FATİH DOĞAÇ
1901042654
CSE344 MIDTERM PROJECT REPORT

1. Introduction

This project is to design and implement a file server that enables multiple clients to connect, access, modify and archive the files in a specific directory located at the server side.
The project implemented as a server side and a client side programs

Server side :
neHosServer <dirname> <max. #ofClients>
the Server side is expected enter the specified directory (create dirname if the dirname does not
exits), create a log file for the clients and prompt its PID for the clients to connect. The for each
client connected will fork a copy of itself in order to serve the specified client (commands are given
on the client side). If a kill signal is generated (either by Ctrl-C or from a client side request)
Server is expected to display the request, send kill signals (requests) to its child processes, ensure
the log file is created properly and exit.

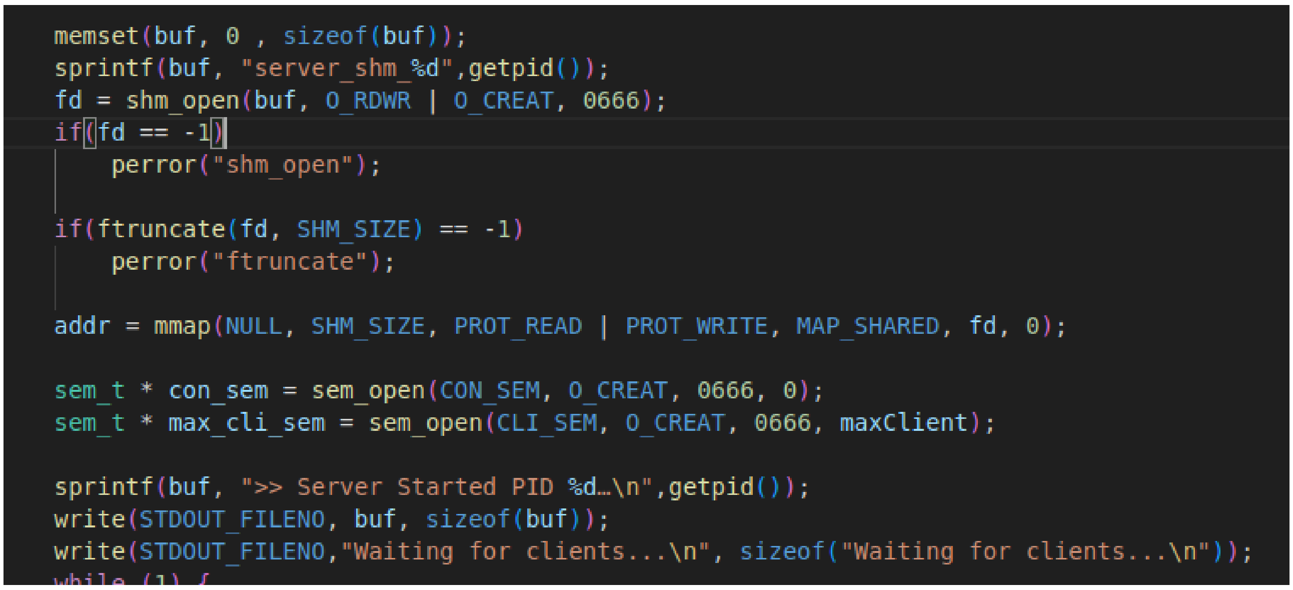
Client side :
neHosClient <Connect/tryConnect> ServerPID
the client program with Connect option request a spot from the Server Que with ServerPID and
connects if a spot is available (if not the client should wait until a spot becomes available,
tryConnect option leaves without waiting if the Que is full). When connected the client can
perform the following requests :
- help
display the list of possible client requests
- list
sends a request to display the list of files in Servers directory
(also displays the list received from the Server)readF <file> <line #>
requests to display the # line of the <file>, if no line number is given
the whole contents of the file is requested (and displayed on the client side)
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
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)
download <file>
request to receive <file> from Servers directory to client side
archServer <fileName>.tar
Using fork, exec and tar utilities create a child process that will collect all the files currently
available on the the Server side and store them in the <filename>.tar archive
killServer
Sends a kill request to the Server
quit
Send write request to Server side log file and quits

2. IPC (Inter Process Communication) Part

First of all, clients cannot tryConnect. I implemented a queue design and clients can only join to queue and wait.

Let me walk you through IPC part:



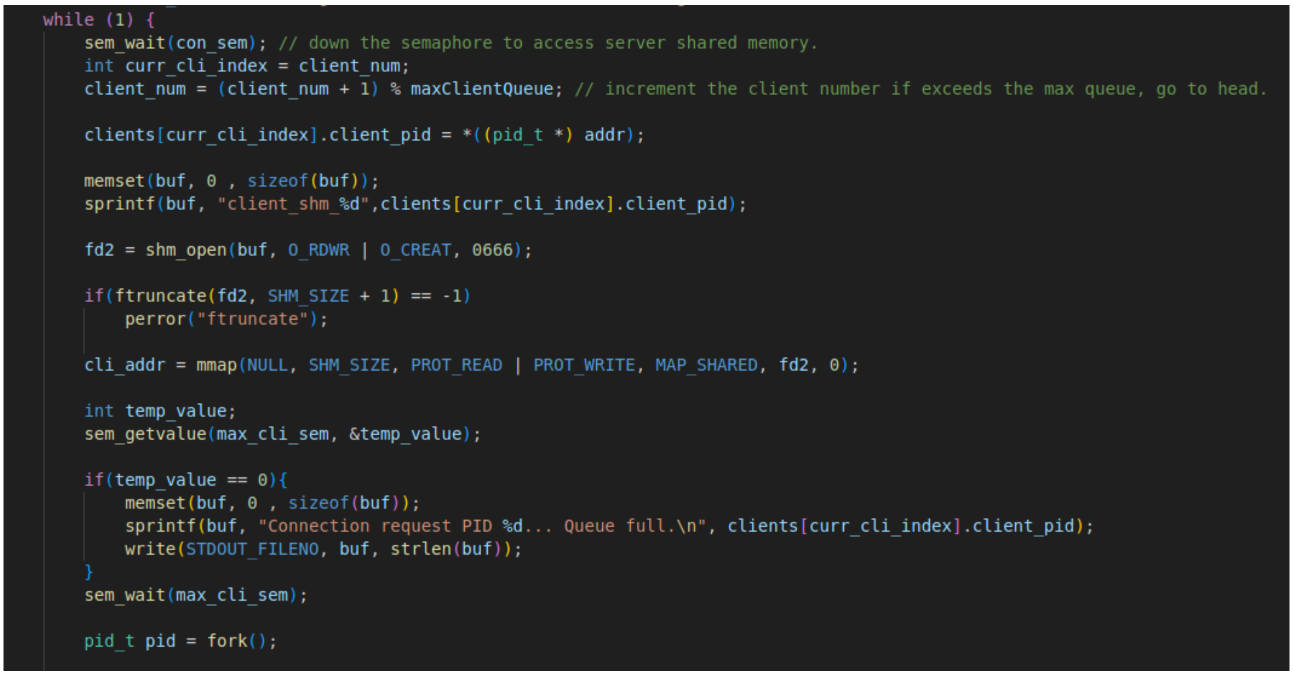
```
memset(buf, 0 , sizeof(buf));
sprintf(buf, "server_shm_%d",getpid());
fd = shm_open(buf, O_RDWR | O_CREAT, 0666);
if(fd == -1)
    perror("shm_open");

if(ftruncate(fd, SHM_SIZE) == -1)
    perror("ftruncate");

addr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

sem_t * con_sem = sem_open(CON_SEM, O_CREAT, 0666, 0);
sem_t * max_cli_sem = sem_open(CLI_SEM, O_CREAT, 0666, maxClient);

sprintf(buf, ">> Server Started PID %d…\n",getpid());
write(STDOUT_FILENO, buf, sizeof(buf));
write(STDOUT_FILENO,"Waiting for clients...\n", sizeof("Waiting for clients...\n"));
while (1) {
```

First, server creates a shared memory named “server_shm_{SERVER_PID}” and sets a size to it with ftruncate. Then mmap’s it. Then creates 2 semaphores:

con_sem is created for if two clients wants to connect the server at the same time, they will have to wait for con_sem. It stands for connection semaphore.

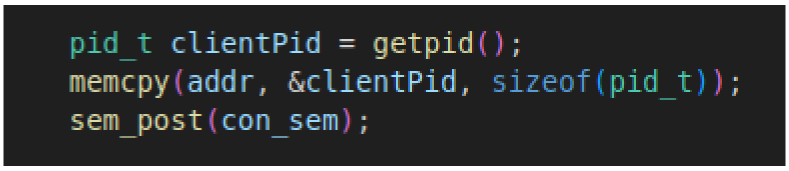
max_cli_sem is created to count current working clients. It starts with maxClient number and for each client that joins, it gets decremented by one.

Now, server started:



```
while (1) {
    sem_wait(con_sem); // down the semaphore to access server shared memory.
    int curr_cli_index = client_num;
    client_num = (client_num + 1) % maxClientQueue; // increment the client number if exceeds the max queue, go to head.

    clients[curr_cli_index].client_pid = *((pid_t *) addr);

    memset(buf, 0 , sizeof(buf));
    sprintf(buf, "client_shm_%d",clients[curr_cli_index].client_pid);

    fd2 = shm_open(buf, O_RDWR | O_CREAT, 0666);

    if(ftruncate(fd2, SHM_SIZE + 1) == -1)
        perror("ftruncate");

    cli_addr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd2, 0);

    int temp_value;
    sem_getvalue(max_cli_sem, &temp_value);

    if(temp_value == 0){
        memset(buf, 0 , sizeof(buf));
        sprintf(buf, "Connection request PID %d... Queue full.\n", clients[curr_cli_index].client_pid);
        write(STDOUT_FILENO, buf, strlen(buf));
    }
    sem_wait(max_cli_sem);

    pid_t pid = fork();
```

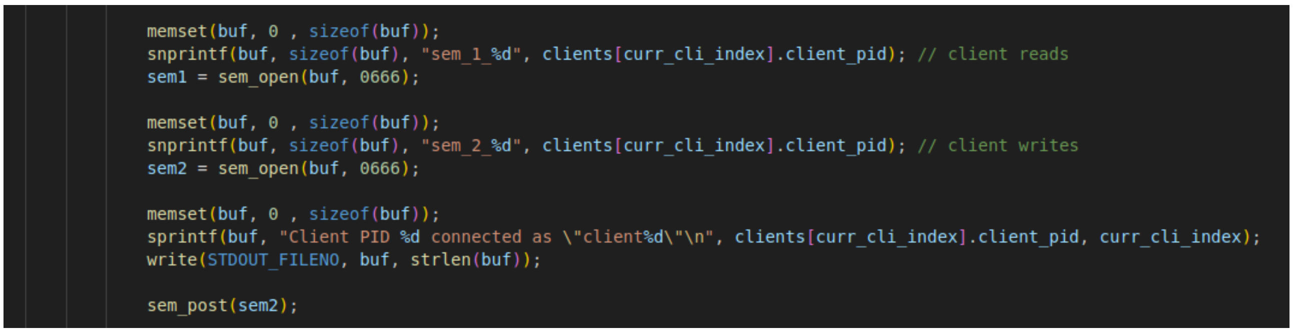
Client side:



```
pid_t clientPid = getpid();
memcpy(addr, &clientPid, sizeof(pid_t));
sem_post(con_sem);
```

Clients know about the connection semaphore(con_sem) and when a client joins, it posts the con_sem. And the server infinitely waits for clients with sem_wait(con_sem), if a client posts the con_sem, server forks itself and creates a child to attend to that specific client.

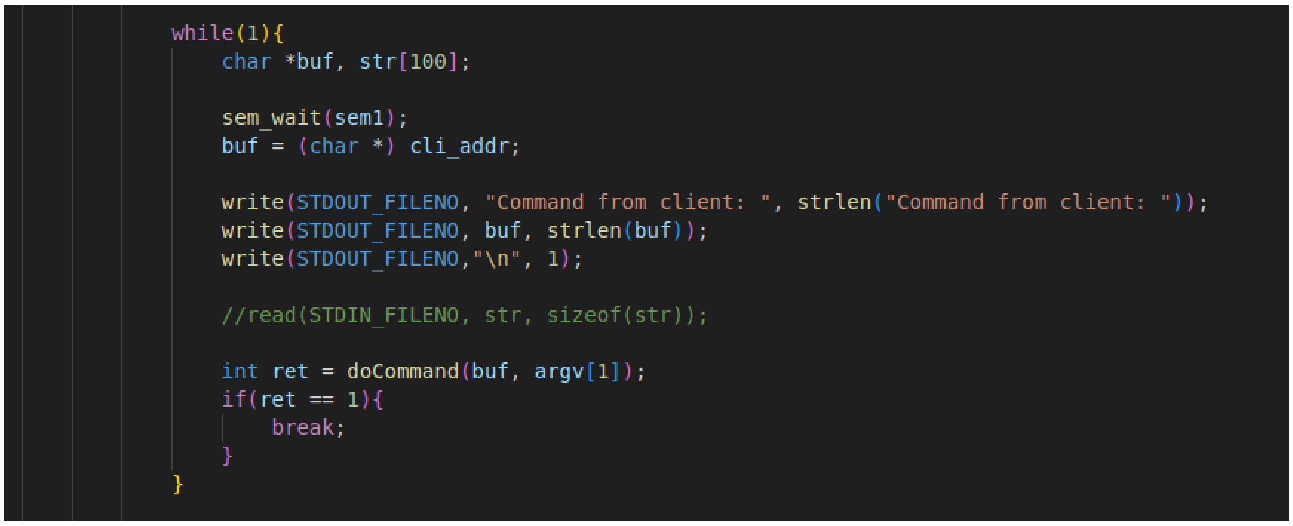
Before forking itself, server creates the client's unique shared memory. And decrements the available slot number (max_cli_sem) by one.. So others will know one of the places is occupied.



```c
memset(buf, 0 , sizeof(buf));
snprintf(buf, sizeof(buf), "sem_1_%d", clients[curr_cli_index].client_pid); // client reads
sem1 = sem_open(buf, 0666);

memset(buf, 0 , sizeof(buf));
snprintf(buf, sizeof(buf), "sem_2_%d", clients[curr_cli_index].client_pid); // client writes
sem2 = sem_open(buf, 0666);

memset(buf, 0 , sizeof(buf));
sprintf(buf, "Client PID %d connected as \"client%d\"\n", clients[curr_cli_index].client_pid, curr_cli_index);
write(STDOUT_FILENO, buf, strlen(buf));

sem_post(sem2);
```

When the client connects, server opens its unique semaphores with client. (Server doesn’t create them, client does.) And posts the second unique semaphore (sem2) to notify the client that it has accepted to server.

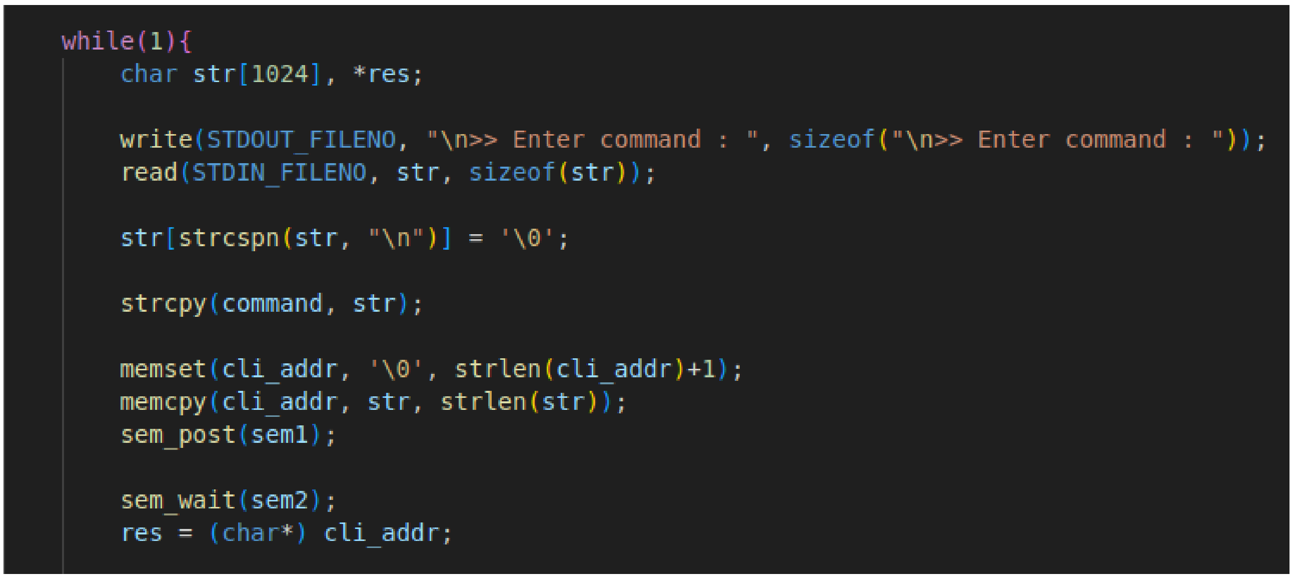
And then server’s child enters an infinite loop for the client:



```c
while(1){
    char *buf, str[100];

    sem_wait(sem1);
    buf = (char *) cli_addr;

    write(STDOUT_FILENO, "Command from client: ", strlen("Command from client: "));
    write(STDOUT_FILENO, buf, strlen(buf));
    write(STDOUT_FILENO,"\n", 1);

    //read(STDIN_FILENO, str, sizeof(str));

    int ret = doCommand(buf, argv[1]);
    if(ret == 1){
        break;
    }
}
```

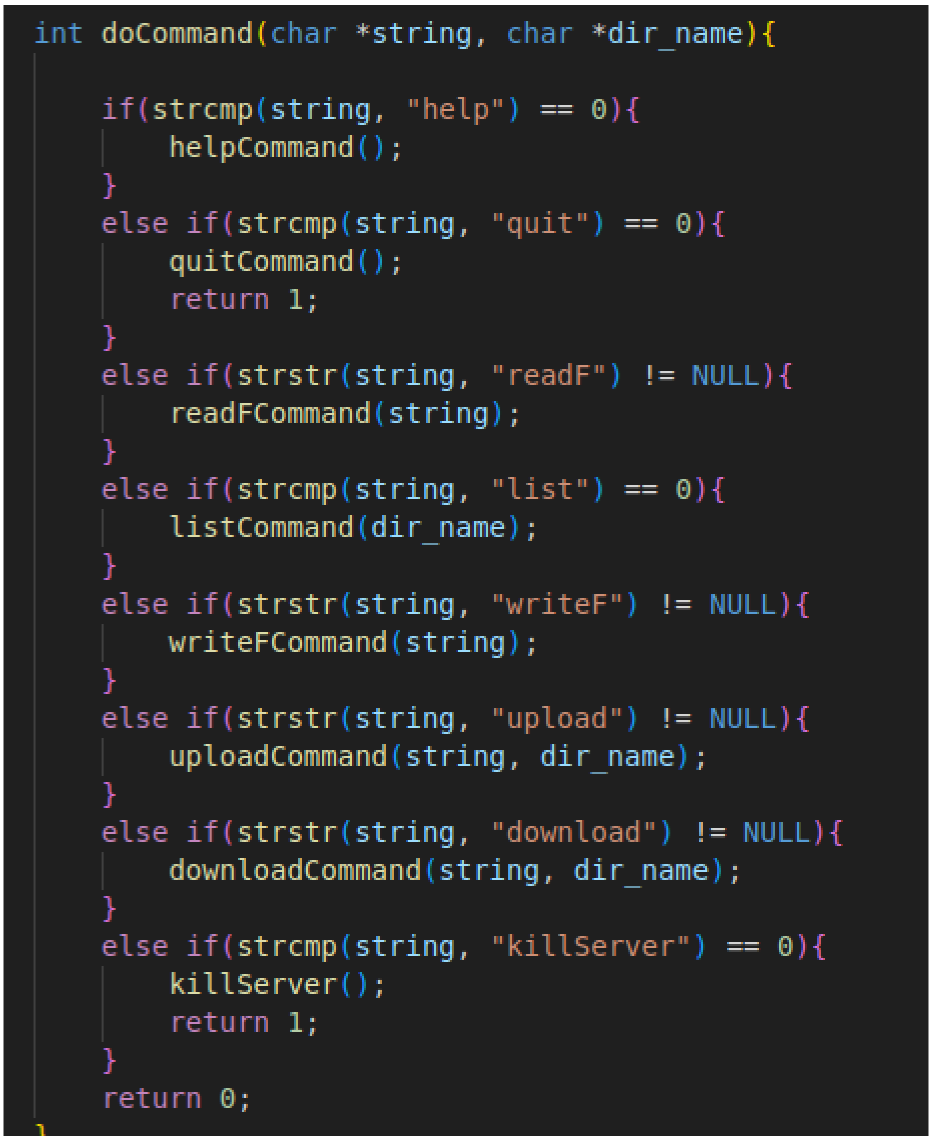
It basicly waits for sem1. When client increases the sem1, that means client wrote to the shared memory, server can read from it now.

Client:



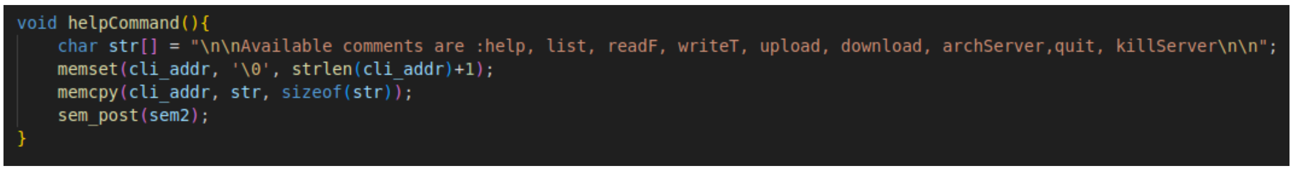
```c
while(1){
    char str[1024], *res;

    write(STDOUT_FILENO, "\n>> Enter command : ", sizeof("\n>> Enter command : "));
    read(STDIN_FILENO, str, sizeof(str));

    str[strcspn(str, "\n")] = '\0';

    strcpy(command, str);

    memset(cli_addr, '\0', strlen(cli_addr)+1);
    memcpy(cli_addr, str, strlen(str));
    sem_post(sem1);

    sem_wait(sem2);
    res = (char*) cli_addr;
```

After server reads the command, it performs the doCommand() function. It is basically the main function of the child process.



```c
int doCommand(char *string, char *dir_name){

    if(strcmp(string, "help") == 0){
        helpCommand();
    }
    else if(strcmp(string, "quit") == 0){
        quitCommand();
        return 1;
    }
    else if(strstr(string, "readF") != NULL){
        readFCommand(string);
    }
    else if(strcmp(string, "list") == 0){
        listCommand(dir_name);
    }
    else if(strstr(string, "writeF") != NULL){
        writeFCommand(string);
    }
    else if(strstr(string, "upload") != NULL){
        uploadCommand(string, dir_name);
    }
    else if(strstr(string, "download") != NULL){
        downloadCommand(string, dir_name);
    }
    else if(strcmp(string, "killServer") == 0){
        killServer();
        return 1;
    }
    return 0;
```

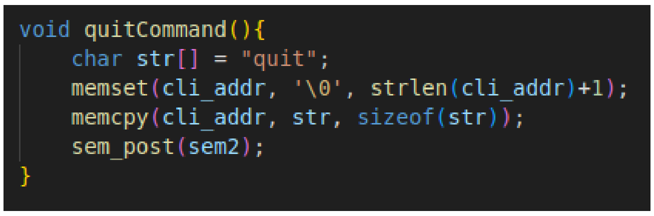
If we were to look into those functions:
Help:



```c
void helpCommand(){
    char str[] = "\n\nAvailable comments are :help, list, readF, writeT, upload, download, archServer,quit, killServer\n\n";
    memset(cli_addr, '\0', strlen(cli_addr)+1);
    memcpy(cli_addr, str, sizeof(str));
    sem_post(sem2);
}
```

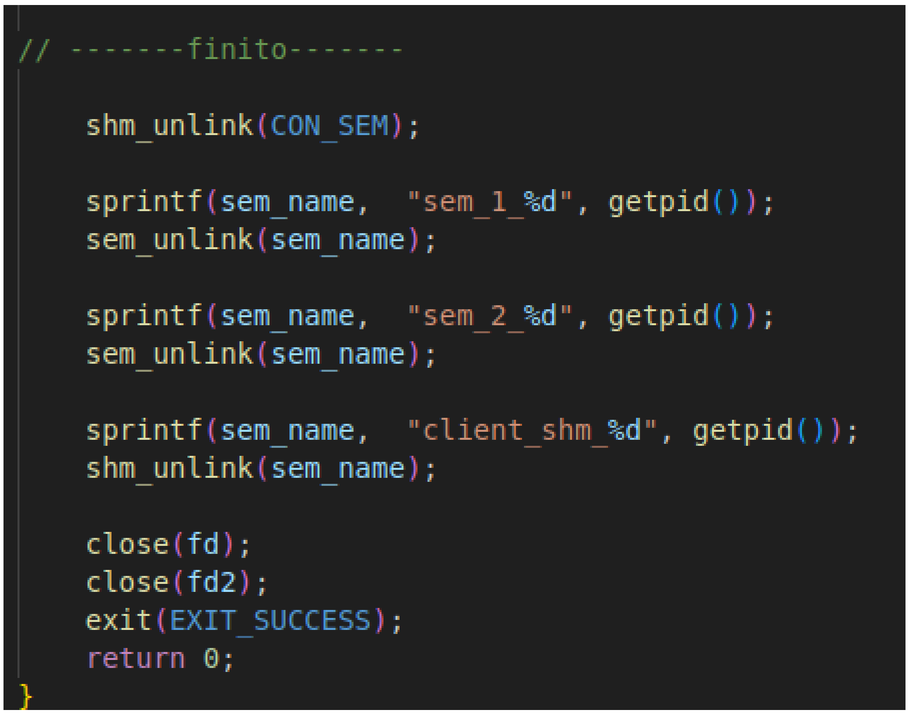
This is the most basic communication between server and client. There is a string and server writes this string to shared memory than notifies the client by incrementing the sem2.

Quit:



```c
void quitCommand(){
    char str[] = "quit";
    memset(cli_addr, '\0', strlen(cli_addr)+1);
    memcpy(cli_addr, str, sizeof(str));
    sem_post(sem2);
}
```

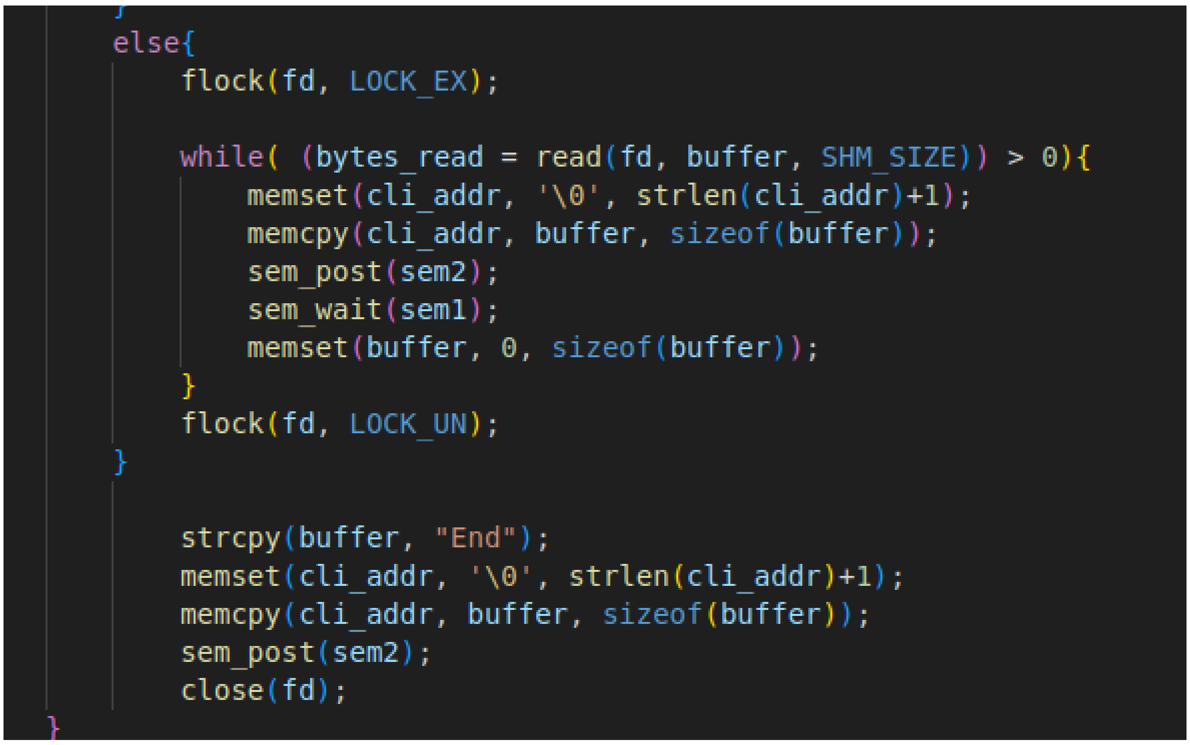
quit command notifies the user one last time. Then client performs its quit actions:



```c
// -------finito-------

    shm_unlink(CON_SEM);

    sprintf(sem_name,  "sem_1_%d", getpid());
    sem_unlink(sem_name);

    sprintf(sem_name,  "sem_2_%d", getpid());
    sem_unlink(sem_name);

    sprintf(sem_name,  "client_shm_%d", getpid());
    shm_unlink(sem_name);

    close(fd);
    close(fd2);
    exit(EXIT_SUCCESS);
    return 0;
}
```

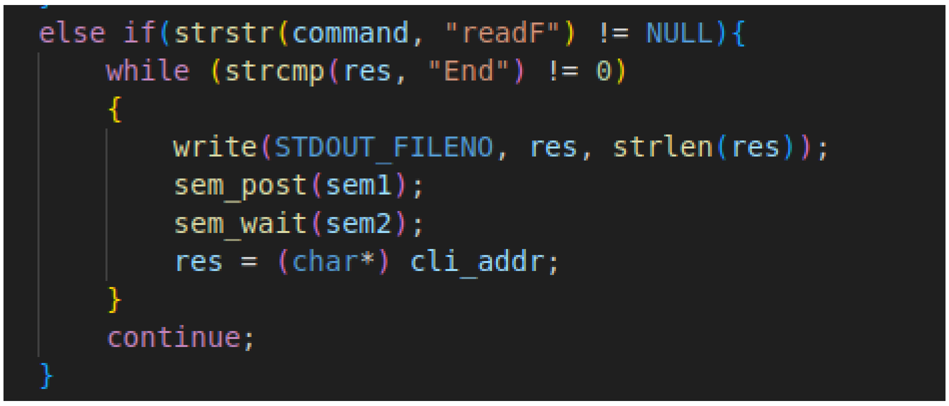
It cleans up the semaphores, shared memories and file descriptors. Then leaves

readF:



```c
        else{
            flock(fd, LOCK_EX);

            while( (bytes_read = read(fd, buffer, SHM_SIZE)) > 0){
                memset(cli_addr, '\0', strlen(cli_addr)+1);
                memcpy(cli_addr, buffer, sizeof(buffer));
                sem_post(sem2);
                sem_wait(sem1);
                memset(buffer, 0, sizeof(buffer));
            }
            flock(fd, LOCK_UN);
        }

            strcpy(buffer, "End");
            memset(cli_addr, '\0', strlen(cli_addr)+1);
            memcpy(cli_addr, buffer, sizeof(buffer));
            sem_post(sem2);
            close(fd);
    }
```

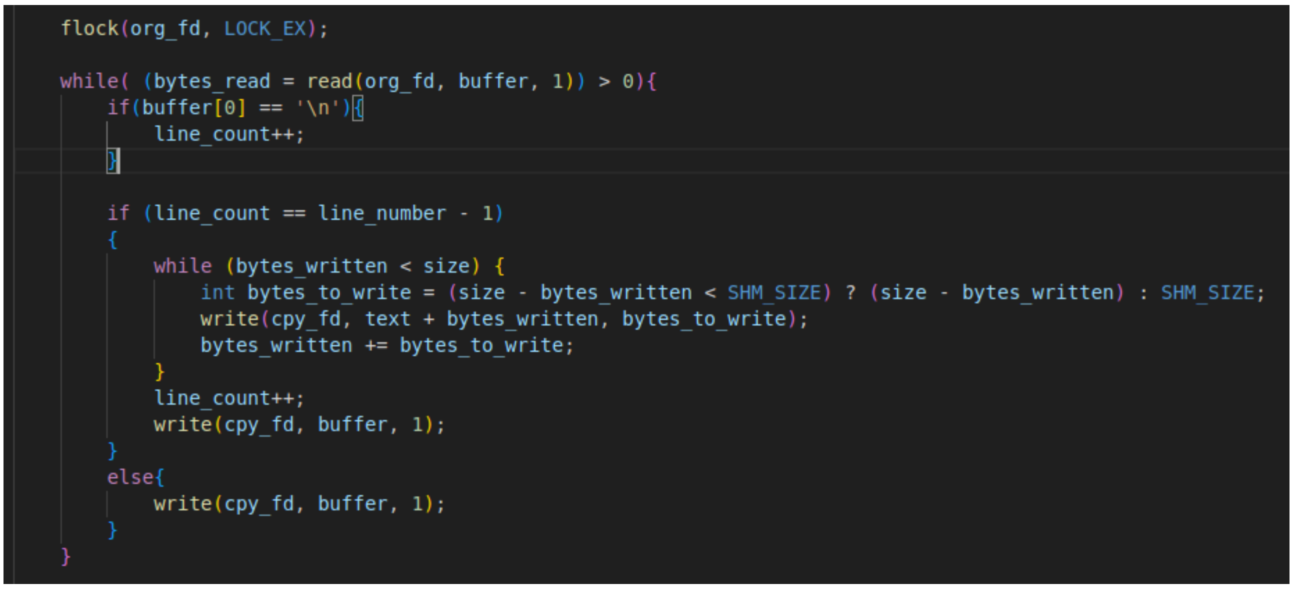
readF command, locks the file to be read, reads it SHM_SIZE at a time, which is 1024. Then it enters a loop with client and writes 1024 bytes at a time to shared memory.



```
else if(strstr(command, "readF") != NULL){
    while (strcmp(res, "End") != 0)
    {
        write(STDOUT_FILENO, res, strlen(res));
        sem_post(sem1);
        sem_wait(sem2);
        res = (char*) cli_addr;
    }
    continue;
}
```

And client is reading from shared memory and prints it into their terminal until an “End” string is received. Then it exits from the loop and continues to enter commands.

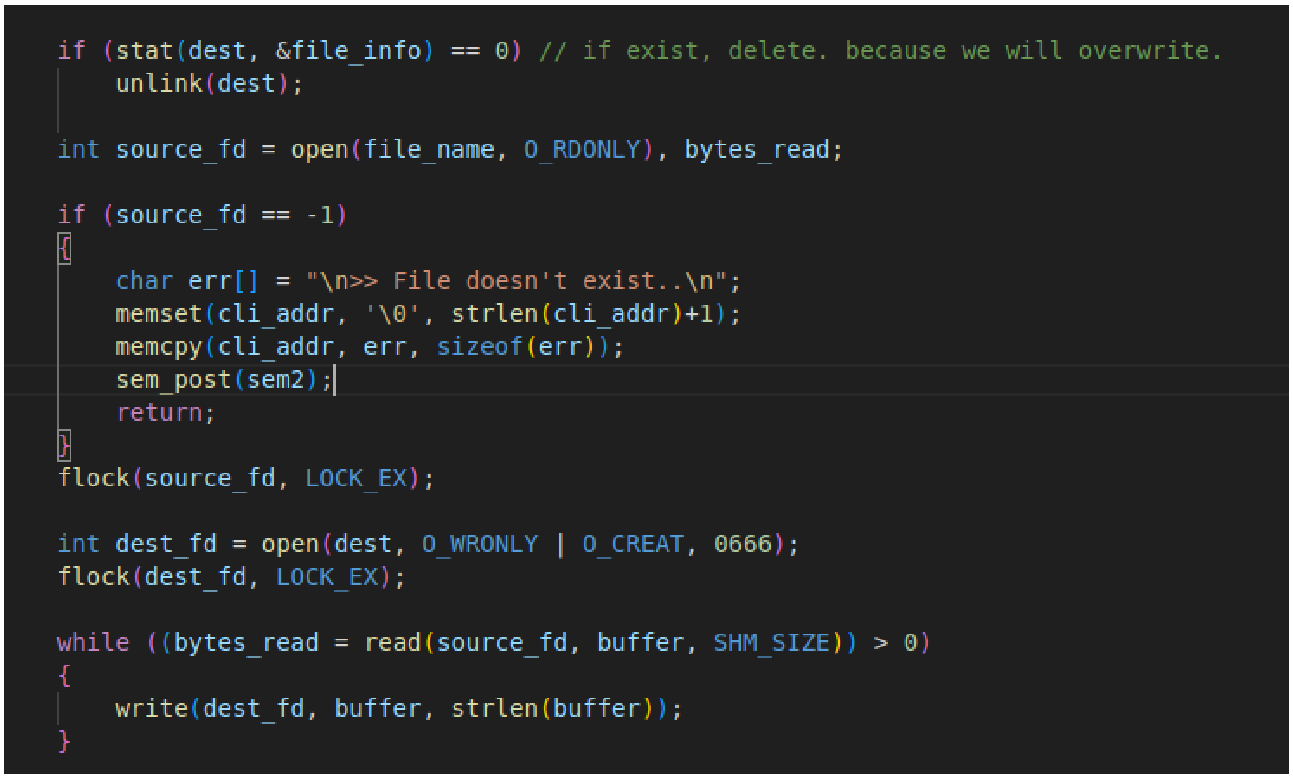
WriteF:



```
    flock(org_fd, LOCK_EX);

    while( (bytes_read = read(org_fd, buffer, 1)) > 0){
        if(buffer[0] == '\n'){
            line_count++;
        }

        if (line_count == line_number - 1)
        {
            while (bytes_written < size) {
                int bytes_to_write = (size - bytes_written < SHM_SIZE) ? (size - bytes_written) : SHM_SIZE;
                write(cpy_fd, text + bytes_written, bytes_to_write);
                bytes_written += bytes_to_write;
            }
            line_count++;
            write(cpy_fd, buffer, 1);
        }
        else{
            write(cpy_fd, buffer, 1);
        }
    }
```

WriteF, reads from the file that specified. Creates a temp file and starts writing to it. Then when the given line number is reached it appends the given string. If there is no given line number, it appends to the end of file.

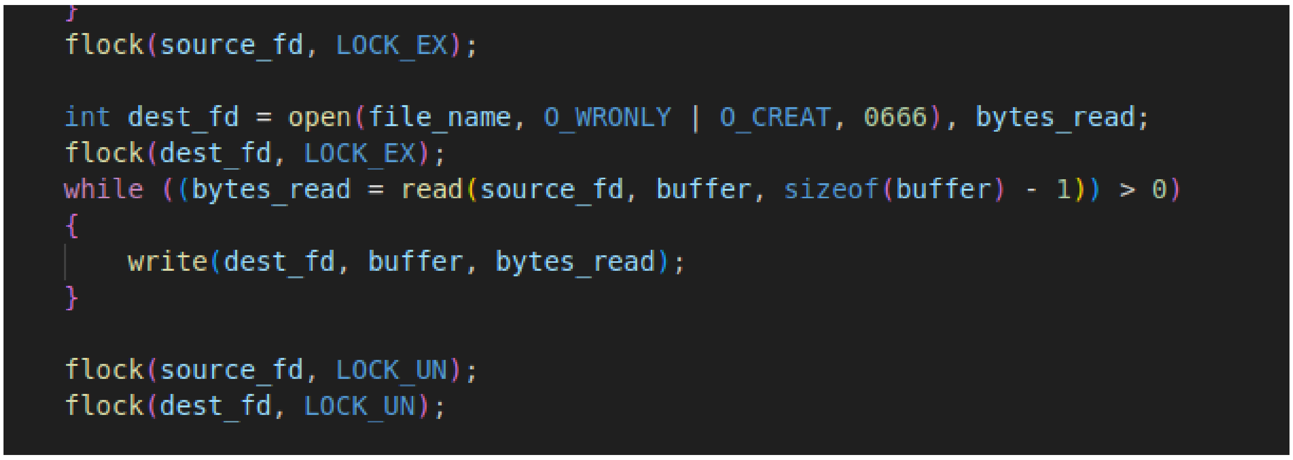
Upload:



```
if (stat(dest, &file_info) == 0) // if exist, delete. because we will overwrite.
    unlink(dest);

int source_fd = open(file_name, O_RDONLY), bytes_read;

if (source_fd == -1)
{
    char err[] = "\n>> File doesn't exist..\n";
    memset(cli_addr, '\0', strlen(cli_addr)+1);
    memcpy(cli_addr, err, sizeof(err));
    sem_post(sem2);
    return;
}
flock(source_fd, LOCK_EX);

int dest_fd = open(dest, O_WRONLY | O_CREAT, 0666);
flock(dest_fd, LOCK_EX);

while ((bytes_read = read(source_fd, buffer, SHM_SIZE)) > 0)
{
    write(dest_fd, buffer, strlen(buffer));
}
```

Upload doesn’t know about directories, it can only upload files in its current path. It basicly locks the file and if the file exists in the destination, it deletes the file to override. And then reads from source file and writes to destination file.

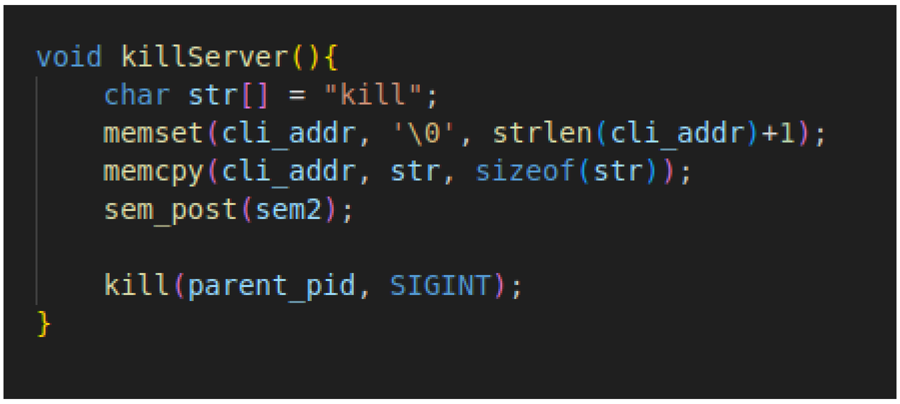
Download:



```
}
flock(source_fd, LOCK_EX);

int dest_fd = open(file_name, O_WRONLY | O_CREAT, 0666), bytes_read;
flock(dest_fd, LOCK_EX);
while ((bytes_read = read(source_fd, buffer, sizeof(buffer) - 1)) > 0)
{
    write(dest_fd, buffer, bytes_read);
}

flock(source_fd, LOCK_UN);
flock(dest_fd, LOCK_UN);
```

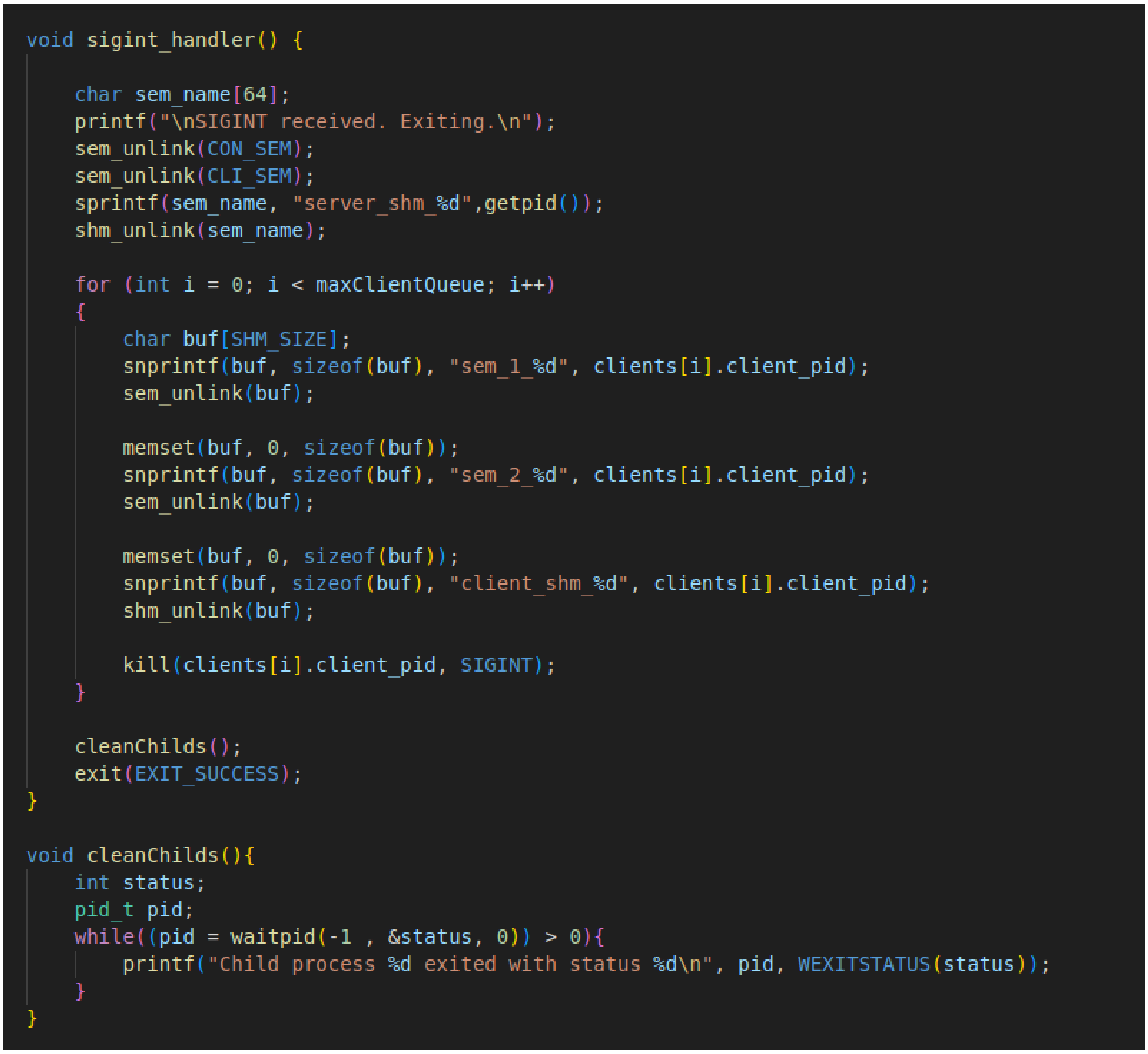
Download does the same but backwards.

KillServer:



```c
void killServer(){
    char str[] = "kill";
    memset(cli_addr, '\0', strlen(cli_addr)+1);
    memcpy(cli_addr, str, sizeof(str));
    sem_post(sem2);

    kill(parent_pid, SIGINT);
}
```

killServer, sends a string “kill” to client to let it exit on its own. If not no worries. It kills the parent process by signaling it SIGINT. So, SIGINT handler receives it.



```c
void sigint_handler() {

    char sem_name[64];
    printf("\nSIGINT received. Exiting.\n");
    sem_unlink(CON_SEM);
    sem_unlink(CLI_SEM);
    sprintf(sem_name, "server_shm_%d",getpid());
    shm_unlink(sem_name);

    for (int i = 0; i < maxClientQueue; i++)
    {
        char buf[SHM_SIZE];
        snprintf(buf, sizeof(buf), "sem_1_%d", clients[i].client_pid);
        sem_unlink(buf);

        memset(buf, 0, sizeof(buf));
        snprintf(buf, sizeof(buf), "sem_2_%d", clients[i].client_pid);
        sem_unlink(buf);

        memset(buf, 0, sizeof(buf));
        snprintf(buf, sizeof(buf), "client_shm_%d", clients[i].client_pid);
        shm_unlink(buf);

        kill(clients[i].client_pid, SIGINT);
    }

    cleanChilds();
    exit(EXIT_SUCCESS);
}

void cleanChilds(){
    int status;
    pid_t pid;
    while((pid = waitpid(-1 , &status, 0)) > 0){
        printf("Child process %d exited with status %d\n", pid, WEXITSTATUS(status));
    }
}
```

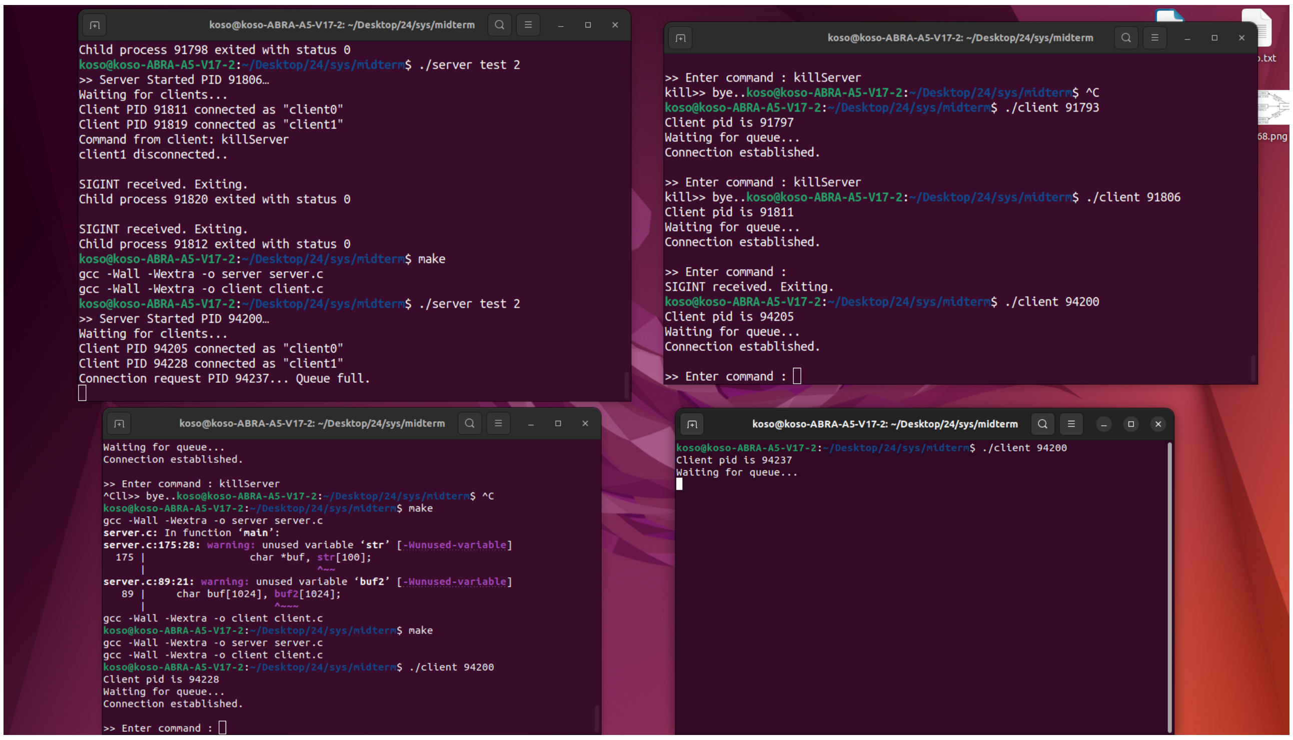
SIGINT handler, removes all semaphores of all clients, server's semaphores, client's shared memories and its own shared memory. Then sends a SIGINT signal to every client to let them exit. Then cleans the childs corresponds to those clients. So everything is cleaned up and ready to move on.

3. Missing parts

   - Clients cannot tryConnect, they have to join to the queue.
   - There is no logging operation
   - There is no individual help command for each commands
   - Upload and download doesn't work with complex branches, files have to be in the same path with the client.
   - ArchServer method doesn't exists.

Example Usage That Demonstrates That The System Meets the Requirements:

Max Client number is 2 and Third client waits for one of the clients to exit.

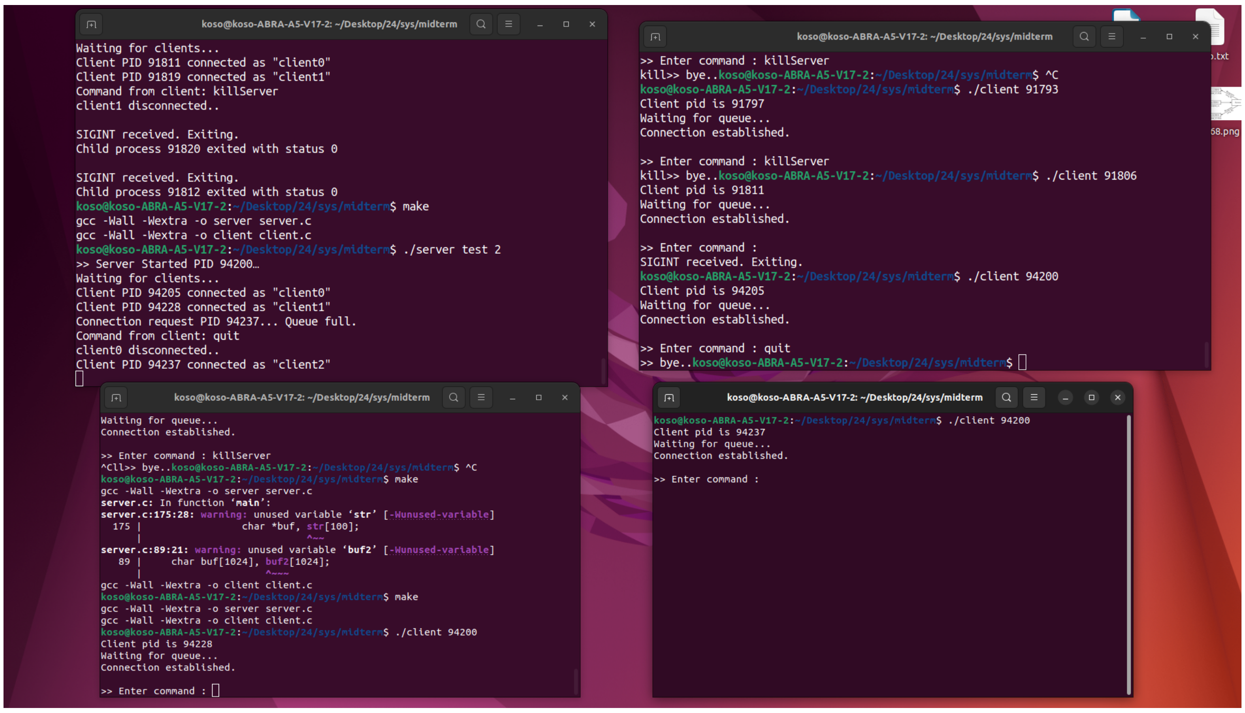


```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/midterm
Child process 91798 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 91806…
Waiting for clients...
Client PID 91811 connected as "client0"
Client PID 91819 connected as "client1"
Command from client: killServer
client1 disconnected..

SIGINT received. Exiting.
Child process 91820 exited with status 0

SIGINT received. Exiting.
Child process 91812 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 94200…
Waiting for clients...
Client PID 94205 connected as "client0"
Client PID 94228 connected as "client1"
Connection request PID 94237... Queue full.
```

```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/midterm

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91793
Client pid is 91797
Waiting for queue...
Connection established.

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91806
Client pid is 91811
Waiting for queue...
Connection established.

>> Enter command :
SIGINT received. Exiting.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94205
Waiting for queue...
Connection established.

>> Enter command :
```

```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/midterm
Waiting for queue...
Connection established.

>> Enter command : killServer
^Cll>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
server.c: In function 'main':
server.c:175:28: warning: unused variable 'str' [-Wunused-variable]
  175 |                 char *buf, str[100];
      |                            ^~~
server.c:89:21: warning: unused variable 'buf2' [-Wunused-variable]
   89 |     char buf[1024], buf2[1024];
      |                     ^~~~
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94228
Waiting for queue...
Connection established.

>> Enter command :
```

```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/midterm
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94237
Waiting for queue...
```



```
Waiting for clients...
Client PID 91811 connected as "client0"
Client PID 91819 connected as "client1"
Command from client: killServer
client1 disconnected..

SIGINT received. Exiting.
Child process 91820 exited with status 0

SIGINT received. Exiting.
Child process 91812 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 94200…
Waiting for clients...
Client PID 94205 connected as "client0"
Client PID 94228 connected as "client1"
Connection request PID 94237... Queue full.
Command from client: quit
client0 disconnected..
Client PID 94237 connected as "client2"
```

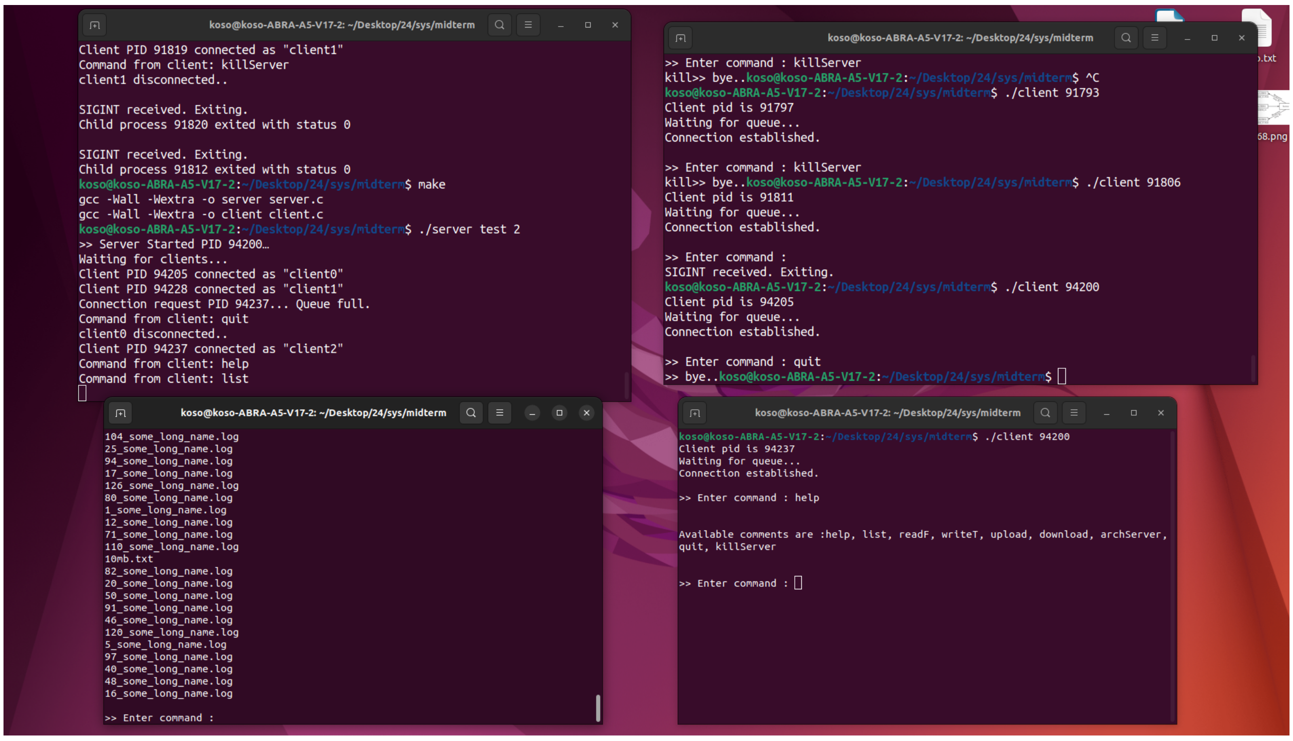
```
>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91793
Client pid is 91797
Waiting for queue...
Connection established.

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91806
Client pid is 91811
Waiting for queue...
Connection established.

>> Enter command :
SIGINT received. Exiting.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94205
Waiting for queue...
Connection established.

>> Enter command : quit
>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$
```

```
Waiting for queue...
Connection established.

>> Enter command : killServer
^Cll>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
server.c: In function ‘main’:
server.c:175:28: warning: unused variable ‘str’ [-Wunused-variable]
  175 |                  char *buf, str[100];
      |                             ^~~
server.c:89:21: warning: unused variable ‘buf2’ [-Wunused-variable]
   89 |     char buf[1024], buf2[1024];
      |                     ^~~~
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94228
Waiting for queue...
Connection established.

>> Enter command :
```

```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94237
Waiting for queue...
Connection established.

>> Enter command :
```

First client gone, third client in.

Help and list commands:



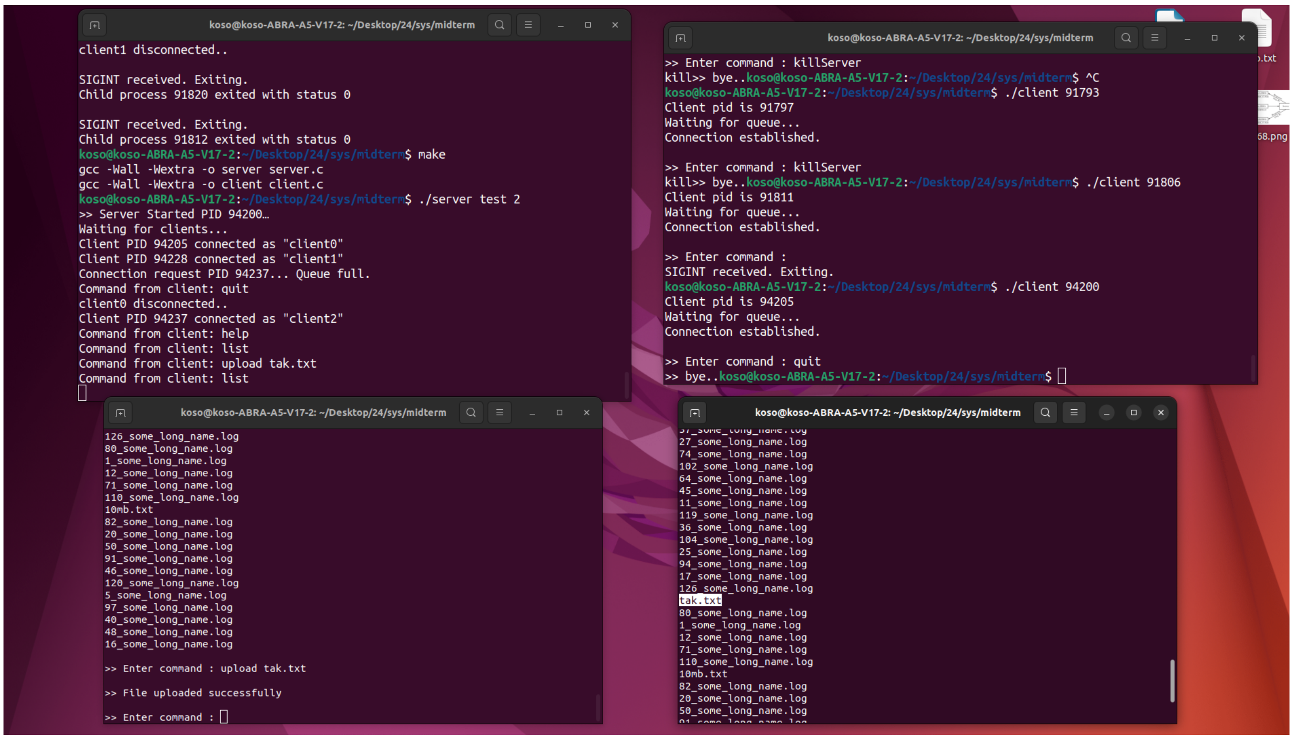
```
Client PID 91819 connected as "client1"
Command from client: killServer
client1 disconnected..

SIGINT received. Exiting.
Child process 91820 exited with status 0

SIGINT received. Exiting.
Child process 91812 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 94200…
Waiting for clients...
Client PID 94205 connected as "client0"
Client PID 94228 connected as "client1"
Connection request PID 94237... Queue full.
Command from client: quit
client0 disconnected..
Client PID 94237 connected as "client2"
Command from client: help
Command from client: list
```

```
>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91793
Client pid is 91797
Waiting for queue...
Connection established.

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91806
Client pid is 91811
Waiting for queue...
Connection established.

>> Enter command :
SIGINT received. Exiting.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94205
Waiting for queue...
Connection established.

>> Enter command : quit
>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$
```

```
104_some_long_name.log
25_some_long_name.log
94_some_long_name.log
17_some_long_name.log
126_some_long_name.log
80_some_long_name.log
1_some_long_name.log
12_some_long_name.log
71_some_long_name.log
110_some_long_name.log
10mb.txt
82_some_long_name.log
20_some_long_name.log
50_some_long_name.log
91_some_long_name.log
46_some_long_name.log
120_some_long_name.log
5_some_long_name.log
97_some_long_name.log
40_some_long_name.log
48_some_long_name.log
16_some_long_name.log

>> Enter command :
```

```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94237
Waiting for queue...
Connection established.

>> Enter command : help


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


>> Enter command :
```

upload:



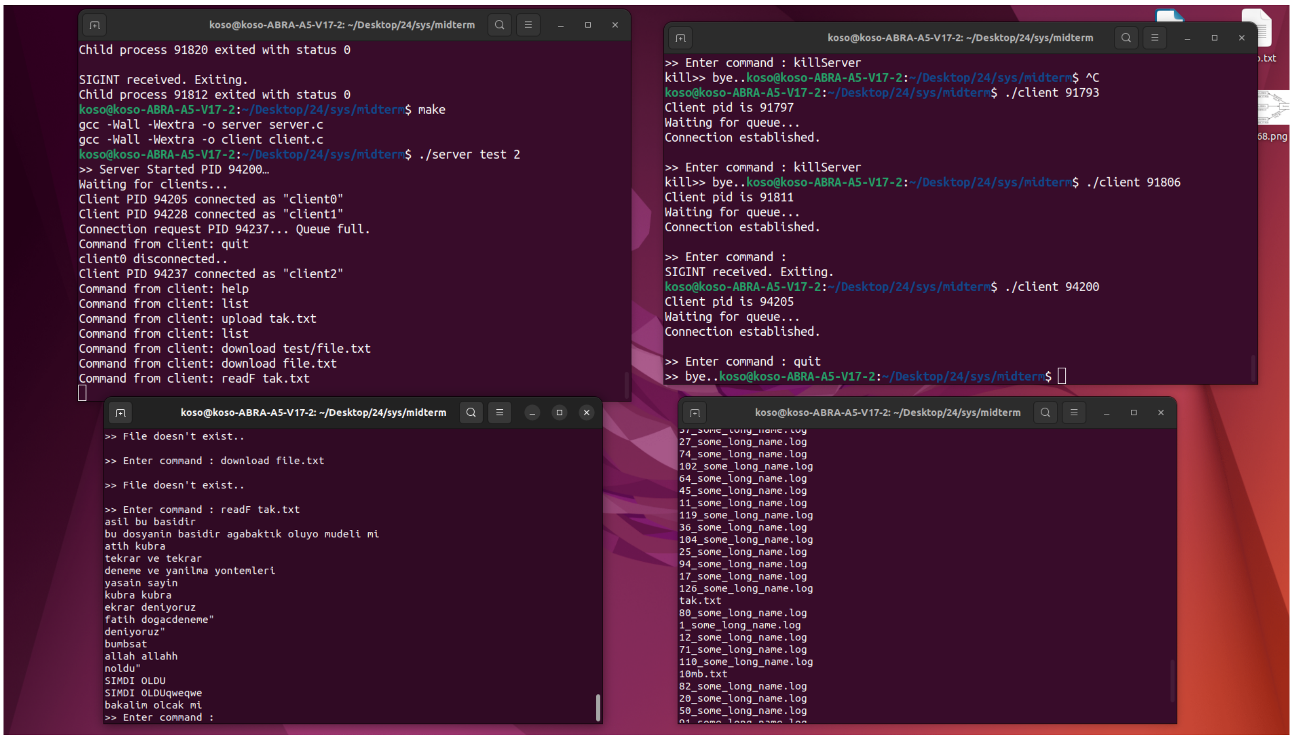
```
client1 disconnected..

SIGINT received. Exiting.
Child process 91820 exited with status 0

SIGINT received. Exiting.
Child process 91812 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 94200…
Waiting for clients...
Client PID 94205 connected as "client0"
Client PID 94228 connected as "client1"
Connection request PID 94237... Queue full.
Command from client: quit
client0 disconnected..
Client PID 94237 connected as "client2"
Command from client: help
Command from client: list
Command from client: upload tak.txt
Command from client: list
```

```
>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91793
Client pid is 91797
Waiting for queue...
Connection established.

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91806
Client pid is 91811
Waiting for queue...
Connection established.

>> Enter command :
SIGINT received. Exiting.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94205
Waiting for queue...
Connection established.

>> Enter command : quit
>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$
```

```
126_some_long_name.log
80_some_long_name.log
1_some_long_name.log
12_some_long_name.log
71_some_long_name.log
110_some_long_name.log
10mb.txt
82_some_long_name.log
20_some_long_name.log
50_some_long_name.log
91_some_long_name.log
46_some_long_name.log
120_some_long_name.log
5_some_long_name.log
97_some_long_name.log
40_some_long_name.log
48_some_long_name.log
16_some_long_name.log

>> Enter command : upload tak.txt

>> File uploaded successfully

>> Enter command :
```

```
27_some_long_name.log
74_some_long_name.log
102_some_long_name.log
64_some_long_name.log
45_some_long_name.log
11_some_long_name.log
119_some_long_name.log
36_some_long_name.log
104_some_long_name.log
25_some_long_name.log
94_some_long_name.log
17_some_long_name.log
126_some_long_name.log
tak.txt
80_some_long_name.log
1_some_long_name.log
12_some_long_name.log
71_some_long_name.log
110_some_long_name.log
10mb.txt
82_some_long_name.log
20_some_long_name.log
50_some_long_name.log
```

readF:



```
Child process 91820 exited with status 0

SIGINT received. Exiting.
Child process 91812 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ make
gcc -Wall -Wextra -o server server.c
gcc -Wall -Wextra -o client client.c
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./server test 2
>> Server Started PID 94200…
Waiting for clients...
Client PID 94205 connected as "client0"
Client PID 94228 connected as "client1"
Connection request PID 94237... Queue full.
Command from client: quit
client0 disconnected..
Client PID 94237 connected as "client2"
Command from client: help
Command from client: list
Command from client: upload tak.txt
Command from client: list
Command from client: download test/file.txt
Command from client: download file.txt
Command from client: readF tak.txt
```

```
>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ^C
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91793
Client pid is 91797
Waiting for queue...
Connection established.

>> Enter command : killServer
kill>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 91806
Client pid is 91811
Waiting for queue...
Connection established.

>> Enter command :
SIGINT received. Exiting.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$ ./client 94200
Client pid is 94205
Waiting for queue...
Connection established.

>> Enter command : quit
>> bye..koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/midterm$
```

```
>> File doesn't exist..

>> Enter command : download file.txt

>> File doesn't exist..

>> Enter command : readF tak.txt
asil bu basidir
bu dosyanin basidir agabaktık oluyo mudeli mi
atih kubra
tekrar ve tekrar
deneme ve yanilma yontemleri
yasain sayin
kubra kubra
ekrar deniyoruz
fatih dogacdeneme"
deniyoruz"
bumbsat
allah allahh
noldu"
SIMDI OLDU
SIMDI OLDUqweqwe
bakalim olcak mi
>> Enter command :
```

```
27_some_long_name.log
74_some_long_name.log
102_some_long_name.log
64_some_long_name.log
45_some_long_name.log
11_some_long_name.log
119_some_long_name.log
36_some_long_name.log
104_some_long_name.log
25_some_long_name.log
94_some_long_name.log
17_some_long_name.log
126_some_long_name.log
tak.txt
80_some_long_name.log
1_some_long_name.log
12_some_long_name.log
71_some_long_name.log
110_some_long_name.log
10mb.txt
82_some_long_name.log
20_some_long_name.log
50_some_long_name.log
```