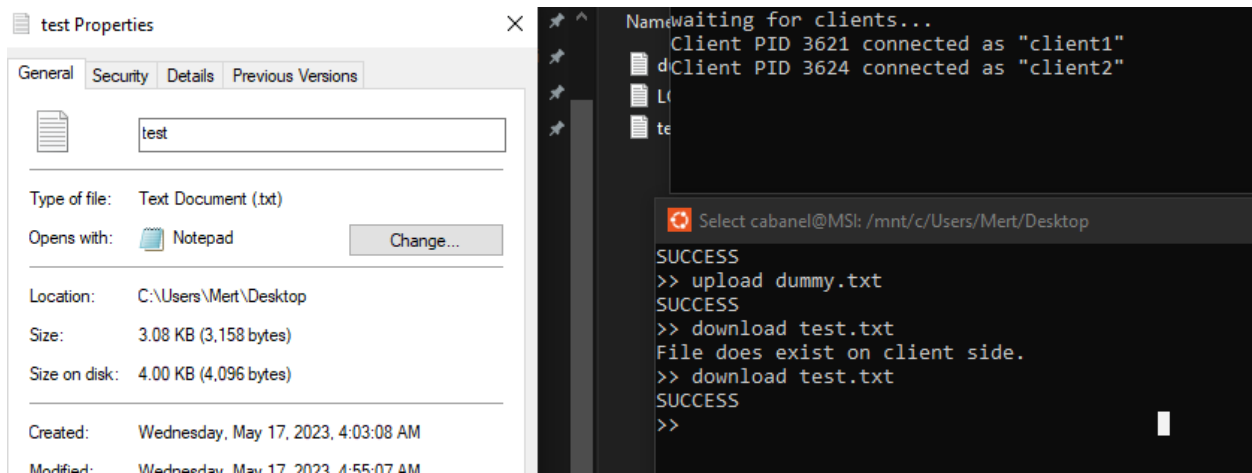
### 8) Can client download a file from server's directory?

```
>> download test.txt
SUCCESS
>>
```

test.txt is at the server's side but not in the client's side. If it exists at client's side too, we would get this message:

```
>> download test.txt
File does exist on client side.
>>
```

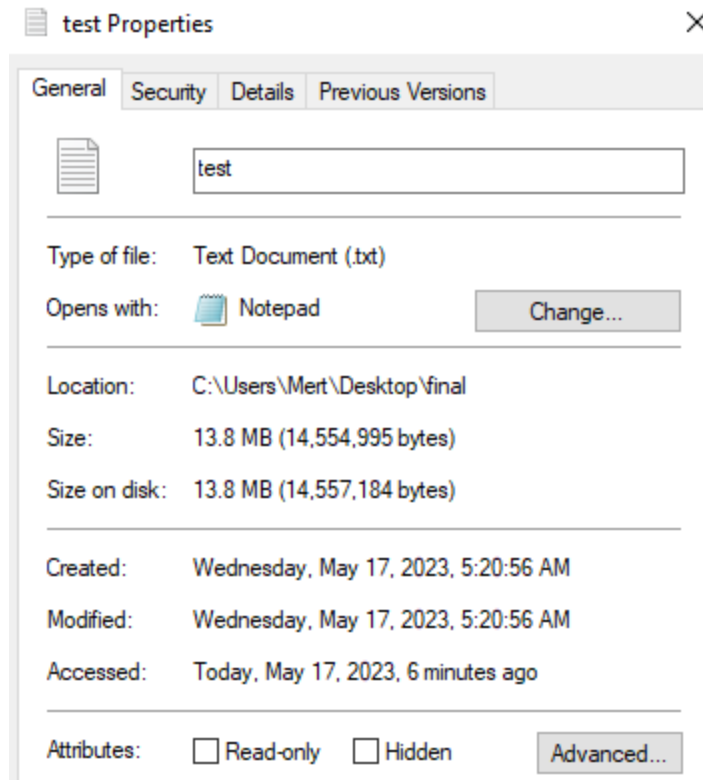
We can see the test.txt at client's side:



We can try bigger file for downloading such as this new test.txt file at server's side:

Name	Date modified	Type	Size
dummy	17-May-23 4:49 AM	Text Document	2,372 KB
LOG	17-May-23 4:16 AM	Text Document	1 KB
test	16-May-23 10:06 PM	Text Document	14,214 KB

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboClient Connect 4061
>> Waiting for Que...Connection established:
>> download test.txt
SUCCESS
>>
```

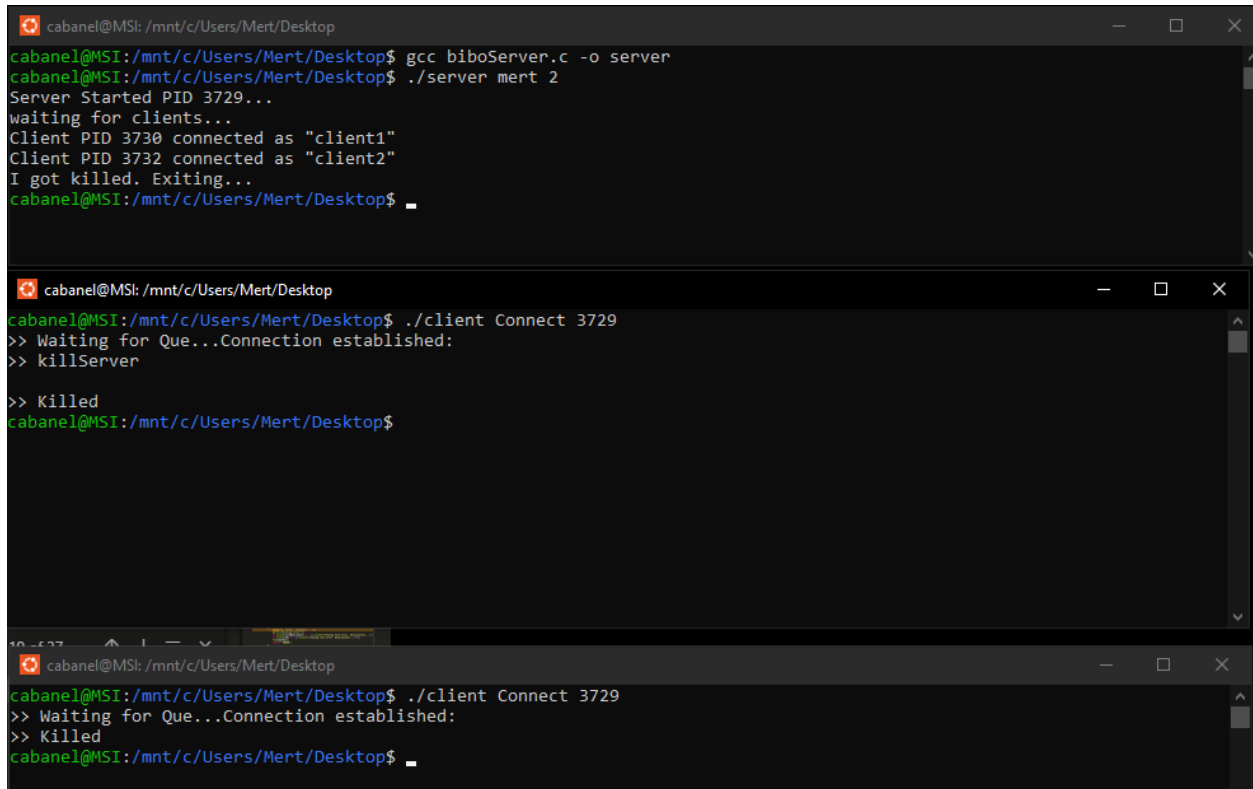


### 9) Can log file be created and filled properly?

At the log file, you can see all the processes that are connected, disconnected, or killed:

```
LOG - Notepad
File Edit Format View Help
3611 connected.
3611 disconnected.
3615 connected.
3611 is killed.
3615 is killed.
3621 connected.
3624 connected.
|
```

### 10) Can client sends a kill request to server?



```
cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ gcc biboServer.c -o server
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./server mert 2
Server Started PID 3729...
waiting for clients...
Client PID 3730 connected as "client1"
Client PID 3732 connected as "client2"
I got killed. Exiting...
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ _

cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3729
>> Waiting for Que...Connection established:
>> killServer

>> Killed
cabanel@MSI:/mnt/c/Users/Mert/Desktop$

cabanel@MSI: /mnt/c/Users/Mert/Desktop
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./client Connect 3729
>> Waiting for Que...Connection established:
>> Killed
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ _
```

As it can be seen, when killServer request is sent, server got killed and all other processes are killed too.

Server needs to perform some operations when it got the kill signal.

There are several things that are needed to be done. All these things are discussed at page 3. Code snippet for handler looks like this:

```

void sigint_handler(int signum) {
    printf("I got killed. Exiting...\n");
    int i;
    int fd;
    char pid[50];
    for (i = 0; i < pid_pos; ++i){
        kill(childPIDs[i], SIGKILL);
        sprintf(pid, "%d", childPIDs[i]);
        write(logFile, pid, sizeof(pid_t));
        write(logFile, " is killed.\n", 12);
        sprintf(pid, "/tmp/%d_res", childPIDs[i]);
        unlink(pid);
        sprintf(pid, "/tmp/%d_req", childPIDs[i]);
        unlink(pid);
    }
    for (fd = 0; fd < getdtablesize(); fd++) {
        close(fd);
    }
    if (buffer!=NULL) free(buffer);
    if (new_buffer!=NULL) free(new_buffer);
    if (bufferTemp.string!=NULL) free(bufferTemp.string);

    sem_destroy(sem);
    sem_destroy(semF);
    if (shm_unlink("/my_sem") == -1) {
        perror("shm_unlink error. Exiting...");
    }
    if (shm_unlink("/my_sem2") == -1) {
        perror("shm_unlink error. Exiting...");
    }
    if (shm_unlink("/my_sem3") == -1) {
        perror("shm_unlink error. Exiting...");
    }
    unlink(fifodir);
    exit(signum);
}

```

### 11) Can client connect with tryConnect when queue is empty?

```

cabanel@MSI: /mnt/c/Users/Mert/Desktop/final
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ make
make: Nothing to be done for 'all'.
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboServer mert 1
Server Started PID 4144...
waiting for clients...
Client PID 4147 connected as "client1"

cabanel@MSI: /mnt/c/Users/Mert/Desktop/final
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboClient tryConnect 4144
>> Waiting for Que...Connection established:
>>

```

Client can connect with tryConnect if there is space.



```
cabanel@MSI: /mnt/c/Users/Mert/Desktop/final
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ make
make: Nothing to be done for 'all'.
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboServer mert 1
Server Started PID 4144...
waiting for clients...
Client PID 4147 connected as "client1"
-

cabanel@MSI: /mnt/c/Users/Mert/Desktop/final
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboClient tryConnect 4144
>> Waiting for Que...Connection established:
>>

cabanel@MSI: /mnt/c/Users/Mert/Desktop/final
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$ ./biboClient tryConnect 4144
>> Waiting for Que...Que is full. Goodbye.
cabanel@MSI:/mnt/c/Users/Mert/Desktop/final$
```

Other client cannot connect because queue is full.