

**CSE 344**  
**MIDTERM Report**  
**Berru Lafci**  
**1901042681**

## How to run: with makefile

Compile: make

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. sınıf/4. sınıf 2. donem/System Programming/midterm$ make
gcc server.c -lrt -lpthread -o neHosServer
gcc client.c -lrt -lpthread -o neHosClient
```

Run:

- Server:

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. sınıf/4. sınıf 2. donem/System Programming/midterm$ ./neHosServer heree 2
Server Started PID 114...
waiting for clients...
Client PID 115 connected as client1
|
```

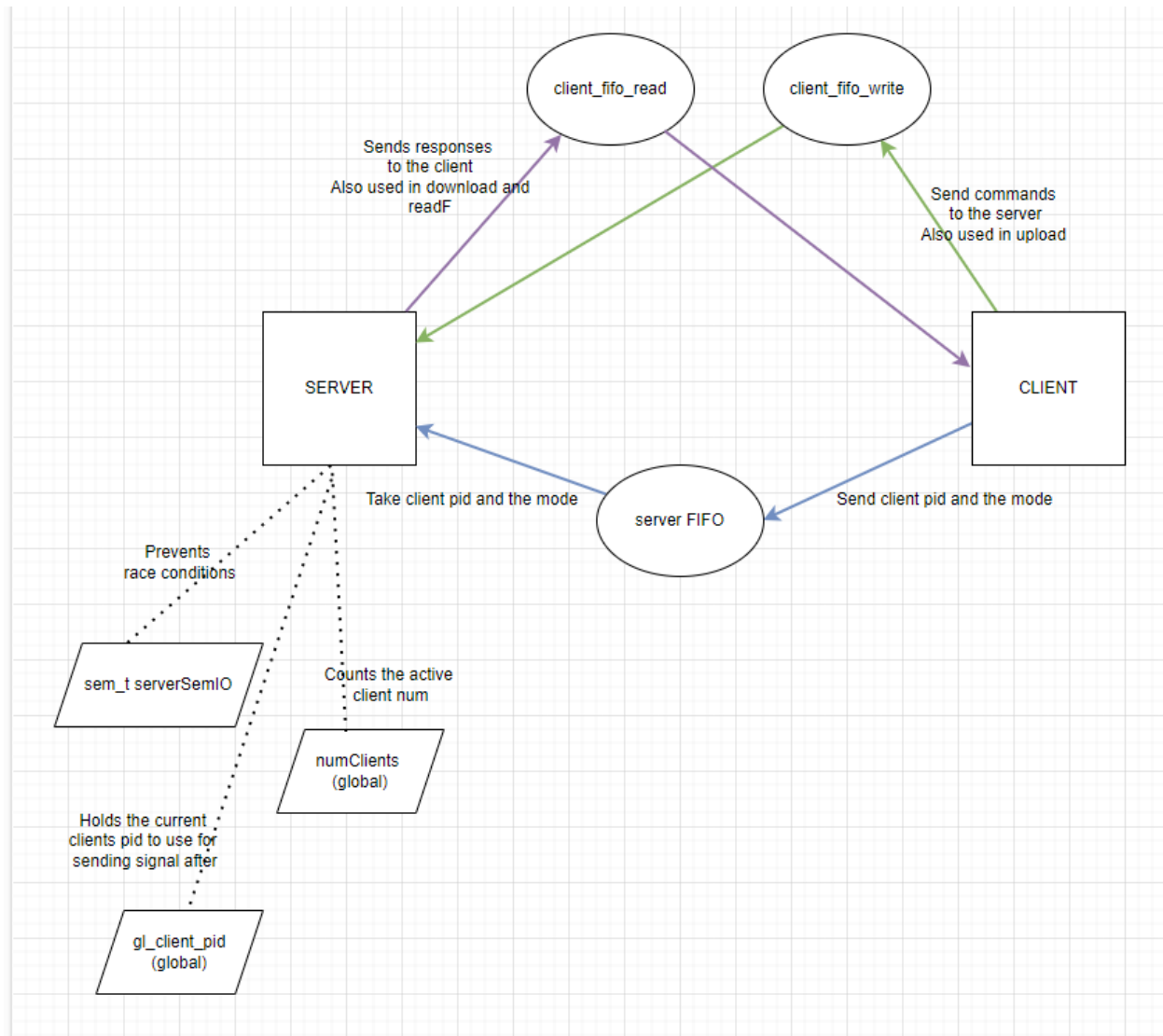
- Client:

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. sınıf/4. sınıf 2. donem/System Programming/midterm$ ./neHosClient tryconnect 114
Waiting for Que.. Connection Established...
Enter command: |
```

Clean: make clean

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. sınıf/4. sınıf 2. donem/System Programming/midterm$ make clean
rm -f neHosServer neHosClient
```

## Architecture:



## Design decisions and implementations:

### Server side:

- **Macro for Server FIFO:** `#define SERVER_FIFO "/tmp/server_fifo"` is absolute path (with /tmp) that the location of a file or directory specified from the root directory. So, there won't be any problem while trying the code in another computer.

### Global Variables:

- **int gl\_client\_pid:** Client PID to be used in signal handler. Because we can't reach the client PID from the child inside.
- **int numClients:** To count the connected active client number.
- **int client\_num:** To count the total client connected to write into terminal. This number always increases.

### Signal Handlers:

- **sigint\_handler:** On receiving **SIGINT**, it ensures all resources such as FIFOs are unlinked and child processes terminated, preventing resource leaks.

sigint\_handler sends kill signal to all child processes.

```
void sigint_handler(int signum)
{
    printf("Received SIGINT signal\n");

    // Unlink server FIFO
    unlink(SERVER_FIFO);

    // Kill all child processes
    int s;
    s = kill(0, SIGINT);
    if (s == -1)
    {
        perror("kill in sigint_handler");
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

- **sigint\_handler\_ch**: Sends kill signal to corresponding client pids so, the clients close smoothly. I handled its **sigaction** inside the child process (fork pid = 0) to handle from child.

```

9
0 void sigint_handler_ch(int signum)
1 {
2     printf("Received SIGINT signal in child process\n");
3
4     // Send kill signal to client
5     int s;
6     if(gl_client_pid > 0) {
7         s = kill(gl_client_pid, SIGINT);
8         if (s == -1)
9         {
10             perror("kill in sigint_handler_ch");
11             exit(EXIT_FAILURE);
12         }
13     }
14     exit(EXIT_SUCCESS);
15 }
16

```

```

if (pid == 0)
{
    // Sigint handler for child process
    struct sigaction sa_ch;
    sa_ch.sa_handler = sigint_handler_ch;
    sigemptyset(&sa_ch.sa_mask);
    sa_ch.sa_flags = 0;
    if (sigaction(SIGINT, &sa_ch, NULL) == -1)
    {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }
}

```

- **sigchld\_handler**: It decrements the **numClients** counter whenever a child process terminates, which is important for accurately tracking the number of active clients. It helps in maintaining the maximum client limit and ensuring new client connections can be accepted when there is capacity.

```

7 void sigchld_handler(int signum)
8 {
9     printf("Received SIGCHLD signal\n");
10
11     // Decrease number of clients
12     numClients--;
13 }
14

```

## Main Function

- With mkdir and chdir, I created and changed the directory of server with given argument.

```
mkdir(dirname, 0755);
chdir(dirname);
```

- Open log file, the I wrote to that file in some part of code like when server created, client connected or in success messages.

```
// Open log file
int log_fd = open("log.txt", O_WRONLY | O_CREAT, 0666);
if (log_fd < 0)
{
    perror("Failed to open log file");
    exit(EXIT_FAILURE);
}
```

- I created semaphore “serverSemIO” function to prevent **race condition** for file IO operations. This means when I try to read or write the same file at the same time in different clients, there may be confusion for the buffer. It opens the semaphore inside the function.

```
sem_t *serverSemIO = makeSem("serverSemIO", serverPID);
if(serverSemIO == NULL || serverSemIO < 0) {
    exit(EXIT_FAILURE);
}
```

- To use the semaphore, I used **sem\_wait** and **sem\_post**. Sem\_wait decreases the semaphore and if it is 0, it blocks it so no other client can enter its critical region. When it has done, it increases the semaphore so the other clients can enter their critical region to make their duty. Example usage:

```
// Execute upload
sem_wait(serverSemIO);
execute_upload(second_word, client_fifo_read, client_fifo_
sem_post(serverSemIO);
```

## Infinite Loop for Listening to Client Requests

I used a while loop to keep the server running and accepting clients.

- **Reading from the Server FIFO:**

The server attempts to read data from its FIFO (server\_fd) into a buffer (fifo\_message). The client writes the client pid and the mode (tryconnect or connect).

Once a message is successfully read, the server parses the FIFO message to extract the client's process ID (client\_pid) and a connection mode or command from the message using sscanf. This allows the server to understand who the client is and what initial command or mode they are requesting.

```
if (read(server_fd, fifo_message, 256) > 0)
{
    char *connect_mode = &fifo_message[32];
    sscanf(fifo_message, "%d%s", &client_pid, connect_mode);
}
```

- **I couldn't implement the "connect" mode. This is my implementation of "tryconnect" mode:**

The server checks if the current number of connected clients (numClients) is less than the maximum allowed (maxClients). If the max number reached, the server waits for the decrease of numClients (by quit or by sigint signal) and logs that its full. numClients increases in the parent process and decreases in if(quit) or in sigchld\_handler. *(There is a test run for this at the continuation of this report.)*

```
void sigchld_handler(int signum)
{
    printf("Received SIGCHLD signal\n");

    // Decrease number of clients
    numClients--;
}
```

```

else {
    if (strcmp(first_word, "quit") == 0)
    {
        printf("Client %d disconnected\n", client_pid);

        // Write client disconnected message to log file
        char *client_disconnected_log = (char *)malloc(
            sizeof(char) * 1024);
        sprintf(client_disconnected_log, "Client %d disconnected\n", client_pid);
        if(write(log_fd, client_disconnected_log, strlen(client_disconnected_log)) < 0)
            perror("Failed to write to log file");
        exit(EXIT_FAILURE);
    }
    free(client_disconnected_log);
    numClients--;
}

```

```

else
{
    // Parent process
    printf("Client PID %d connected as client%d\n", client_pid, client_num);
    numClients++;
}

```

- **Forking a Process for Each Client:**

If *numClients* < *maxClients*, the server forks a new process using `pid = fork()`. This child process will handle the specific client, allowing the parent process to return to listening for other clients.

- **Child Process:**

Sets up its own signal handlers, particularly for SIGINT to handle interruptions specifically for the child process.

Prepares client-specific FIFOs for bidirectional communication, formats them using the client's PID, and then calls `handle_client` to manage all interactions with this client.

**Client\_fifo\_read** used for server write and client read.

**Client\_fifo\_write** used for server read and client write messages.

```

char client_fifo_read[32];
char client_fifo_write[32];
sprintf(client_fifo_read, "/tmp/client_fifo_read_%d", client_pid);
sprintf(client_fifo_write, "/tmp/client_fifo_write_%d", client_pid);

```

Terminates after handling the client by calling `exit(EXIT_SUCCESS)`.

- **Parent Process:** Logs the connection and continues to monitor the FIFO for new connections.



- **Logging and Error Handling:** In both child and parent contexts, any errors during operations (like writing to log files) are checked, and appropriate error handling is executed, including potentially terminating the process if a critical operation fails.
- **Resource Management:** Throughout the loop, resources such as dynamically allocated memory for logs are managed carefully (allocated and freed within the same loop iteration), preventing memory leaks.

## handle\_client function:

```
void handle_client(char *client_fifo_read, char *client_fifo_write, int client_pid, int log_fd, char *dirname, sem_t *serverSemIO, int forked_pid, int client_num)
{
```

I read the message (command) coming from client in while loop.

```
char message[256];
while (1)
{
    // Read message from client
    client_fd_write = open(client_fifo_write, O_RDONLY);
    if (client_fd_write < 0)
    {
        perror("Failed to open client FIFO read");
        exit(EXIT_FAILURE);
    }
    if (read(client_fd_write, message, 256) > 0)
    {
        printf("Client %d: %s\n", client_pid, message);
    }
}
```

Then I parsed the message using strchr. I parsed firstly the first, second and remaining words.

```
char *space = strchr(message, ' ');
if (space == NULL)
```

By looking the first word, I went into corresponding commands:

- I first check the **upload, download and readF** commands because I made **error handling** for non-existing filename for them.

```

// If upload, download or readF command is received, check if file exists
if(strcmp(first_word, "upload") == 0) { ...
else if(strcmp(first_word, "download") == 0) { ...
else if(strcmp(first_word, "readF") == 0) { ...
else {
    if (strcmp(first_word, "quit") == 0)
    {

```

- For upload, I check the file exists in client.c and for download and readF I check the file in server.c because of their corresponding directory places. I simply checked it with access function:

```

else if(strcmp(first_word, "download") == 0) {
    // Check if file exists in server directory
    if(access(second_word, F_OK) == -1) {
        printf("File does not exist\n");
    }
}

```

- **If there isn't any file with the given name**, I send a message to client fifo as **"error"**. And then I wrote the situation to log file. Then I continued (with "continue") to the code where is the beginning of the while loop again.

```

sprintf(response, "error");
if(write(client_fd_read, response, 64) < 0) {
    perror("Failed to write to client FIFO");
}

```

- **If there is a file with given name**, I send a message to client fifo as **"success"**. Then I wrote that in log file. Then I called the **execute\_download** function.
- In client side, I simply read the fifo error or success message and continued the code (with "continue") in error or handled the function in success according to the message.

*The download and readF logic are the same in server. The upload logic is the exact opposite of them.*

Then I implemented the other commands.

## Client side:

- **sigint\_handler**: On receiving **SIGINT**, it ensures all resources such as FIFOs are unlinked and the client exited.

```

void sigint_handler() {
    printf("SIGINT received. Terminating client.\n");

    char client_fifo_read[32];
    char client_fifo_write[32];
    int client_pid = getpid();
    sprintf(client_fifo_read, "/tmp/client_fifo_read_%d", client_pid);
    sprintf(client_fifo_write, "/tmp/client_fifo_write_%d", client_pid);

    unlink(client_fifo_read);
    unlink(client_fifo_write);

    exit(EXIT_SUCCESS);
}

```

- I wrote the given command from user to server fifo.

```

if(write(server_fd, pid_msg, 256) < 0) {
    perror("Failed to write to server FIFO");
}

```

- I made 2 client fifo as **client\_fifo\_read** and **client\_fifo\_write**. One of them reads from the server and the other one is writes to the server.

```

if(mkfifo(client_fifo_read, 0666) < 0) {
    if(errno != EEXIST) {
        perror("Failed to create client FIFO read");
        exit(EXIT_FAILURE);
    }
}

if(mkfifo(client_fifo_write, 0666) < 0) {
    if(errno != EEXIST) {
        perror("Failed to create client FIFO write");
        exit(EXIT_FAILURE);
    }
}

```

- In while loop, I take the commands from user. Then I send the command to the server with fifo. If the command is quit, it breaks the loop.

```

// Send command to server
client_fd_write = open(client_fifo_write, O_WRONLY);
if(client_fd_write < 0) {
    perror("Failed to open client FIFO write");
}

```

- Then I parse the command with strchr (same with the server side). Again, I first checked upload, download and readF similar with server side because of the file name existing error.

Then I implemented the commands.

## Commands implementations and runs:

**Help:** prints the commands string.

```
Enter command: help
help, list, readF, writeT, upload, download, archServer, quit, killServer
Enter command: |
```

```
Enter command: help readF
readF <file> <line #>
display the #th line of the <file>, returns with an error if <file> does not exists
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 Servers directory creates and edits
the file at the same time
Enter command: |
```

**List:** prints the files in the server directory

```
Enter command: list
aaa.txt
ahahh
bb.txt
buraya.tar.gz
fff
hahha
hhah
img.jpg
log.txt
nana.png
sss.jph
top.tar.tar.gz
uuu.txt
Enter command: |
```

```
▼ heree
  ≡ aaa.txt
  ≡ ahahh
  ≡ bb.txt
  ≡ buraya.tar.gz
  ≡ fff
  ≡ hahha
  ≡ hhah
  ≡ img.jpg
  ≡ log.txt
  ≡ nana.png
  ≡ sss.jph
  ≡ top.tar.tar.gz
  ≡ uuu.txt
```

## readF:

In server,

- **If line number is not given** it reads the file line by line and find the file size with increasing realloc. It also arranges the file content with realloc with adding the contents continuously with strcat. Then it writes the size and contents to the client fifo.

```
if (write(client_fd_read, &file_size, sizeof(int)) < 0)
{
    perror("Failed to write file size to client FIFO");
    exit(EXIT_FAILURE);
}

if (write(client_fd_read, file_contents, file_size) < 0)
{
    perror("Failed to write file contents to client FIFO");
    exit(EXIT_FAILURE);
}
```

- **If the line number is given**, it reads the file line by line. When it comes to the given line number, it breaks the loop. Then it writes the line to the client fifo.

```
if (write(client_fd_read, line, 256) < 0)
{
    perror("Failed to write to client FIFO");
    exit(EXIT_FAILURE);
}
```

In client, if line number is not given, it reads 2 times from client fifo. If the line number is given it reads 1 times from client.

```
if(read(client_fd_read, line, 256) > 0) {
    printf("Server: %s\n", line);
}
```

Enter command: readF denemee.txt

File Content:

aaaa

bbbbbbbbbb

qqqqqqqqqq

xxxxxx

cccccccccc

dörtttt

Enter command: readF denemee.txt 3

Server: qqqqqqqqqq

Enter command: |

heree > denemee.txt

1 aaaa

2 bbbbbbbbbb

3 qqqqqqqqqq

4 xxxxxx

5 cccccccccc

6 dörtttt

7

## writeT:

In server,

**If line number is not given**, it opens the file in append mode or creates it if it doesn't exist, writes the provided string followed by a newline character, and then closes the file.

**If line number is given**, it reads the file's contents, seeks to the specified line number, inserts the new string, and then writes the modified contents back to the file.

```
// Write the file contents back to the file until the line number
int j = 0;
while(j < i) {
    if(write(file_fd, &file_contents[j], 1) < 0) {
        perror("Failed to write to file");
        exit(EXIT_FAILURE);
    }
    j++;
}
// Write the new line
if(write(file_fd, "\n", 1) < 0) {
    perror("Failed to write to file");
    exit(EXIT_FAILURE);
}
// Write the new string
if(write(file_fd, remaining, strlen(remaining)) < 0) {
    perror("Failed to write to file");
    exit(EXIT_FAILURE);
}

// Write the rest of the file contents back to the file
j = i;
while(j < bytes_read) {
    if(write(file_fd, &file_contents[j], 1) < 0) {
        perror("Failed to write to file");
        exit(EXIT_FAILURE);
    }
    j++;
}
```

```
Enter command: writeT denemee.txt dörtttt
Enter command: writeT denemee.txt 3 qqqqqqqqq
Enter command: |
```

heree > denemee.txt

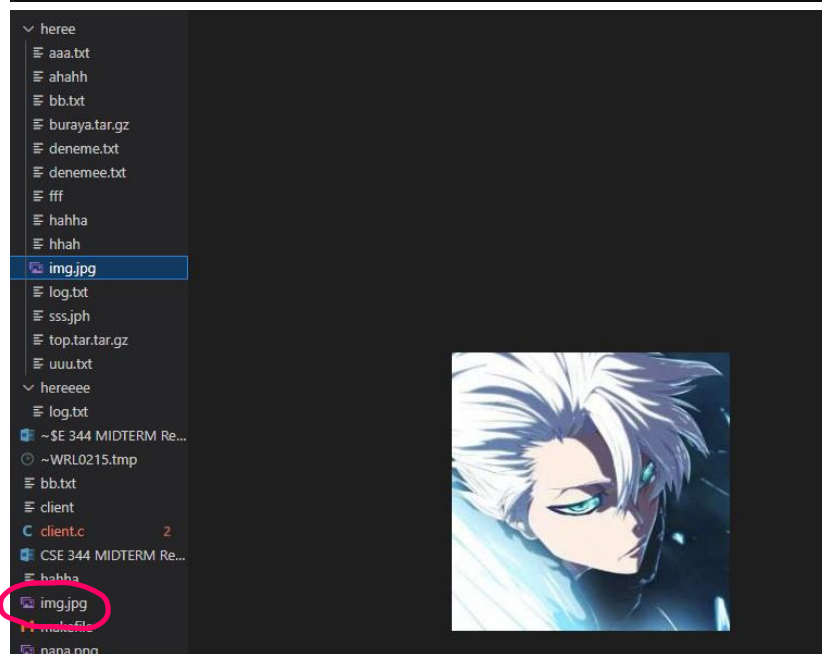
```
1 aaaa
2 bbbbbbbbbb
3 qqqqqqqqqq
4 xxxxxx
5 cccccccccc
6 dörtttt
7
```

## Upload:

The **execute\_upload** function on server side handles the uploading of a file from a client to the server. It receives the file name, client FIFO read and write paths, server directory name, and client file descriptor for writing. It reads the file from the client FIFO, writes it to the server directory, and notifies the client of success or failure.

The **handle\_upload** function on client side is responsible for handling the upload request from the client. It receives the file name, client file descriptor for writing, and client FIFO write path. It reads the file content, sends it to the server through the client FIFO, and closes the client FIFO afterward.

```
Enter command: upload img.jpg
file transfer request received. Beginning file transfer:
13063 bytes transferred
Enter command: |
```

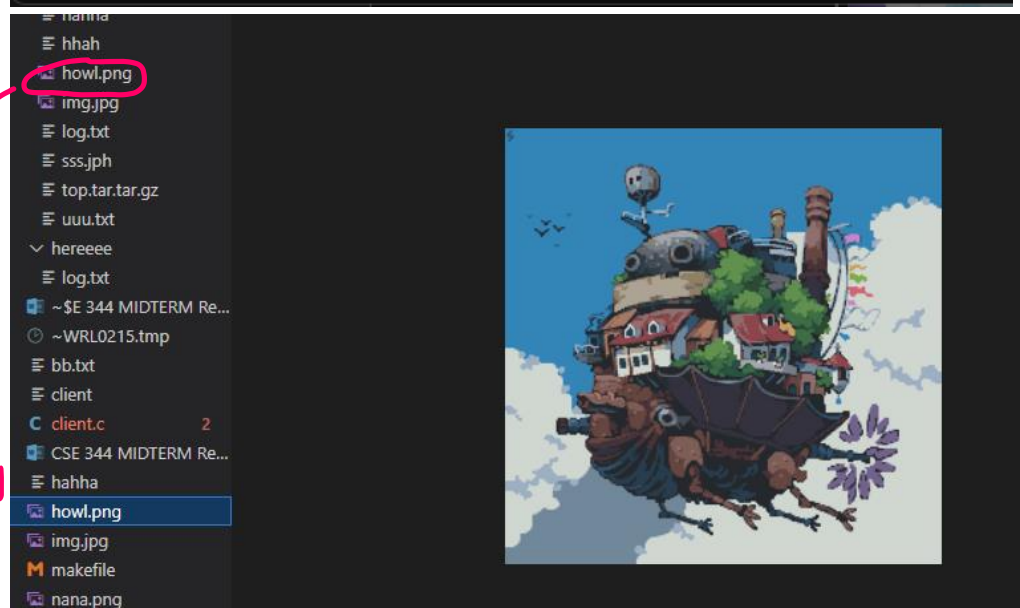


## Download:

The **execute\_download** function in server side is responsible for downloading a file from the server's directory to the client's current working directory. It takes the filename, client FIFO read and write paths, server directory name, and file descriptors as arguments. It opens the file for reading, retrieves its size, sends the size to the client, reads the file contents, and then writes them to the client.

The **handle\_download** function in client side is invoked to handle the download request from the client. It receives the filename and client FIFO paths as arguments. It opens a new file for writing, reads the file size and content from the client, and writes the content to the newly created file. Finally, it closes the file and client FIFO.

```
Enter command: download howl.png
file download request received. Beginning file download:
32068 bytes transferred
Enter command: |
```





## archServer:

It creates a tar archive of files in the server directory. It then extracts the filename from the received command and forks a child process to execute the tar command, compressing the files into a tar.gz archive.

```
if(pid == 0) {  
    // Child process  
    pid_t pid2 = fork();  
    if(pid2 < 0) {  
        perror("Failed to fork");  
        exit(EXIT_FAILURE);  
    }  
    else if(pid2 == 0) {  
        char *tarBuffer = (char *)malloc(256);  
        sprintf(tarBuffer, "tar -czf /tmp/%s.tar.gz .", filename);  
  
        int ret = execl("/bin/sh", "sh", "-c", tarBuffer, (char *)0);  
        if(ret < 0) {
```

The child process waits for the tar operation to complete and checks for success.

```
Enter command: list  
aaa.txt  
ahahh  
bb.txt  
deneme.txt  
denemee.txt  
fff  
hahha  
hhah  
howl.png  
img.jpg  
log.txt  
sss.jph  
uuu.txt  
Enter command: archServer topl  
Successfully created tar archive topl.tar.gz
```

### inside topl.tar.gz:

Name	Type
aaa.txt	Text Document
ahahh	File
bb.txt	Text Document
deneme.txt	Text Document
denemee.txt	Text Document
fff	File
hahha	File
hhah	File
howl.png	PNG File
img.jpg	JPG File
log.txt	Text Document
sss.jph	JPH File
uuu.txt	Text Document

## killServer:

In server side, when I take the command from client, I send the kill signal to the childs.

```
int s;  
s = kill(0, SIGINT);  
if (s == -1)  
{  
    perror("kill inside killserver");  
    exit(EXIT_FAILURE);  
}
```

It firstly sends kill signal to the childs. Then because of there is sigchld\_handler, the corresponding signal for clients successfully reaches them.

```
Enter command: killServer
```

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/De
```

```
kill signal from client1.. terminating server
```

```
byeReceived SIGINT signal in child process
```

```
Received SIGINT signal
```

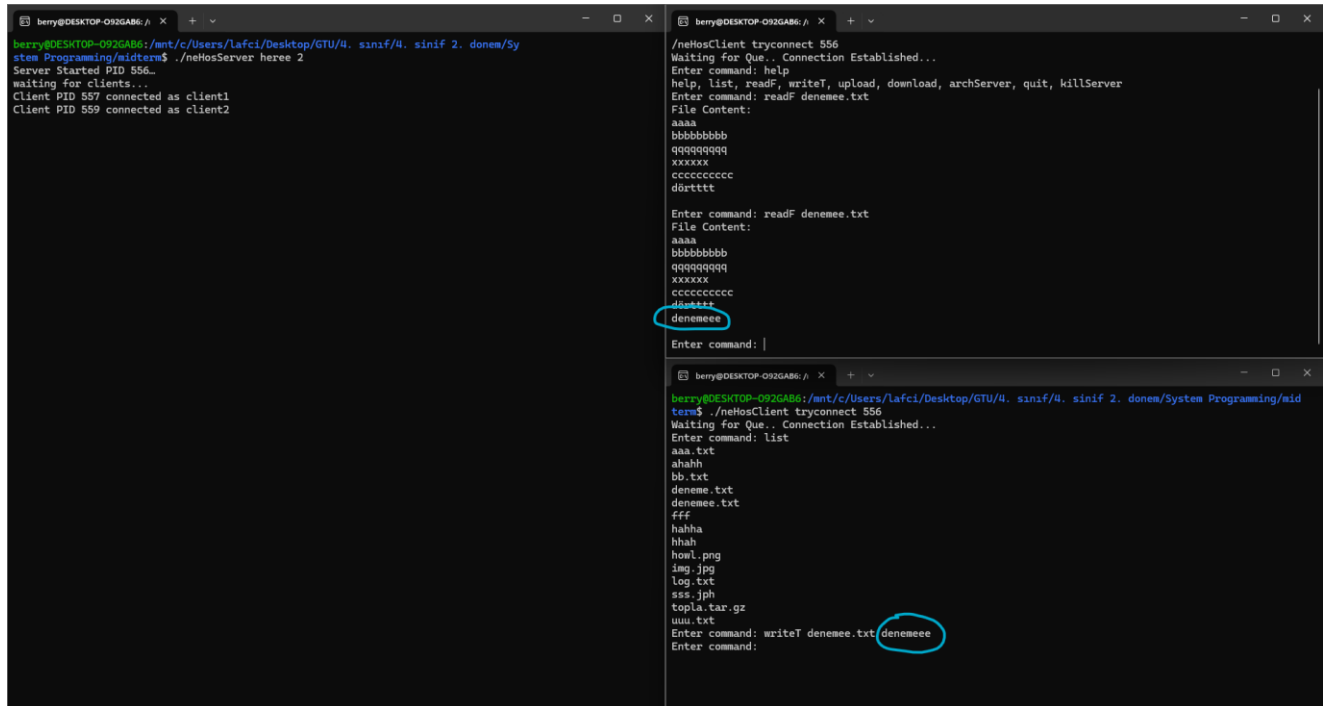
```
kill in sigint_handler_ch: No such process
```

```
Received SIGCHLD signal
```

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/
```

## Test plan and runs:

### 1- readF, writeT from different clients



The screenshot displays three terminal windows. The leftmost window shows the server's startup and client connections. The middle window shows a client performing a readF operation, displaying file content. The rightmost window shows a client performing a writeT operation, with the filename 'denemeee' circled in blue.

```
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556...
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2

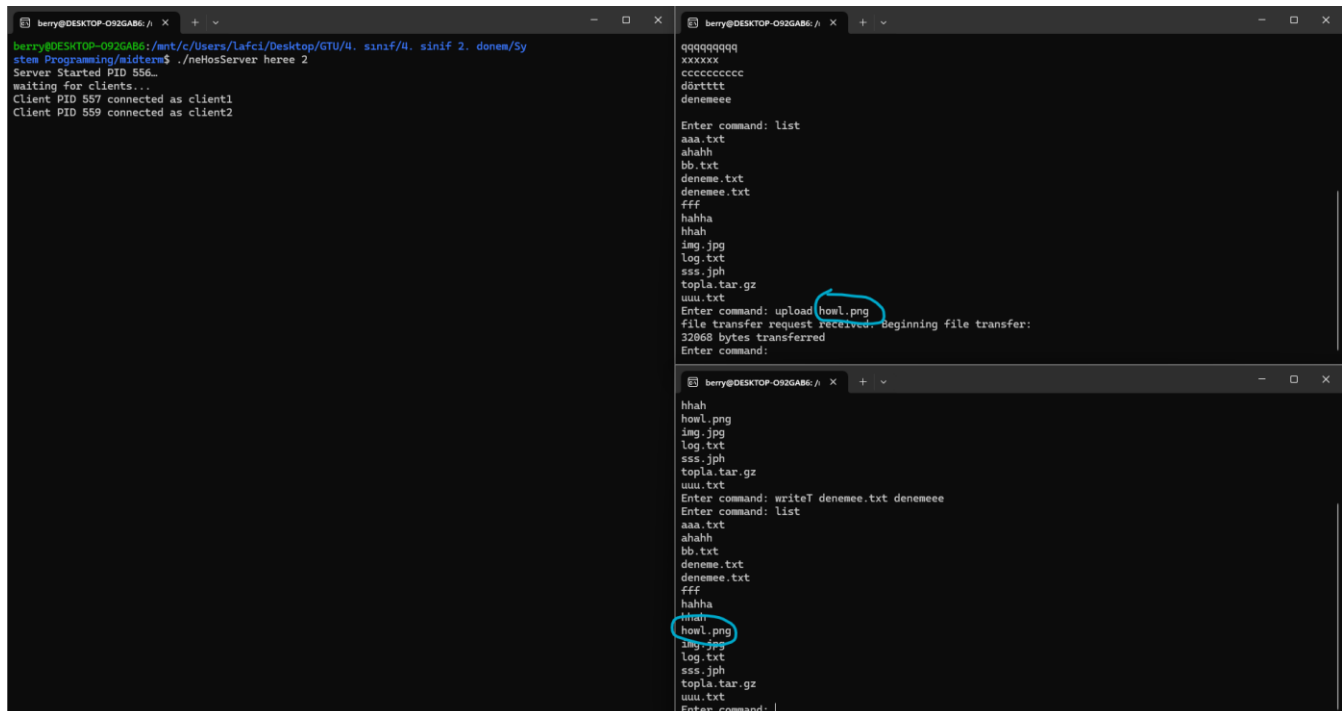
/nHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: help
help, list, readF, writeT, upload, download, archServer, quit, killServer
Enter command: readF denemeee.txt
File Content:
aaaa
bbbbbbbbbb
qqqqqqqqq
xxxxxx
ccccccccc
dortttt

Enter command: readF denemeee.txt
File Content:
aaaa
bbbbbbbbbb
qqqqqqqqq
xxxxxx
ccccccccc
denemeee

Enter command: |

berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: list
aaa.txt
ahahh
bb.txt
deneme.txt
denemeee.txt
fff
hahha
hhah
howl.png
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command: writeT denemeee.txt denemeee
Enter command:
```

### 2- upload shown in different clients



The screenshot displays three terminal windows. The leftmost window shows the server's startup and client connections. The middle window shows a client performing an upload operation, with the filename 'howl.png' circled in blue. The rightmost window shows a client performing a writeT operation, with the filename 'howl.png' circled in blue.

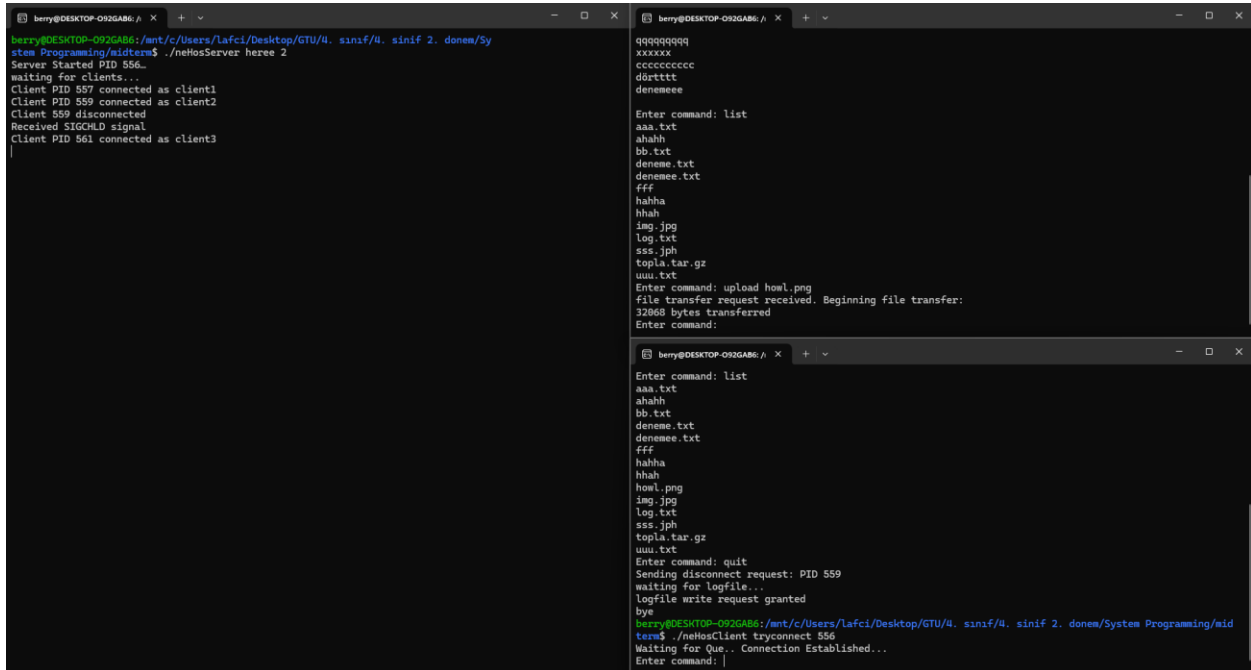
```
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556...
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2

qqqqqqqqq
xxxxxx
ccccccccc
dortttt
denemeee

Enter command: list
aaa.txt
ahahh
bb.txt
deneme.txt
denemeee.txt
fff
hahha
hhah
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command: upload howl.png
file transfer request received. Beginning file transfer:
32068 bytes transferred
Enter command:

berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: writeT denemeee.txt denemeee
Enter command: list
aaa.txt
ahahh
bb.txt
deneme.txt
denemeee.txt
fff
hahha
hhah
howl.png
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command:
```

### 3- send quit from client and then reconnect



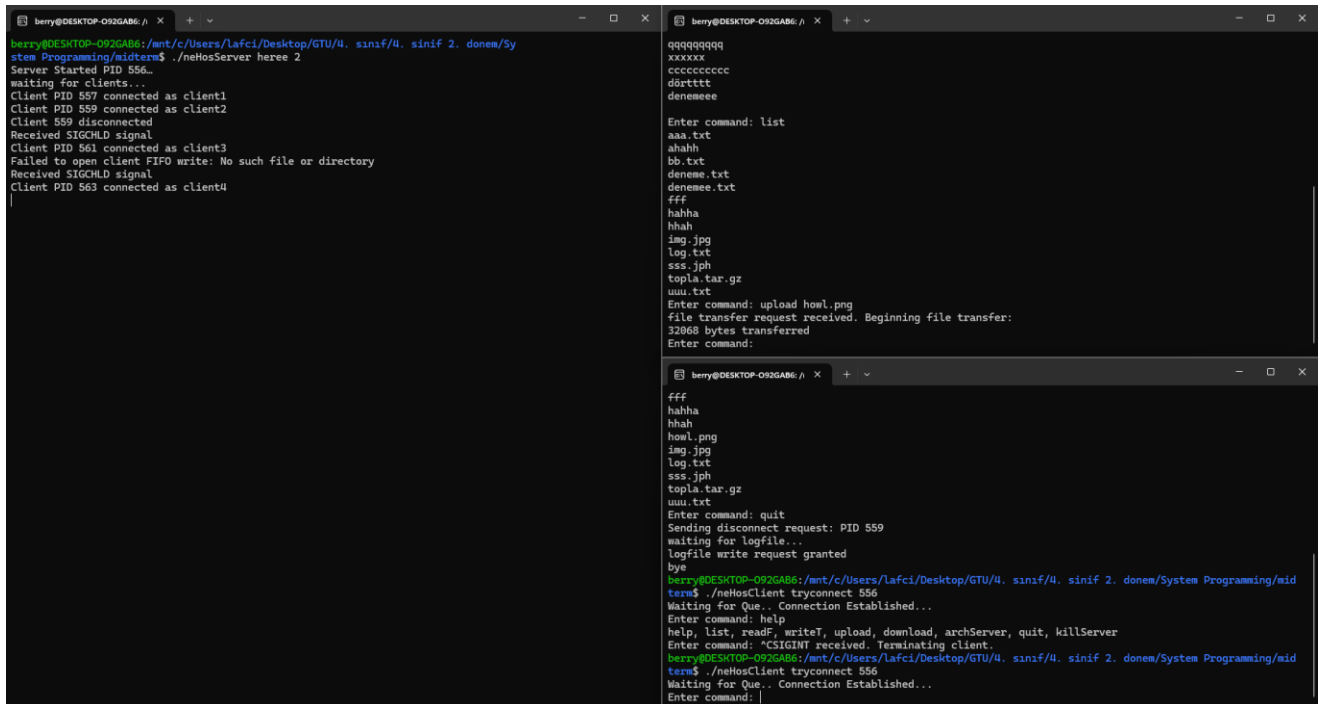
```
berry@DESKTOP-092GAB6:/mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556..
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2
Client PID 559 disconnected
Received SIGCHLD signal
Client PID 561 connected as client3
|

qqqqqqqqq
xxxxxx
ccccccccc
d0rtttt
denemeee

Enter command: List
aaa.txt
ahahh
bb.txt
deneme.txt
denemee.txt
fff
hahha
hhah
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command: upload howl.png
file transfer request received. Beginning file transfer:
32068 bytes transferred
Enter command:

berry@DESKTOP-092GAB6:/mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: |
```

### 4- send sigint signal to client and then reconnect



```
berry@DESKTOP-092GAB6:/mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556..
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2
Client PID 559 disconnected
Received SIGCHLD signal
Client PID 561 connected as client3
Failed to open client FIFO write: No such file or directory
Received SIGCHLD signal
Client PID 563 connected as client4
|

qqqqqqqqq
xxxxxx
ccccccccc
d0rtttt
denemeee

Enter command: List
aaa.txt
ahahh
bb.txt
deneme.txt
denemee.txt
fff
hahha
hhah
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command: upload howl.png
file transfer request received. Beginning file transfer:
32068 bytes transferred
Enter command:

fff
hahha
hhah
howl.png
img.jpg
log.txt
sss.jpg
topla.tar.gz
uuu.txt
Enter command: quit
Sending disconnect request: PID 559
waiting for logfile...
logfile write request granted
bye
berry@DESKTOP-092GAB6:/mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: help
help, list, readf, writef, upload, download, archServer, quit, killServer
Enter command: ^C SIGINT received. Terminating client.
berry@DESKTOP-092GAB6:/mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: |
```

## 5- que full in tryconnect, then quitting another client and connect again

- The server checks if the current number of connected clients (numClients) is less than the maximum allowed (maxClients). If the max number reached, the server waits for the decrease of numClients (by quit or by sigint signal) and logs that its full. And by the sighld\_handler, it arranges the numClient count.

```
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556...
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2
Client 559 disconnected
Received SIGCHLD signal
Client PID 561 connected as client3
Failed to open client FIFO write: No such file or directory
Received SIGCHLD signal
Client PID 563 connected as client4
Connection request PID 579... Que FULL
|
```

```
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4
t tryconnect 556
Waiting for Que.. Connection Established...
Enter command: aaaa
|
```

```
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/Sy
stem Programming/midterm$ ./neHosServer heree 2
Server Started PID 556...
waiting for clients...
Client PID 557 connected as client1
Client PID 559 connected as client2
Client 559 disconnected
Received SIGCHLD signal
Client PID 561 connected as client3
Failed to open client FIFO write: No such file or directory
Received SIGCHLD signal
Client PID 563 connected as client4
Connection request PID 579... Que FULL
Received SIGCHLD signal
Client PID 581 connected as clients5
|
```

```
qqqqqqqq
xxxxxx
ccccccccc
dortttt
denemeee

Enter command: list
aaa.txt
ahahh
bb.txt
deneme.txt
deneme.txt
fff
hahha
hhah
img.jpg
log.txt
sas.jpg
topla.tar.gz
uuu.txt
Enter command: upload howl.png
file transfer request received. Beginning file transfer:
32868 bytes transferred
Enter command:

uuu.txt
Enter command: quit
Sending disconnect request: PID 559
waiting for logfile...
logfile write request granted
bye
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: help
help, list, readF, writeT, upload, download, archServer, quit, killServer
Enter command: ^CSIGINT received. Terminating client.
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: quit
Sending disconnect request: PID 563
waiting for logfile...
logfile write request granted
bye
berry@DESKTOP-092GAB6: /mnt/c/Users/lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 556
Waiting for Que.. Connection Established...
Enter command: |
```

Another example of full que:

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. sınıf/4. sınıf 2. donem/System Programming/midterm$ ./neHosServer heree 2
Server Started PID 458...
waiting for clients...
Client PID 473 connected as client1
Client PID 489 connected as client2
Connection request PID 505... Que FULL
Client 489 disconnected
Received SIGCHLD signal
Client PID 507 connected as client3
Connection request PID 523... Que FULL
Client 507 disconnected
Received SIGCHLD signal
Client 473 disconnected
Received SIGCHLD signal
Client PID 525 connected as client4
```

## 6- killServer and corresponding signals

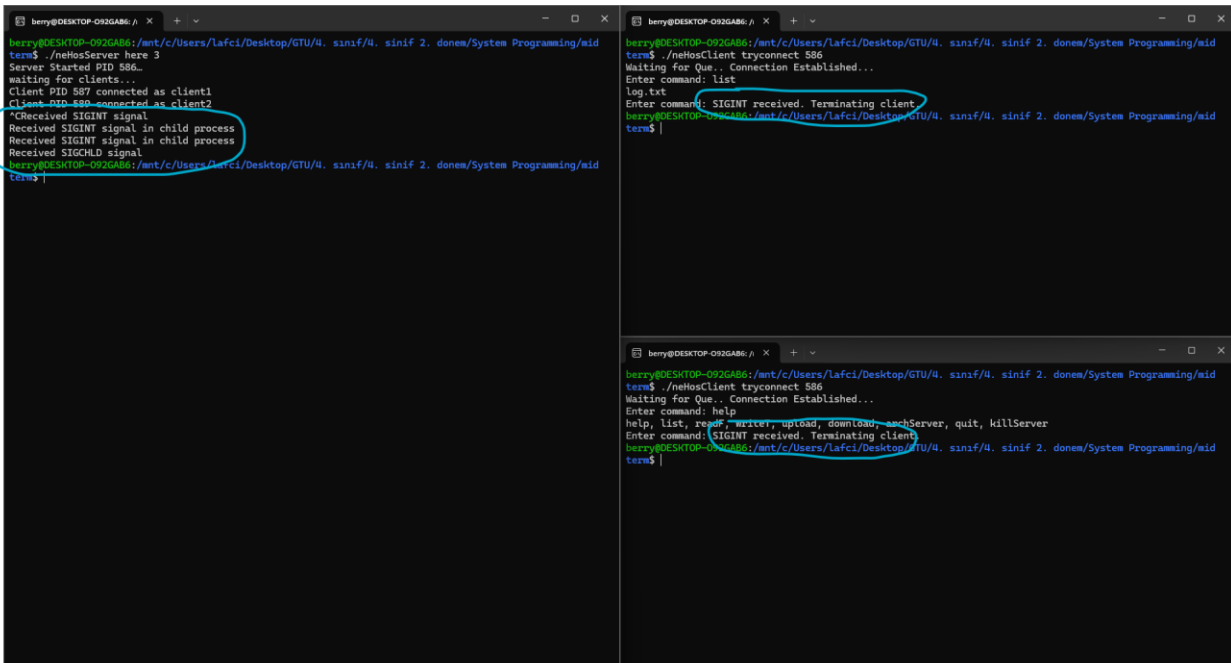
The screenshot displays three terminal windows illustrating the server's operation and signal handling:

- Left Window:** Shows the server's output. It starts with "Server Started PID 556..." and "waiting for clients...". It lists connections for client1 (PID 557), client2 (PID 559), and client3 (PID 561). It then shows a "Failed to open client FIFO write: No such file or directory" error. Subsequent connections for client4 (PID 563) and client5 (PID 565) are shown. A "kill signal from clients.. terminating server" message is highlighted with a red circle. Other messages include "Received SIGINT signal in child process" and "kill in sigint\_handler.ch: No such process".
- Top Right Window:** Shows a client's input and output. It lists commands: "list", "aaa.txt", "ahhhh", "bb.txt", "deneme.txt", "denemeee", "fff", "hahha", "hhah", "img.jpg", "log.txt", "sss.jpg", "topla.tar.gz", "uuu.txt". It then shows "Enter command: upload howl.png" and "file transfer request received. Beginning file transfer: 32868 bytes transferred". A "killServer" command is entered, and the output shows "Enter command: killServer" and "Terminating client".
- Bottom Right Window:** Shows the server's response to the "killServer" command. It displays "Sending disconnect request: PID 559", "waiting for logfile...", "logfile write request granted", and "bye". It then shows "Enter command: help", "help, list, readf, writef, upload, download, archServer, quit, killServer", "Enter command: ^CSIGINT received. Terminating client.", "Waiting for Que.. Connection Established...", "Enter command: quit", "Sending disconnect request: PID 563", "waiting for logfile...", "logfile write request granted", and "bye". Finally, it shows "Enter command: killServer" and "Terminating client".

The corresponding log file for these runs:

```
heree > ≡ log.txt
1  Server PID: 556
2  Client 557 connected.
3  Client 559 connected.
4  Client 557 successfully listed commands
5  Client 559 successfully listed files
6  Client 557 successfully read file denemee.txt
7  Client 557 successfully read file denemee.txt
8  Client 559 successfully wrote to file denemee.txt
9  Client 557 successfully read file denemee.txt
10 Client 557 successfully read file denemee.txt
11 Client 557 successfully listed files
12 Client 557 successfully uploaded file howl.png
13 Client 559 successfully listed files
14 Client 559 disconnected.
15 Client 561 connected.
16 Client 561 successfully listed commands
17 Client 563 connected.
18 Max number of clients reached.
19 Client 563 disconnected.
20 Client 581 connected.
21 Server killed.
...
22 Client 162 disconnected.
23
```

**7- sending sigint to server:** as you can see, it receives the sigint signal in child processes as the number of the clients.



```
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosServer here 3
Server Started PID 586.
waiting for clients...
Client PID 587 connected as client1
Client PID 589 connected as client2
*Received SIGINT signal
Received SIGINT signal in child process
Received SIGINT signal in child process
Received SIGCHLD signal
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$

berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 586
Waiting for Que... Connection Established...
Enter command: list
log.txt
Enter command: SIGINT received. Terminating client
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$

berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$ ./neHosClient tryconnect 586
Waiting for Que... Connection Established...
Enter command: help
help, list, read, writer, upload, download, askServer, quit, killServer
Enter command: SIGINT received. Terminating client
berry@DESKTOP-092GAB6: /mnt/c/Users/Lafci/Desktop/GTU/4. sinif/4. sinif 2. donem/System Programming/mid
term$
```

## Error Scenarios:

### Invalid command:

```
Enter command: hbhevrhbhrhbvre
Invalid command
Enter command:
```

### File not exist:

```
Enter command: readF bhahahbhbce
File does not exist on server
Enter command: upload hebhvhevrhje
File does not exist
Enter command: download hbevhvbher
File does not exist on server
Enter command: |
```

### Wrong arguments:

```
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. s1
term$ ./neHosServer here
Usage: ./neHosServer <dirname> <max.#ofClients>
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. s1
term$ |
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. s1
term$ ./neHosClient tryconnect
Usage: ./neHosClient <Connect/tryConnect> <ServerPID>
berry@DESKTOP-092GAB6:/mnt/c/Users/lafci/Desktop/GTU/4. s1
term$ |
```



