CSE344 SYSTEMS PROGRAMMING

MIDTERM

REPORT

Burak Demirkaya

210104004274

05/05/2024

1. Introduction

This project is a demonstration of a client-server model implementation using FIFOS (named-pipes) for communication between server and client side, semaphores for preventing racing conditions.

2. System Architecture

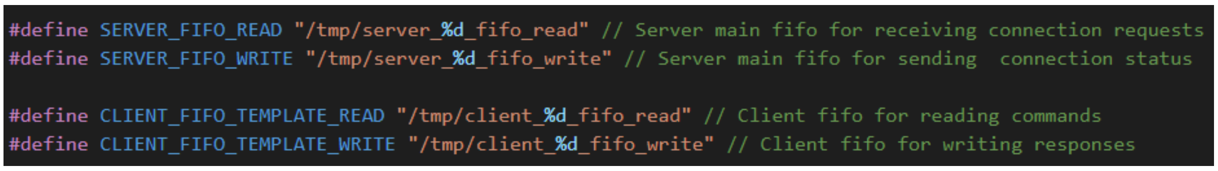
The system is built on a client-server model, where multiple clients can connect and interact with a single server. The server is designed to handle multiple clients concurrently, processing each client's requests in a separate child process. Communication between the client and server is facilitated through FIFOs (Named Pipes), and synchronization is achieved using semaphores and flocks.

3. Design Decisions

FIFOs were used for inter-process communication due to their simplicity and ease of use. They provide a straightforward way for the client and server to send data to each other. Semaphores were used to manage concurrency and ensure that the server can handle multiple clients without conflicts. Flocks were used to handle the access of the same file.

4. Implementation Details

The client and server are implemented as separate programs, each residing in its own directory. This separation of concerns allows for easier management and development of each component.



```c
#define SERVER_FIFO_READ "/tmp/server_%d_fifo_read" // Server main fifo for receiving connection requests
#define SERVER_FIFO_WRITE "/tmp/server_%d_fifo_write" // Server main fifo for sending  connection status

#define CLIENT_FIFO_TEMPLATE_READ "/tmp/client_%d_fifo_read" // Client fifo for reading commands
#define CLIENT_FIFO_TEMPLATE_WRITE "/tmp/client_%d_fifo_write" // Client fifo for writing responses
```

I have defined two different fifos for server and client. Server fifo read is for receiving the connection requests and the server fifo write is to send connection status such as “Full” if the server is full.

Client fifo read is for reading commands and the client fifo write is for sending the responses to the clients. Since the program uses multi-clients connection I have set a unique fifo for each client.

Server Side Implementation:

The server side is designed to handle multiple client connections and respond to various client commands. It achieves this through the use of FIFOs and signal handlers. Server side stores the pids and the file descriptors of the clients in an array to free them in the signal handlers.

Signal Handlers

Two signal handlers are defined on the server side.

1. The first signal handler is triggered when a SIGINT interrupt occurs or when any of the clients enter the command "killServer". This allows the server to gracefully shut down and clean up resources when it's being terminated.

2. The second signal handler is designed to handle child processes and prevent zombie processes. When a child process terminates, it sends a SIGCHLD signal to the parent process. This signal is caught by our signal handler, which then properly reaps the child process, preventing it from becoming a zombie process.

FIFOs

FIFOs are used to handle connections between the server and the clients. Each client connection is associated with a pair of FIFOs: one for the server to read from and one for the server to write to. These FIFOs are created using the client's PID, ensuring that each pair of FIFOs is unique to each client.

When a client sends a command to the server, it writes the command to the server's read FIFO. The server then reads the command from this FIFO, processes the command, and writes the response to the server's write FIFO. The client can then read the server's response from this FIFO.

This design allows the server to handle multiple client connections simultaneously, as each client has its own pair of FIFOs. It also ensures that the server can handle client commands and responses in a first-in, first-out manner, which is essential for maintaining the correct order of commands and responses.

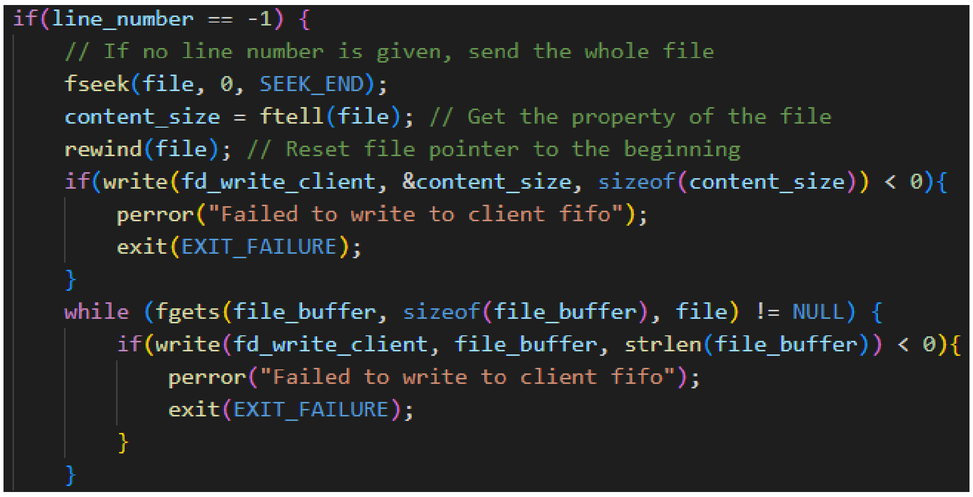
Server side handles the connection of the clients by using a connected_clients counter, if the connected_clients exceeds the maximum number of clients, depending on the command it handles the connection of the client that is trying to connect. After the connection has been established, the server side creates a child process to execute the client commands.

-list

List command lists every file in the server's "dirname" directory that has been created before and sends the name of files to the client side.

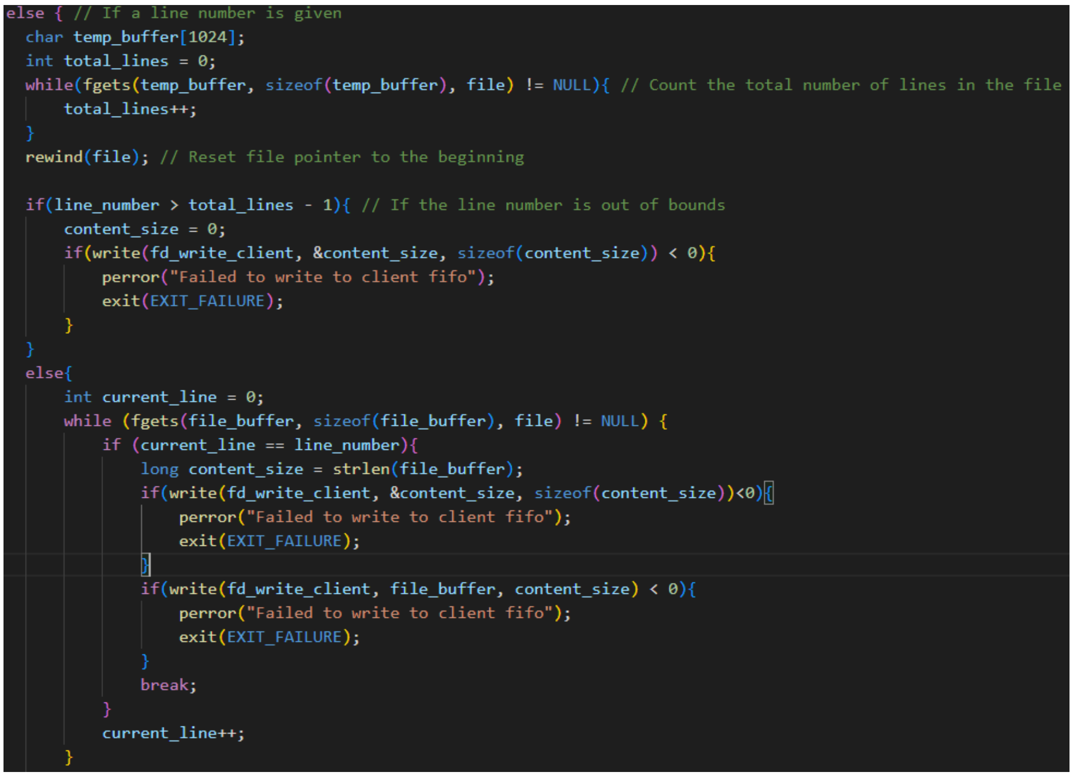
-readF <file> <line #>

Server side first tokenizes the command that has been entered by the client and depending on the values it performs the read operation on the file. Server side uses flock() function to ensure that other clients cannot read/write to the same file at the same time.



```
if(line_number == -1) {
    // If no line number is given, send the whole file
    fseek(file, 0, SEEK_END);
    content_size = ftell(file); // Get the property of the file
    rewind(file); // Reset file pointer to the beginning
    if(write(fd_write_client, &content_size, sizeof(content_size)) < 0){
        perror("Failed to write to client fifo");
        exit(EXIT_FAILURE);
    }
    while (fgets(file_buffer, sizeof(file_buffer), file) != NULL) {
        if(write(fd_write_client, file_buffer, strlen(file_buffer)) < 0){
            perror("Failed to write to client fifo");
            exit(EXIT_FAILURE);
        }
    }
```

If no line number is given it sends the whole file part by part to the client side with the fifo.



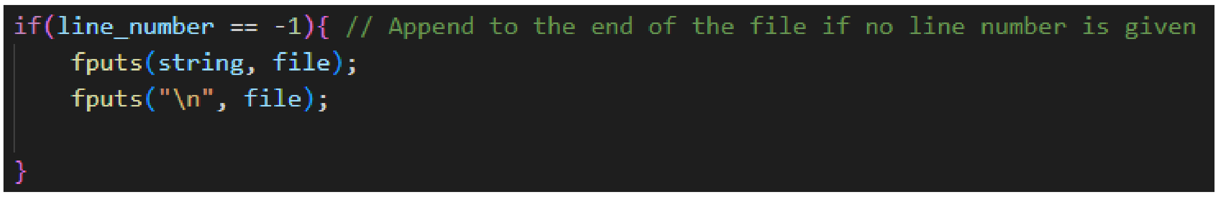
```c
else { // If a line number is given
    char temp_buffer[1024];
    int total_lines = 0;
    while(fgets(temp_buffer, sizeof(temp_buffer), file) != NULL){ // Count the total number of lines in the file
        total_lines++;
    }
    rewind(file); // Reset file pointer to the beginning

    if(line_number > total_lines - 1){ // If the line number is out of bounds
        content_size = 0;
        if(write(fd_write_client, &content_size, sizeof(content_size)) < 0){
            perror("Failed to write to client fifo");
            exit(EXIT_FAILURE);
        }
    }
    else{
        int current_line = 0;
        while (fgets(file_buffer, sizeof(file_buffer), file) != NULL) {
            if (current_line == line_number){
                long content_size = strlen(file_buffer);
                if(write(fd_write_client, &content_size, sizeof(content_size))<0){
                    perror("Failed to write to client fifo");
                    exit(EXIT_FAILURE);
                }
                if(write(fd_write_client, file_buffer, content_size) < 0){
                    perror("Failed to write to client fifo");
                    exit(EXIT_FAILURE);
                }
                break;
            }
            current_line++;
        }
```

If a line number is given, server side first calculates the maximum line of that file and depending on the input for the line number it returns the corresponding line. If the input for the line number exceeds the maximum line number for that file, it returns a string that the line number is out of bounds.

Finally server side unlocks the flock of that file and returns.

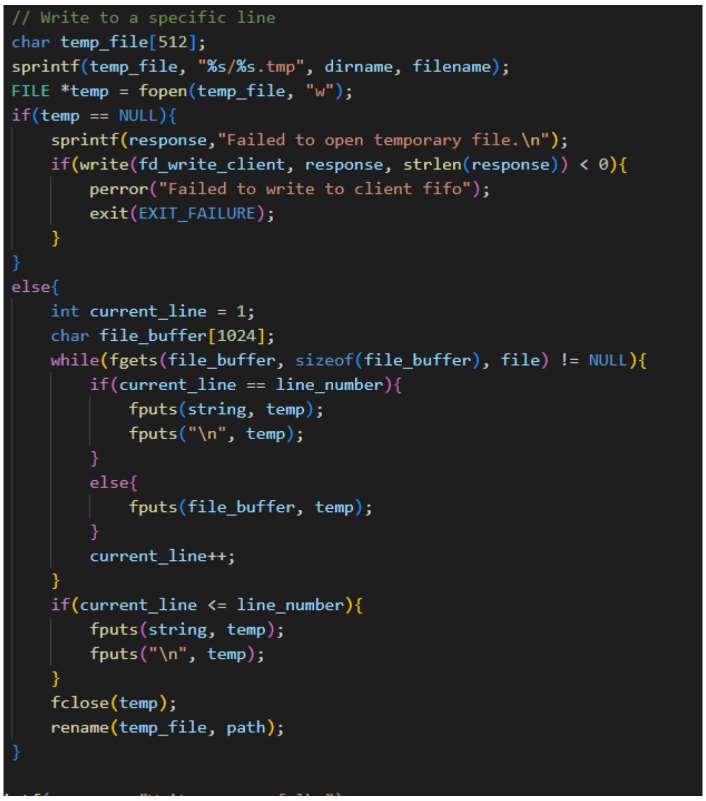
-writeT <file> <line #> <string>

Same thing as the readF command, server side tokenizes the command and depending on the values it writes to a specific line. If no line number is given it writes to the end of the file. Also it uses flock() function same as the readF command.



```c
if(line_number == -1){ // Append to the end of the file if no line number is given
    fputs(string, file);
    fputs("\n", file);

}
```

If no line number is given writes to the end of the file.

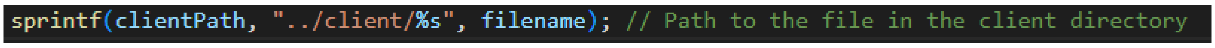


```c
// Write to a specific line
char temp_file[512];
sprintf(temp_file, "%s/%s.tmp", dirname, filename);
FILE *temp = fopen(temp_file, "w");
if(temp == NULL){
    sprintf(response,"Failed to open temporary file.\n");
    if(write(fd_write_client, response, strlen(response)) < 0){
        perror("Failed to write to client fifo");
        exit(EXIT_FAILURE);
    }
}
else{
    int current_line = 1;
    char file_buffer[1024];
    while(fgets(file_buffer, sizeof(file_buffer), file) != NULL){
        if(current_line == line_number){
            fputs(string, temp);
            fputs("\n", temp);
        }
        else{
            fputs(file_buffer, temp);
        }
        current_line++;
    }
    if(current_line <= line_number){
        fputs(string, temp);
        fputs("\n", temp);
    }
    fclose(temp);
    rename(temp_file, path);
}
```

If a line number is given it first creates a temporary file and writes to that file and renames the file to the original one.

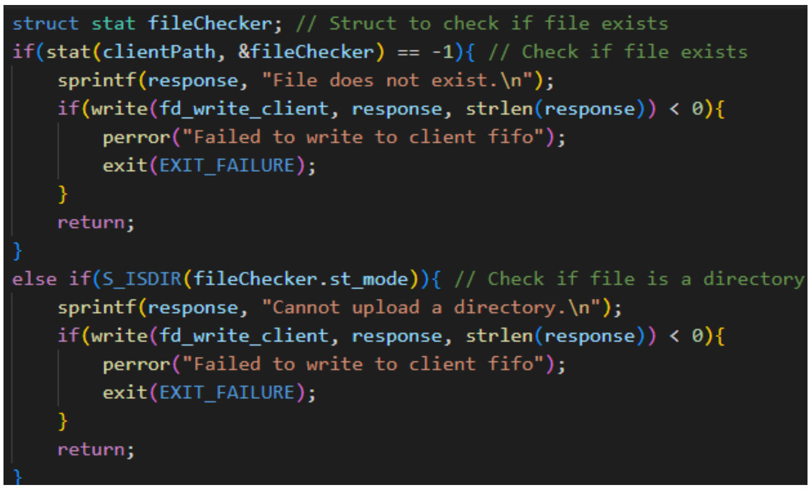
Finally it unlocks the flock and returns.

-upload <file>



```c
sprintf(clientPath, "../client/%s", filename); // Path to the file in the client directory
```

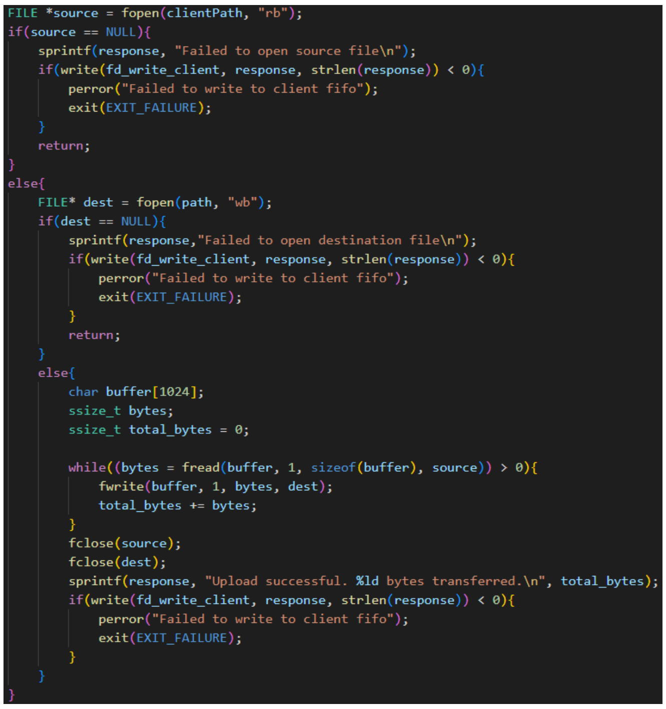
Since I have used two different directories for client and server I have defined the clientPath as this way.



```c
struct stat fileChecker; // Struct to check if file exists
if(stat(clientPath, &fileChecker) == -1){ // Check if file exists
    sprintf(response, "File does not exist.\n");
    if(write(fd_write_client, response, strlen(response)) < 0){
        perror("Failed to write to client fifo");
        exit(EXIT_FAILURE);
    }
    return;
}
else if(S_ISDIR(fileChecker.st_mode)){ // Check if file is a directory
    sprintf(response, "Cannot upload a directory.\n");
    if(write(fd_write_client, response, strlen(response)) < 0){
        perror("Failed to write to client fifo");
        exit(EXIT_FAILURE);
    }
    return;
}
```

Depending on the clientPath I have used stat struct to check if file exists in the client directory.

After that if there does not exist same file name at the server's "dirname" directory it starts uploading the contents to server.



```c
FILE *source = fopen(clientPath, "rb");
if(source == NULL){
    sprintf(response, "Failed to open source file\n");
    if(write(fd_write_client, response, strlen(response)) < 0){
        perror("Failed to write to client fifo");
        exit(EXIT_FAILURE);
    }
    return;
}
else{
    FILE* dest = fopen(path, "wb");
    if(dest == NULL){
        sprintf(response,"Failed to open destination file\n");
        if(write(fd_write_client, response, strlen(response)) < 0){
            perror("Failed to write to client fifo");
            exit(EXIT_FAILURE);
        }
        return;
    }
    else{
        char buffer[1024];
        ssize_t bytes;
        ssize_t total_bytes = 0;

        while((bytes = fread(buffer, 1, sizeof(buffer), source)) > 0){
            fwrite(buffer, 1, bytes, dest);
            total_bytes += bytes;
        }
        fclose(source);
        fclose(dest);
        sprintf(response, "Upload successful. %ld bytes transferred.\n", total_bytes);
        if(write(fd_write_client, response, strlen(response)) < 0){
            perror("Failed to write to client fifo");
            exit(EXIT_FAILURE);
        }
    }
}
```

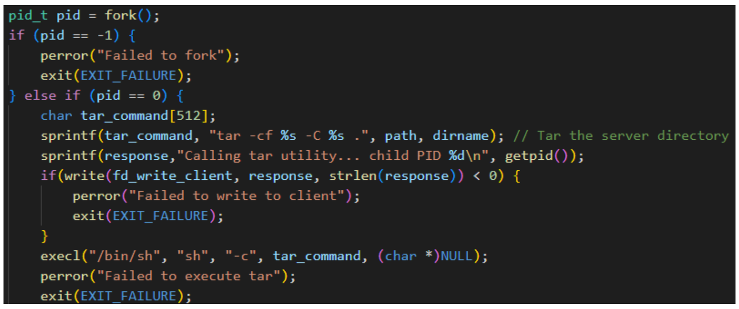
By using source and destination I have copied the contents of the file from client side to the server side.

-download <file>

Download command works almost same as upload command. Only difference is the source and destination files.

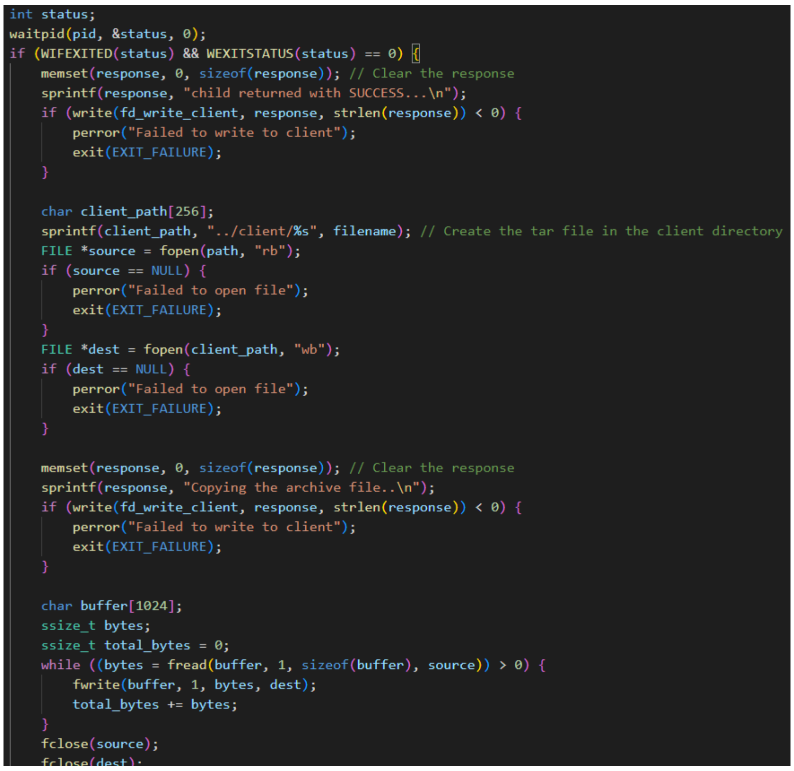
-archServer <fileName> .tar

ArchServer command uses fork and exec utilities to archive the contents of the "dirname" directory in the server side to the client side and stores them inside a .tar file.



```c
pid_t pid = fork();
if (pid == -1) {
    perror("Failed to fork");
    exit(EXIT_FAILURE);
} else if (pid == 0) {
    char tar_command[512];
    sprintf(tar_command, "tar -cf %s -C %s .", path, dirname); // Tar the server directory
    sprintf(response,"Calling tar utility... child PID %d\n", getpid());
    if(write(fd_write_client, response, strlen(response)) < 0) {
        perror("Failed to write to client");
        exit(EXIT_FAILURE);
    }
    execl("/bin/sh", "sh", "-c", tar_command, (char *)NULL);
    perror("Failed to execute tar");
    exit(EXIT_FAILURE);
```

Child process creates a .tar file in the server directory using execl.



```c
int status;
waitpid(pid, &status, 0);
if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
    memset(response, 0, sizeof(response)); // Clear the response
    sprintf(response, "child returned with SUCCESS...\n");
    if (write(fd_write_client, response, strlen(response)) < 0) {
        perror("Failed to write to client");
        exit(EXIT_FAILURE);
    }

    char client_path[256];
    sprintf(client_path, "../client/%s", filename); // Create the tar file in the client directory
    FILE *source = fopen(path, "rb");
    if (source == NULL) {
        perror("Failed to open file");
        exit(EXIT_FAILURE);
    }
    FILE *dest = fopen(client_path, "wb");
    if (dest == NULL) {
        perror("Failed to open file");
        exit(EXIT_FAILURE);
    }

    memset(response, 0, sizeof(response)); // Clear the response
    sprintf(response, "Copying the archive file..\n");
    if (write(fd_write_client, response, strlen(response)) < 0) {
        perror("Failed to write to client");
        exit(EXIT_FAILURE);
    }

    char buffer[1024];
    ssize_t bytes;
    ssize_t total_bytes = 0;
    while ((bytes = fread(buffer, 1, sizeof(buffer), source)) > 0) {
        fwrite(buffer, 1, bytes, dest);
        total_bytes += bytes;
    }
    fclose(source);
    fclose(dest);
```

Parent process of the archServer command copies the .tar in the server directory into the client side and removes the server side .tar file.

-killServer

When killServer is read from the client side, server sends a specific signal named SIGUSR1 and this is handled by the signal handler that I have defined for this command. In the signal handler, server side sends SIGURS2 signal to every client and frees up every resource it has allocated, also kills every child process.

-quit

When one of the clients sends quit command, the server side unlinks the corresponding FIFOs for that client and posts the semaphore so that if there exists a waiting client at the queue it can connect to the server.

Client Side Implementation:

The client side is designed to interact with the server, send commands and handle responses. It achieves this through the use of FIFOs (First In, First Out) and signal handlers.

FIFOs

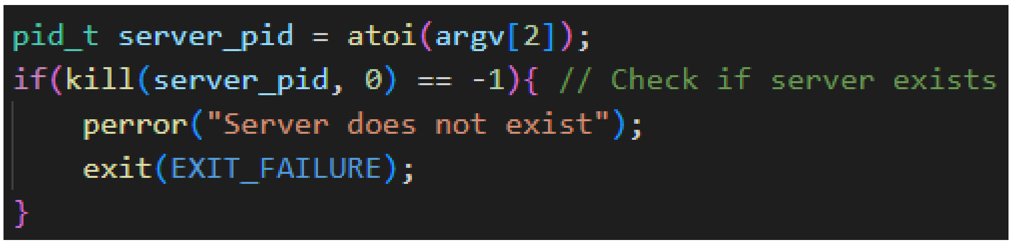
FIFOs are used to handle communication between the client and the server. Each client connection is associated with a pair of FIFOs: one for the client to read from and one for the client to write to. These FIFOs are created using the client's PID, ensuring that each pair of FIFOs is unique to each client.

Signal Handlers

The client has a signal handler function, handle_server_kill, which is designed to handle two types of signals:

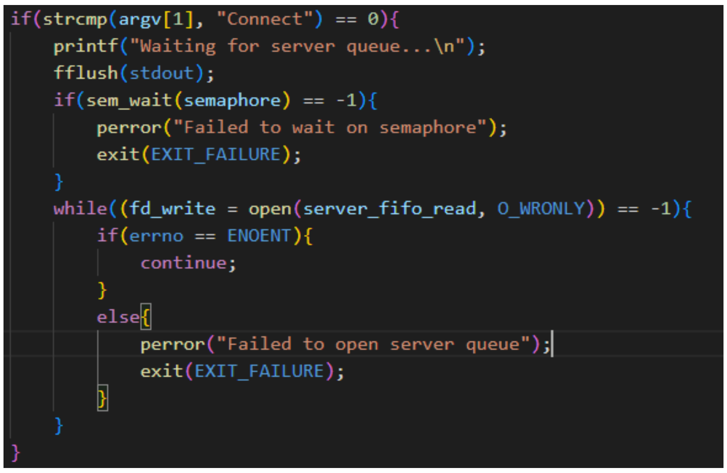
SIGUSR2: This signal is sent by the server when it is shutting down. When the client receives this signal, it prints a message indicating that the server is shutting down, closes the FIFOs, and exits gracefully.

SIGINT: This signal is sent when the user presses Ctrl+C. When the client receives this signal, it sends a "quit" command to the server, indicating that it wants to disconnect. If the command is successfully sent, the client closes the FIFOs and exits.



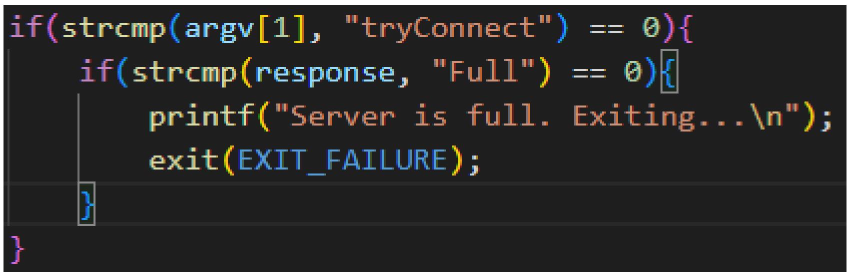
```c
pid_t server_pid = atoi(argv[2]);
if(kill(server_pid, 0) == -1){ // Check if server exists
    perror("Server does not exist");
    exit(EXIT_FAILURE);
}
```

Client side sends a 0 signal to the entered server PID to check if the server exists with that PID.



```c
if(strcmp(argv[1], "Connect") == 0){
    printf("Waiting for server queue...\n");
    fflush(stdout);
    if(sem_wait(semaphore) == -1){
        perror("Failed to wait on semaphore");
        exit(EXIT_FAILURE);
    }
    while((fd_write = open(server_fifo_read, O_WRONLY)) == -1){
        if(errno == ENOENT){
            continue;
        }
        else{
            perror("Failed to open server queue");
            exit(EXIT_FAILURE);
        }
    }
}
```

Client side has a semaphore to handle queue if the server is at full capacity.



```
if(strcmp(argv[1], "tryConnect") == 0){
    if(strcmp(response, "Full") == 0){
        printf("Server is full. Exiting...\n");
        exit(EXIT_FAILURE);
    }
}
```

If the server responses with a full when the command is tryConnect, client immediately exits.

When a user enters the help command, the client displays a list of available commands and their usage. Importantly, the help command is handled entirely on the client side and does not involve any communication with the server. This design decision was made to reduce unnecessary load on the server and to ensure that help information is always available to the user.

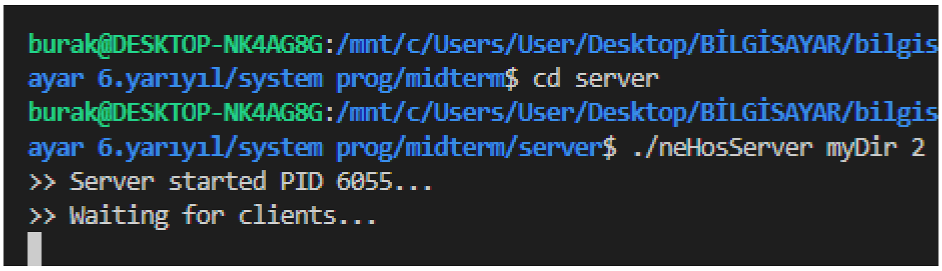
Client side handles every command other than the help with the server side, it sends the corresponding data to the server side and waits for the response of the server side and displays the response on the screen.

6. Test Plans and Results

Since I have used two different directories for server and client, I need to manually change directory to run the program. But I have created one makefile to compile the whole program at once.

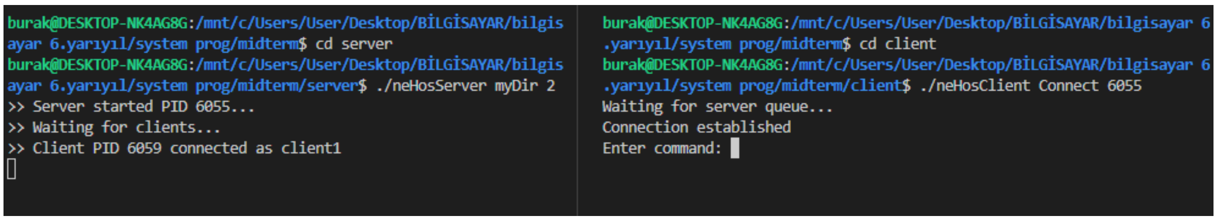
Steps:

- Compile the project from the midterm directory by entering “make”
- Open up more than one terminal
- Change directory to the server side “cd server” at one of the terminals
- Change directory to the client side “cd client” at one of the terminals
- Run server side “./neHosServer <dirname> <max. #of clients>
- Run client side “./neHosClient <Connect/tryConnect> ServerPID



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6055...
>> Waiting for clients...
```

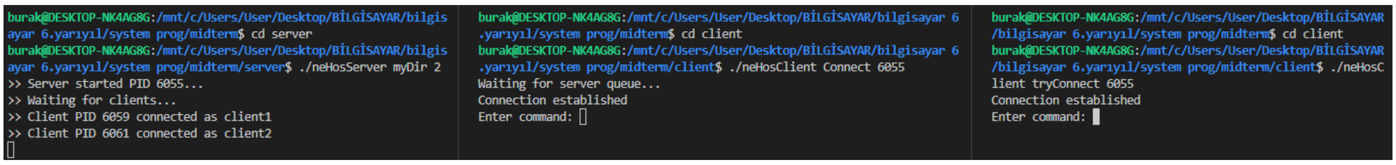
Server started running



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6055...
>> Waiting for clients...
>> Client PID 6059 connected as client1
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6055
Waiting for server queue...
Connection established
Enter command:
```

Client with PID 6059 has connected.

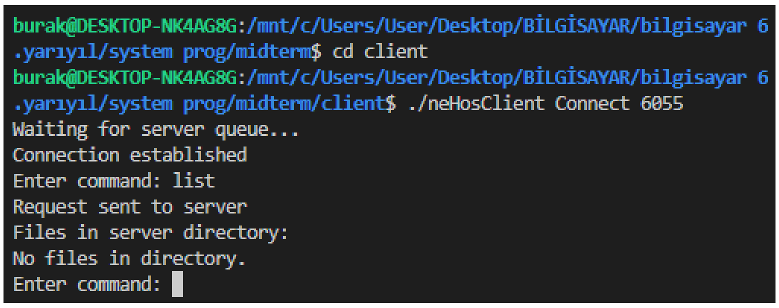


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6055...
>> Waiting for clients...
>> Client PID 6059 connected as client1
>> Client PID 6061 connected as client2
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6055
Waiting for server queue...
Connection established
Enter command:
```

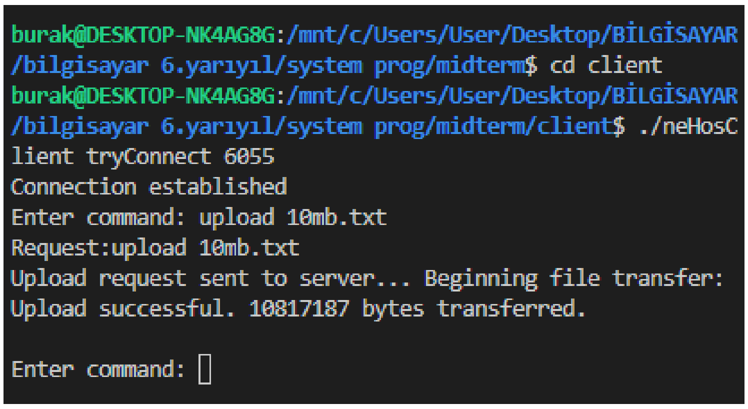
```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient tryConnect 6055
Connection established
Enter command:
```

Second client with PID 6061 has connected.



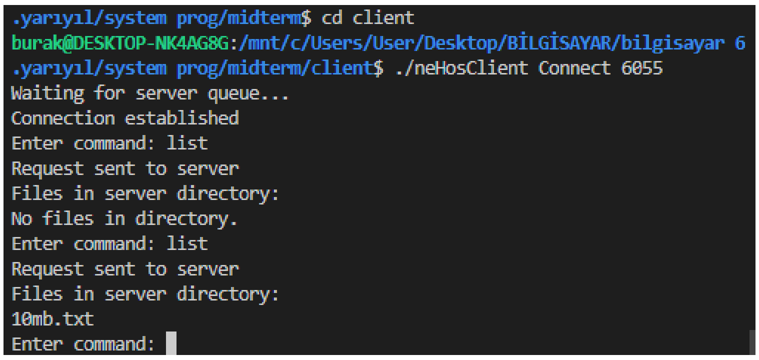
```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6055
Waiting for server queue...
Connection established
Enter command: list
Request sent to server
Files in server directory:
No files in directory.
Enter command:
```

First client has entered list.



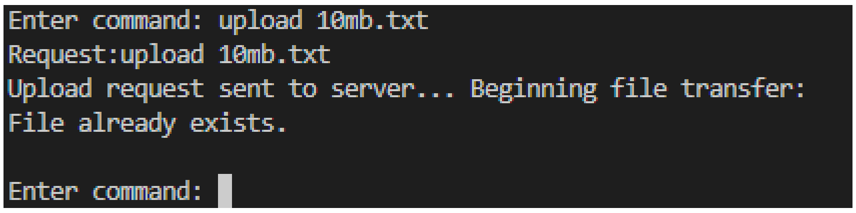
```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient tryConnect 6055
Connection established
Enter command: upload 10mb.txt
Request:upload 10mb.txt
Upload request sent to server... Beginning file transfer:
Upload successful. 10817187 bytes transferred.

Enter command:
```

Second client has uploaded a file to the server directory.



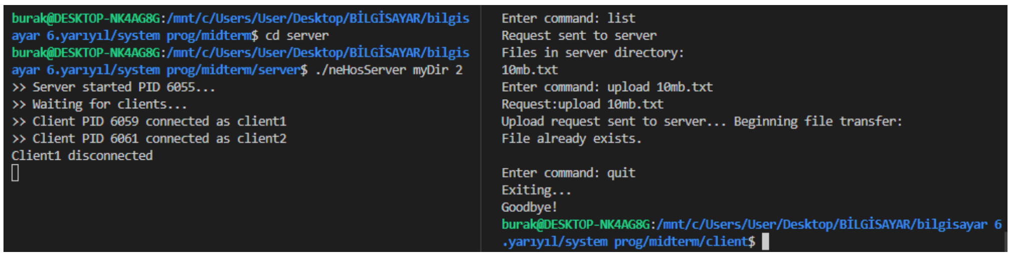
```
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6055
Waiting for server queue...
Connection established
Enter command: list
Request sent to server
Files in server directory:
No files in directory.
Enter command: list
Request sent to server
Files in server directory:
10mb.txt
Enter command:
```

First client has entered list and can see the 10mb.txt file.



```
Enter command: upload 10mb.txt
Request:upload 10mb.txt
Upload request sent to server... Beginning file transfer:
File already exists.

Enter command: 
```

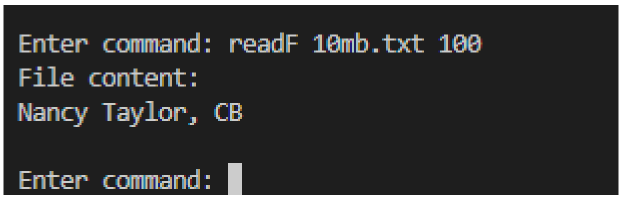
First client tries to re-upload the same file.



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6055...
>> Waiting for clients...
>> Client PID 6059 connected as client1
>> Client PID 6061 connected as client2
Client1 disconnected
```

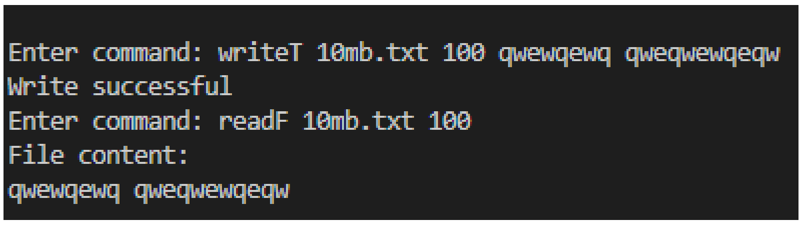
```
Enter command: list
Request sent to server
Files in server directory:
10mb.txt
Enter command: upload 10mb.txt
Request:upload 10mb.txt
Upload request sent to server... Beginning file transfer:
File already exists.

Enter command: quit
Exiting...
Goodbye!
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ 
```

First client has disconnected by entering quit.



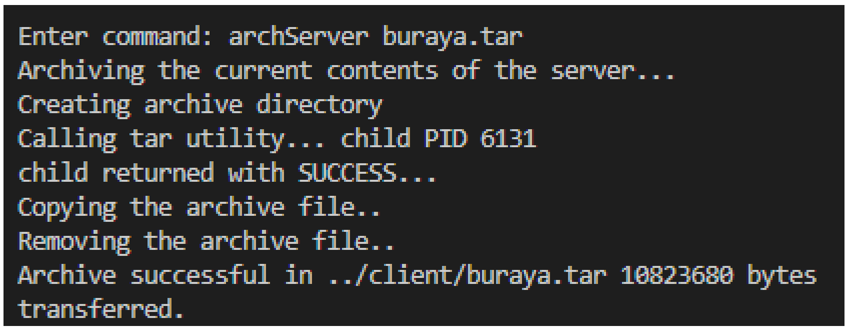
```
Enter command: readF 10mb.txt 100
File content:
Nancy Taylor, CB

Enter command: 
```

Second client has entered readF command.



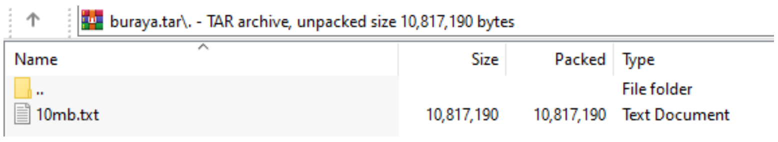
```
Enter command: writeT 10mb.txt 100 qwewqewq qweqwewqeqw
Write successful
Enter command: readF 10mb.txt 100
File content:
qwewqewq qweqwewqeqw
```

Second client has written and read to the 10mb.txt file.



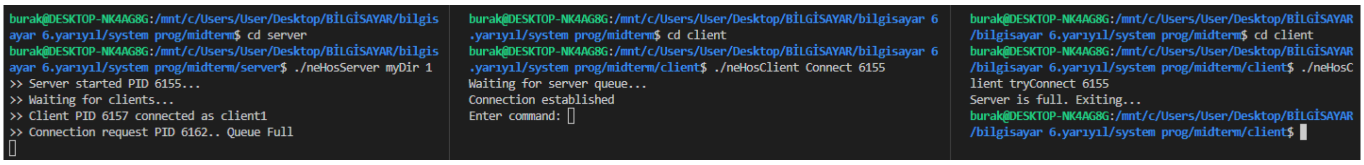
```
Enter command: archServer buraya.tar
Archiving the current contents of the server...
Creating archive directory
Calling tar utility... child PID 6131
child returned with SUCCESS...
Copying the archive file..
Removing the archive file..
Archive successful in ../client/buraya.tar 10823680 bytes
transferred.
```

Second client has archived the content of the server directory.



buraya.tar\. - TAR archive, unpacked size 10,817,190 bytes

Name	Size	Packed	Type
..			File folder
10mb.txt	10,817,190	10,817,190	Text Document

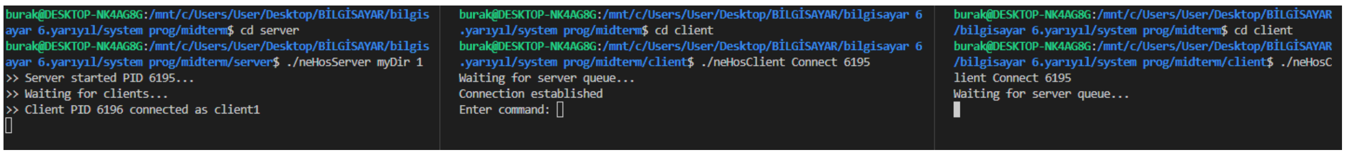


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 1
>> Server started PID 6155...
>> Waiting for clients...
>> Client PID 6157 connected as client1
>> Connection request PID 6162.. Queue Full
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6155
Waiting for server queue...
Connection established
Enter command:
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient tryConnect 6155
Server is full. Exiting...
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$
```

When the server is full and another client wants to join with tryConnect command.

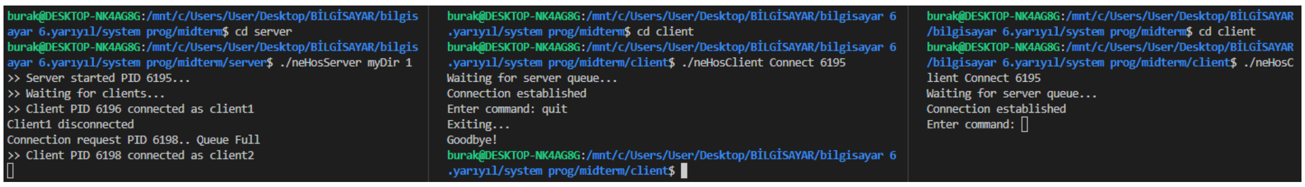


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 1
>> Server started PID 6195...
>> Waiting for clients...
>> Client PID 6196 connected as client1
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6195
Waiting for server queue...
Connection established
Enter command:
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient Connect 6195
Waiting for server queue...
```

When the server is full and another client wants to join with Connect command.

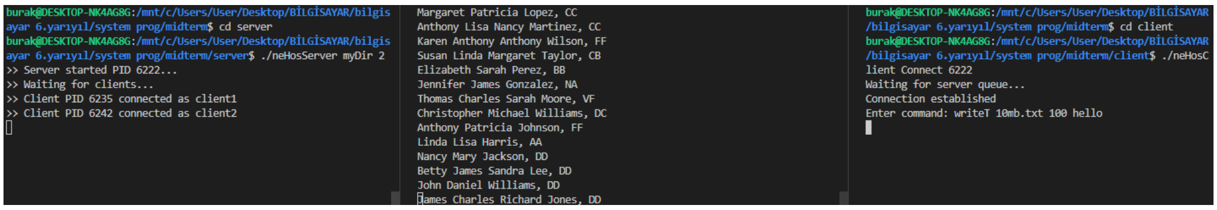


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 1
>> Server started PID 6195...
>> Waiting for clients...
>> Client PID 6196 connected as client1
Client1 disconnected
Connection request PID 6198.. Queue Full
>> Client PID 6198 connected as client2
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6195
Waiting for server queue...
Connection established
Enter command: quit
Exiting...
Goodbye!
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient Connect 6195
Waiting for server queue...
Connection established
Enter command:
```

When a free spot is available for the second client, it connects to the server.

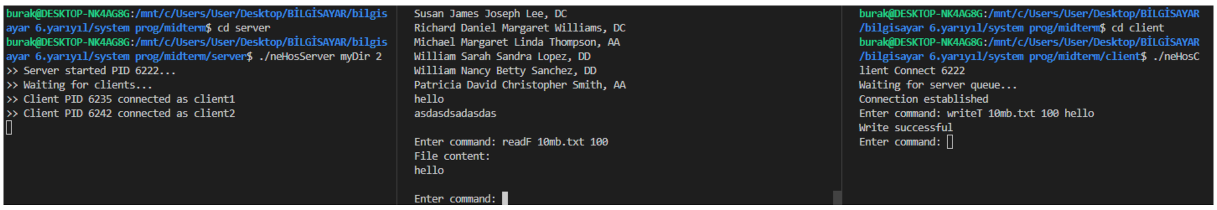


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6222...
>> Waiting for clients...
>> Client PID 6235 connected as client1
>> Client PID 6242 connected as client2
```

```
Margaret Patricia Lopez, CC
Anthony Lisa Nancy Martinez, CC
Karen Anthony Anthony Wilson, FF
Susan Linda Margaret Taylor, CB
Elizabeth Sarah Perez, BB
Jennifer James Gonzalez, NA
Thomas Charles Sarah Moore, VF
Christopher Michael Williams, DC
Anthony Patricia Johnson, FF
Linda Lisa Harris, AA
Nancy Mary Jackson, DD
Betty James Sandra Lee, DD
John Daniel Williams, DD
James Charles Richard Jones, DD
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient Connect 6222
Waiting for server queue...
Connection established
Enter command: writeT 10mb.txt 100 hello
```

While first client is printing every content of the 10mb.txt file, another client wants to write to the same file, the other clients waits for the other client to finish it job.

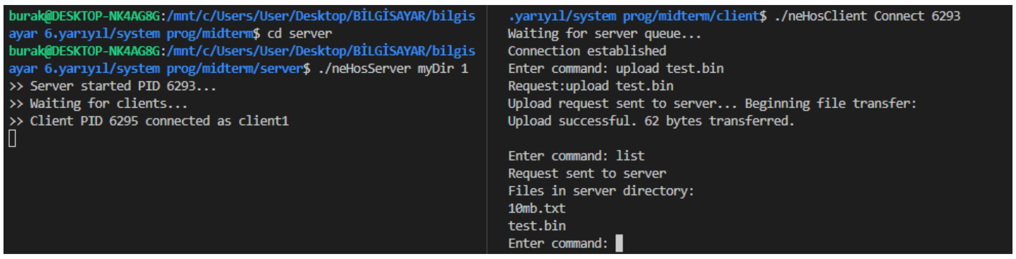


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6222...
>> Waiting for clients...
>> Client PID 6235 connected as client1
>> Client PID 6242 connected as client2
```

```
Susan James Joseph Lee, DC
Richard Daniel Margaret Williams, DC
Michael Margaret Linda Thompson, AA
William Sarah Sandra Lopez, DD
William Nancy Betty Sanchez, DD
Patricia David Christopher Smith, AA
hello
asdasdsadasdas

Enter command: readF 10mb.txt 100
File content:
hello

Enter command:
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient Connect 6222
Waiting for server queue...
Connection established
Enter command: writeT 10mb.txt 100 hello
Write successful
Enter command:
```

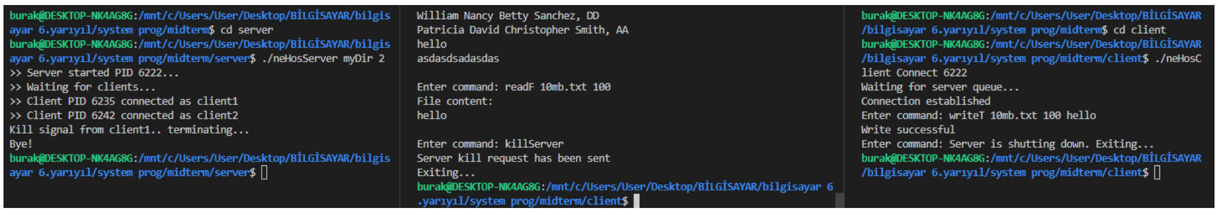
After first client finishes, second client writes to the file.



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 1
>> Server started PID 6293...
>> Waiting for clients...
>> Client PID 6295 connected as client1
```

```
.yarıyıl/system prog/midterm/client$ ./neHosClient Connect 6293
Waiting for server queue...
Connection established
Enter command: upload test.bin
Request:upload test.bin
Upload request sent to server... Beginning file transfer:
Upload successful. 62 bytes transferred.

Enter command: list
Request sent to server
Files in server directory:
10mb.txt
test.bin
Enter command:
```

After uploading test.bin file.

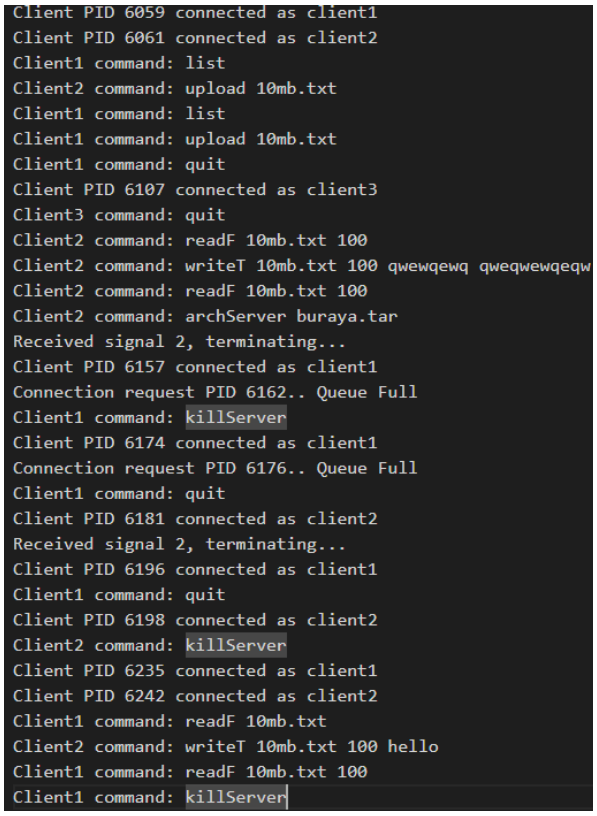


```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm$ cd server
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$ ./neHosServer myDir 2
>> Server started PID 6222...
>> Waiting for clients...
>> Client PID 6235 connected as client1
>> Client PID 6242 connected as client2
Kill signal from client1.. terminating...
Bye!
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgis
ayar 6.yarıyıl/system prog/midterm/server$
```

```
William Nancy Betty Sanchez, DD
Patricia David Christopher Smith, AA
hello
asdasdsadasdas

Enter command: readF 10mb.txt 100
File content:
hello

Enter command: killServer
Server kill request has been sent
Exiting...
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6
.yarıyıl/system prog/midterm/client$
```

```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm$ cd client
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$ ./neHosC
lient Connect 6222
Waiting for server queue...
Connection established
Enter command: writeT 10mb.txt 100 hello
Write successful
Enter command: Server is shutting down. Exiting...
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR
/bilgisayar 6.yarıyıl/system prog/midterm/client$
```

When killServer command is entered from one client every other client is killed by the server.



```
Client PID 6059 connected as client1
Client PID 6061 connected as client2
Client1 command: list
Client2 command: upload 10mb.txt
Client1 command: list
Client1 command: upload 10mb.txt
Client1 command: quit
Client PID 6107 connected as client3
Client3 command: quit
Client2 command: readF 10mb.txt 100
Client2 command: writeT 10mb.txt 100 qwewqewq qweqwewqeqw
Client2 command: readF 10mb.txt 100
Client2 command: archServer buraya.tar
Received signal 2, terminating...
Client PID 6157 connected as client1
Connection request PID 6162.. Queue Full
Client1 command: killServer
Client PID 6174 connected as client1
Connection request PID 6176.. Queue Full
Client1 command: quit
Client PID 6181 connected as client2
Received signal 2, terminating...
Client PID 6196 connected as client1
Client1 command: quit
Client PID 6198 connected as client2
Client2 command: killServer
Client PID 6235 connected as client1
Client PID 6242 connected as client2
Client1 command: readF 10mb.txt
Client2 command: writeT 10mb.txt 100 hello
Client1 command: readF 10mb.txt 100
Client1 command: killServer
```

This is the server log file that stores the client id and the command that has been entered.