CSE 344
SYSTEM PROGRAMMING

MIDTERM
REPORT

OGUZ MUTLU
1801042624

1. PROBLEM DEFINITION

Design and implement a file server that enables multiple clients to connect, access and modify the contents of files in a specific directory
- in server side = biboServer <dirname> <max.#ofClients>
enters specified directory and create a log file for the clients and prompt it its PID,

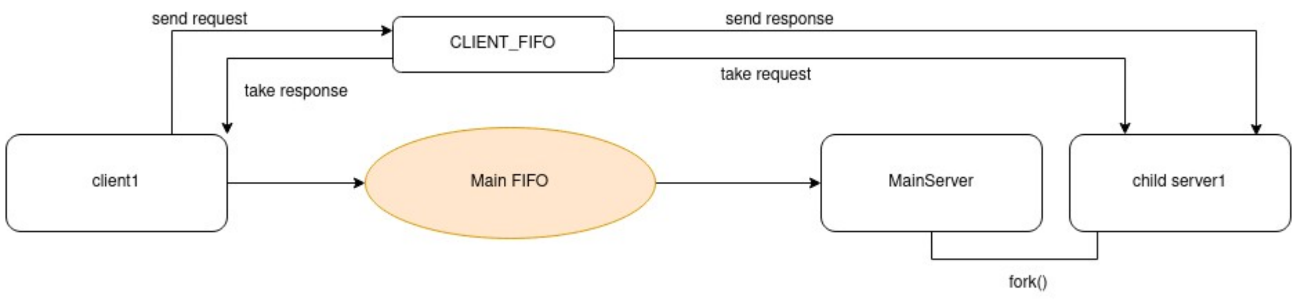
- in client side the client program with Connect option request a spot from the Server Que with ServerPID and
connects if a spot is available (if not the client should wait until a spot becomes available,

help, list, readF, writeT, upload, download, quit, killServer implement → implement commands.

• Program should be able to handle large files (i.e. > 10 MB).
•Program should be able to handle different file formats (i.e. text, binary).
• Program should use multiple processes for file access and synchronization.
• Signals should be handled properly on both client and Server sides

2. PROBLEM SOLUTION APPROACH

1.st Approach

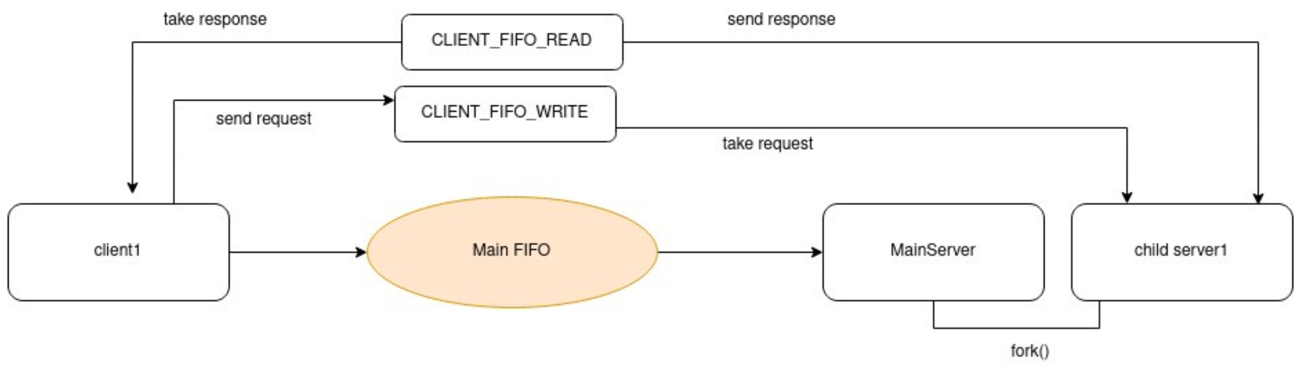


In this midterm exam, I tried two different approaches, the first approach was as I tried to explain in the graphic, first a client is created, then this created client communicates with the main server over MainFifo then the main server creates its own child by fork() and after this child is created, a new fifo file is opened which depends on client_fifo_pid All communication was intended to be received here.

In order to read and write from the same file, it was necessary to operate in the same file descriptor, so in some cases the server would read again what the server had written, or in some cases it would read what the client wrote again. For this reason, I thought of using a semaphore. First of all, I thought of using a single semaphore, but I tried to use two fifos because I could not control the writing of two fifo files.

I used semaphores in these code blocks so that the client does not read the request sent to the server again, and the written code blocks, and the server does not read the server's response again.I was not successful in the process here, and I tried my second approach.

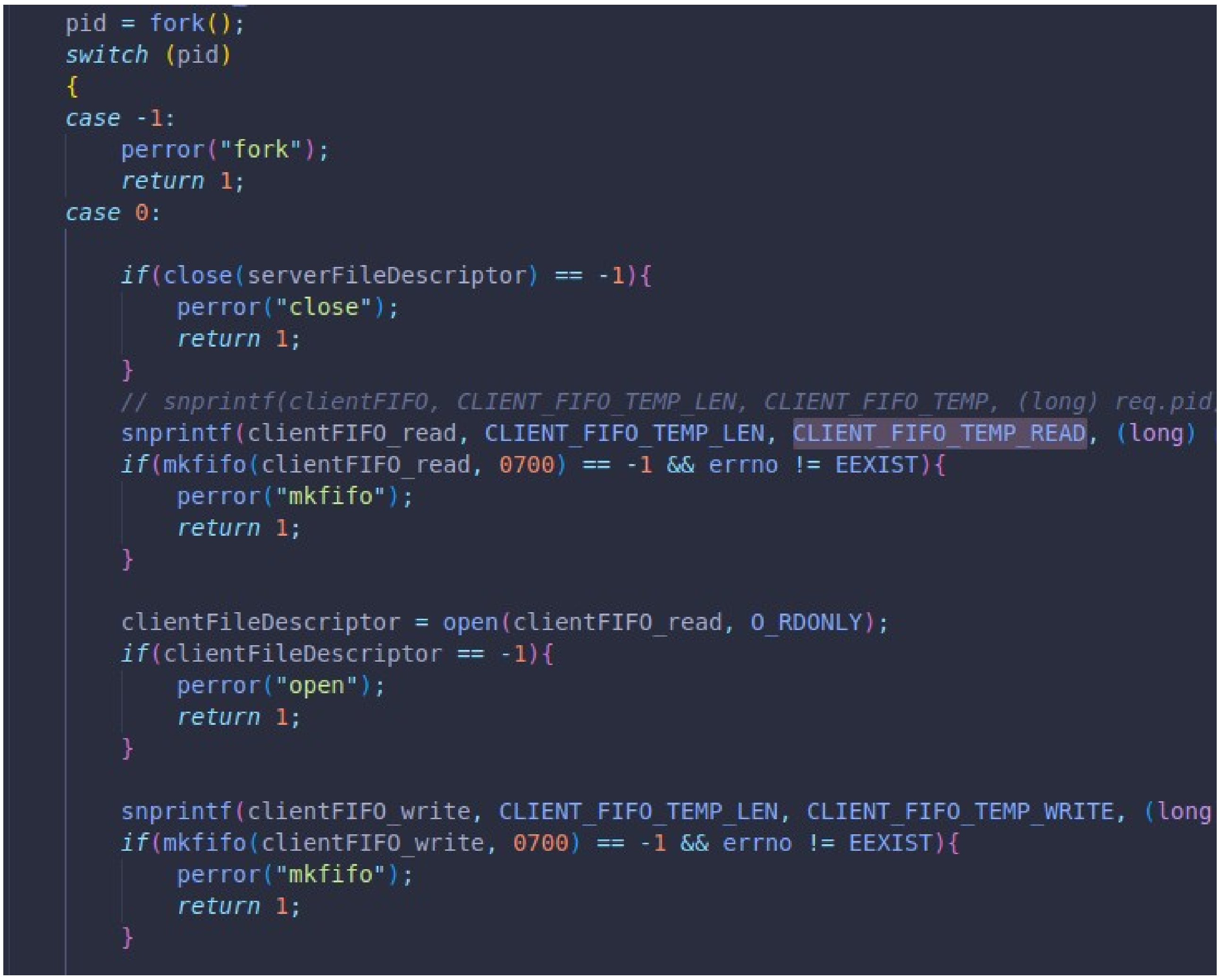
2nd Approach



The Second approach was as I tried to explain in the graphic, first a client is created, then this created client communicates with the main server over MainFifo then the main server creates its own child by fork() and after this child is created, a new 2 fifo file one is read from the client and other one is writes.All read and write fifos opened which depends on client_fifo_pid All communication was intended to be received in these two file.

In the first approach, after I failed in two-way communication over a single file, I aimed to establish one-way communication by opening two communication files.

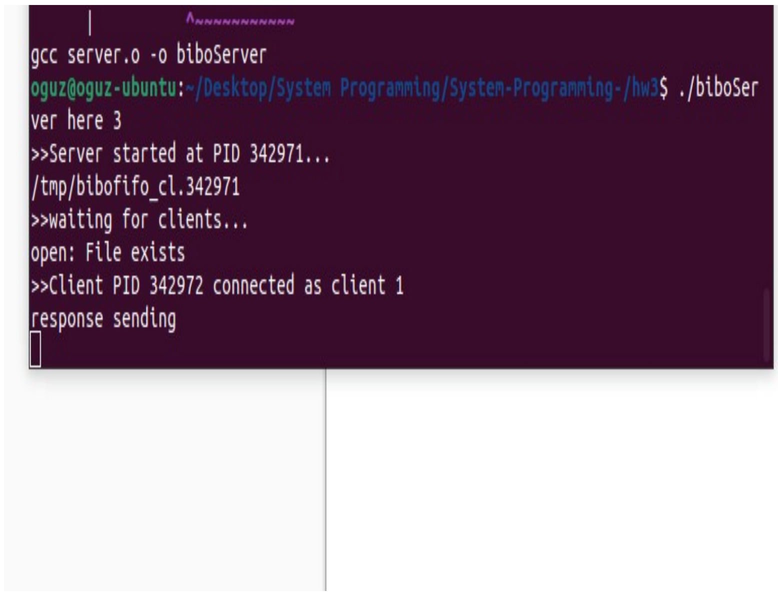
After the client connection is established, it sends a request to the server with CLIENT_FIFO_WRITE, the server reads this request from the same file and the server sends a response to the file that the client will read



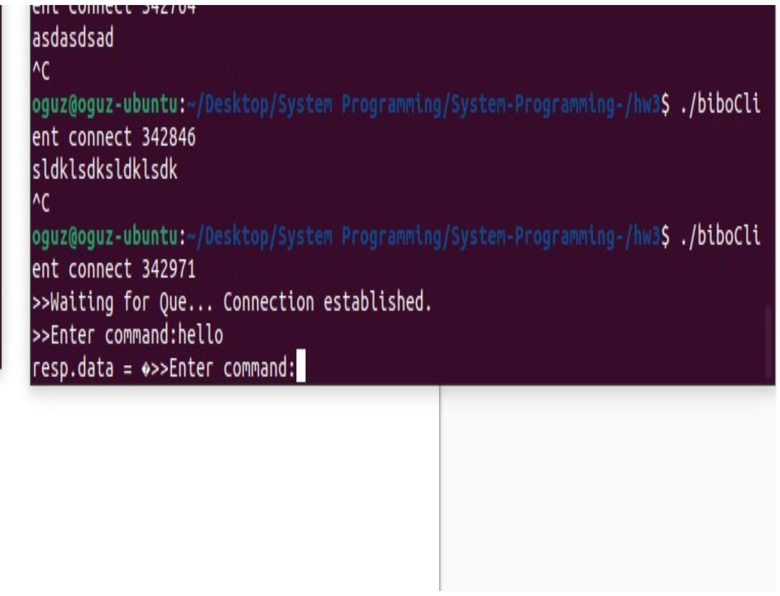
```
    pid = fork();
    switch (pid)
    {
    case -1:
        perror("fork");
        return 1;
    case 0:

        if(close(serverFileDescriptor) == -1){
            perror("close");
            return 1;
        }
        // snprintf(clientFIFO, CLIENT_FIFO_TEMP_LEN, CLIENT_FIFO_TEMP, (long) req.pid,
        snprintf(clientFIFO_read, CLIENT_FIFO_TEMP_LEN, CLIENT_FIFO_TEMP_READ, (long)
        if(mkfifo(clientFIFO_read, 0700) == -1 && errno != EEXIST){
            perror("mkfifo");
            return 1;
        }

        clientFileDescriptor = open(clientFIFO_read, O_RDONLY);
        if(clientFileDescriptor == -1){
            perror("open");
            return 1;
        }

        snprintf(clientFIFO_write, CLIENT_FIFO_TEMP_LEN, CLIENT_FIFO_TEMP_WRITE, (long
        if(mkfifo(clientFIFO_write, 0700) == -1 && errno != EEXIST){
            perror("mkfifo");
            return 1;
        }
```

clintFIFO_read => response to the client
clintFIFO_write => response to the write

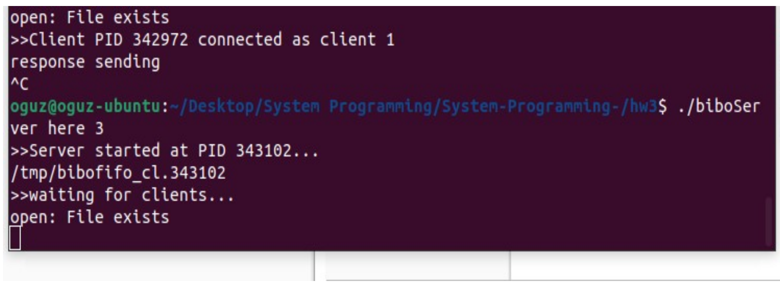
3.TEST CASES AND RESULTS



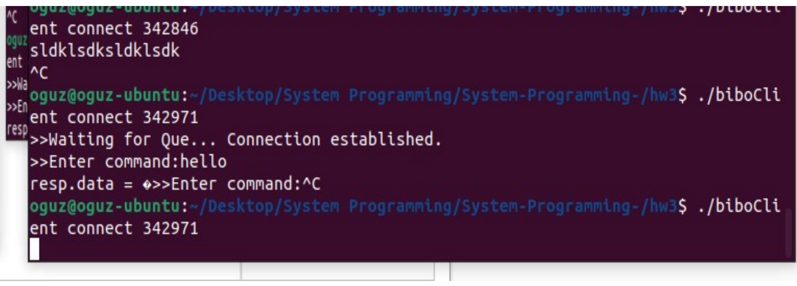
```
     |          ^~~~~~~~~~~
gcc server.o -o biboServer
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboSer
ver here 3
>>Server started at PID 342971...
/tmp/bibofifo_cl.342971
>>waiting for clients...
open: File exists
>>Client PID 342972 connected as client 1
response sending
```



```
ent connect 342764
asdasdsad
^C
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboCli
ent connect 342846
sldklsdksldklsdk
^C
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboCli
ent connect 342971
>>Waiting for Que... Connection established.
>>Enter command:hello
resp.data = ◆>>Enter command:
```



```
open: File exists
>>Client PID 342972 connected as client 1
response sending
^C
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboSer
ver here 3
>>Server started at PID 343102...
/tmp/bibofifo_cl.343102
>>waiting for clients...
open: File exists
```



```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboCli
ent connect 342846
sldklsdksldklsdk
^C
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboCli
ent connect 342971
>>Waiting for Que... Connection established.
>>Enter command:hello
resp.data = ◆>>Enter command:^C
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw3$ ./biboCli
ent connect 342971
```

4. CONCLUSION AND NOTES

I did not achieve to reach end of the midterm my results does not work according to the some badfile errors or unknown read errors.