Mehmet Acar

1801042095

CSE 344 – HW4 Report

System architecture, Design decisions, and Implementation details

BiboServer.c

My server code can handle multiple client connections and respond to their requests. The server communicates with clients using FIFOs (named pipes) and shared memory. In my design, each thread handles with connected clients to the server. In thread function, in the first loop, thread continue to take new connected client request. In the second loop, thread takes a command from the connected client.

Here is a summary of the server code:

- The code includes several necessary header files for handling system calls, signals, file operations, inter-process communication, and directory operations.
- It defines two structures: struct request and struct response which are used for communication between the server and clients. Each structure contains a process ID (PID) and a message.
- The code defines several constants and variables to manage FIFOs, file descriptors, semaphores, and client counts.
- The `handle_signal` function is a signal handler for catching SIGINT (Ctrl+C) , SIGTSTP(Ctrl+Z) and SIGQUIT(Ctrl+\) signals. It unlinks the FIFOs and closes the file pointer when the server is terminated.
- The create_fifo() function creates a FIFO with the given name.
- The client_handle() function is responsible for handling client requests. It takes a request message and a pointer to the response structure. Based on the request, it performs various operations such as displaying available commands, listing files in the directory, reading a specific line from a file, writing to a file, uploading a file to the server, downloading a file from the server, quitting the client, and killing the server.
- The log_client_process() function is used to log the client's process ID, request, and response to a file.

- The main function is the entry point of the program. It takes command-line arguments for the directory name, the maximum number of clients and pool size. In my design, maximum number of clients must not be bigger than pool size. Therefore, I check that maximum number of clients is bigger than pool size. If it is, I print the error message and finish the program. Otherwise, main program creates the directory if it doesn't exist and changes the current working directory to the specified directory. Then, it creates a log file which holds the pids of connected clients and its requests and responses.
- The main function sets up a signal handler to catch SIGINT, SIGTSTP and SIGQUIT signals. Then, it creates thread pool in loop according to pool size value. It also creates a FIFO for the server to listen for client connections.Later, it creates shared memory for inter-process communication between the server and clients.
- Inside the main loop, the server waits for client connections by opening the server FIFO. If server takes connection request from client, I check that server has maximum client number at that time between of mutex lock and mutex unlock operation. If server reaches maximum client number, this client waits until a spot becomes available with pthread_cond_wait() function in order to wait until pthread_cond_signal() function is called from thread when one client is disconnected, or leaves without waiting according to connection option. In order to use pthread_cond_wait() function, I also initialize condition variable as a global variable.   The reason of I used mutex is preventing connect more than one waiting clients to the server when one client is disconnected. Then, I printed connected client pid and number. Then, I send connection response as a "Success" to the client. Then, I used condition variable in order to send pthread_cond_signal() to the available thread which waits for taking client command from connected clients. This pthread_cond_signal() function is called between of mutex lock and mutex unlock operation.
- In thread function which is start_thread(), in the first loop, I used mutex lock and unlock operation and between mutex lock and unlock operation, thread is waiting for new client connects to the server and waiting operation is done with pthread_cond_wait() function. Also, I take connected client pid and according to this, I initialize two FIFO names. Then, in the second loop, thread takes command request from one FIFO and then calls the client_handle() function to process the request and prepare the response, and writes the response back to the FIFO. Also, I called log_client_process() function in order to write client pid, command request and response of this command to the server log file. If client sends a "quit" or SIGINT(Ctrl+C) , SIGTSTP(Ctrl+Z), SIGQUIT(Ctrl+\)

message to the server, thread is exiting from the inner loop and prints disconnected client number. Then, I used mutex lock and unlock operation and between mutex lock and unlock operation, I decrement current working client number in server and sends pthread_cond_signal() operation to the waited clients for connecting to the server in main loop.

- The server process keeps accepting new client connections until it reaches the maximum number of clients specified in the command-line argument or until it is terminated by a signal.
- When the server is terminated, it unlinks the FIFOs and closes the log file.

BiboClient.c

My client code communicates with a server using named pipes (FIFOs) and shared memory. It allows the user to send requests to the server and receive responses.

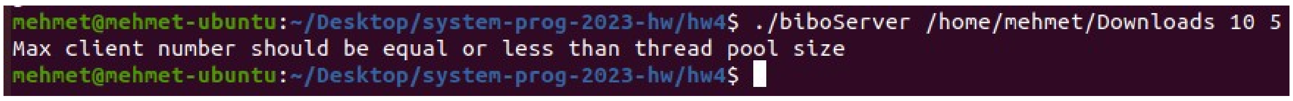
Here's a summary of the client code:

1. The code includes necessary header files and defines some constants and variables.
2. The handle_signal() function is a signal handler for handling the SIGINT(Ctrl+C) , SIGTSTP(Ctrl+Z) and SIGQUIT(Ctrl+\) signals. It sends a termination request to the server, waits for the server's response, closes file descriptors, unlinks FIFOs, and exits the program.
3. The create_fifo() function creates a FIFO with the given name.
4. The log_client_process() function logs the client's PID, request, and response into a file.
5. The `main` function is the entry point of the program.
6. The program expects two command-line arguments: the request type ("Connect" or "tryConnect") and the server's PID.
7. The program sets up a signal handler using sigaction to handle SIGINT, SIGTSTP and SIGQUIT signals.
8. It creates a log file for the client, using the PID to generate a unique filename.
9. It creates two FIFOs for the client: client_connect_fifo and client_command_fifo.

10. It sends a connection request (struct request) to the server by opening the server's FIFO (server_connect_fifo) and writing the request. The request includes the client's PID and the request type.
11. The program waits for the server's response by opening the client's FIFO (client_connect_fifo) and reading the response (struct response).
12. If the response message is "Exit", it means the server rejected the connection request or closed the connection. In this case, the program unlinks the FIFOs and exits.
13. Otherwise, the program enters a loop to handle user commands. It prompts the user for a command and sends a request to the server. It waits for the server's response and logs the client's request and response.
14. Inside the loop, the program also handles specific commands, such as "upload" and "download". For "upload", it reads the contents of the specified file and writes them to shared memory. For "download", it writes the contents of shared memory to a file.
15. If the server's response is "quit" or "killServer", it means the server is terminating the connection. In this case, the program closes file descriptors, unlinks FIFOs, and exits.
16. Finally, the program cleans up shared memory, detaches it, and removes it using shmdt() and shmctl().

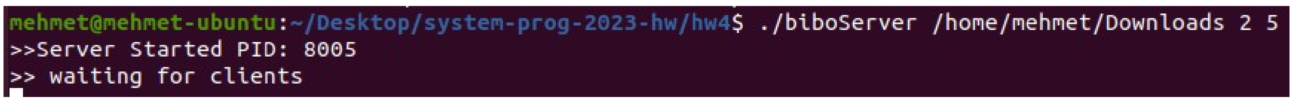
Test Cases and Results

1- If given max client number is bigger than thread pool size



```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 10 5
Max client number should be equal or less than thread pool size
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$
```

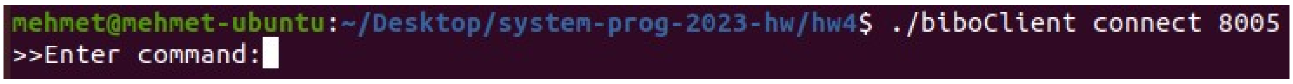
2- Server waits for client connection



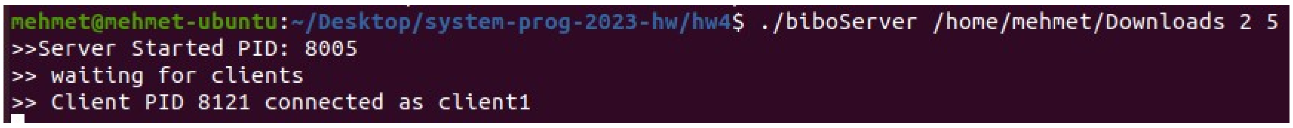
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 8005
>> waiting for clients
```

Server enters the /*home*/mehmet/Downloads directory and defines max of clients number as 2 and thread pool size as 5.

3- Client connects to server with ‘connect’ and ‘tryconnect’ option

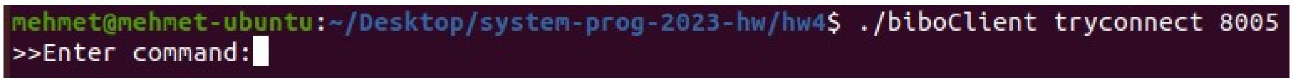


```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboClient connect 8005
>>Enter command:
```

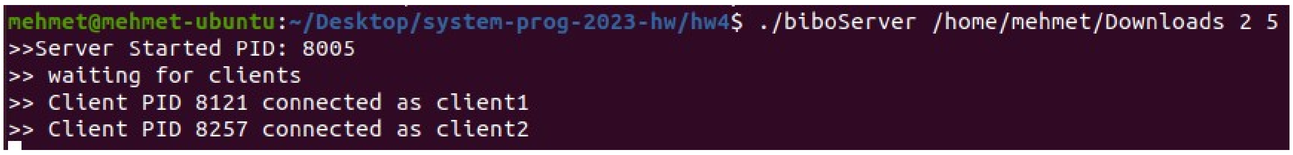


```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 8005
>> waiting for clients
>> Client PID 8121 connected as client1
```

Client which has PID 8121 connected to server and server has 1 client now.



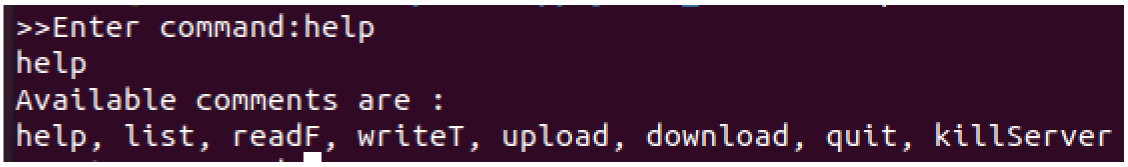
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboClient tryconnect 8005
>>Enter command:
```



```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 8005
>> waiting for clients
>> Client PID 8121 connected as client1
>> Client PID 8257 connected as client2
```

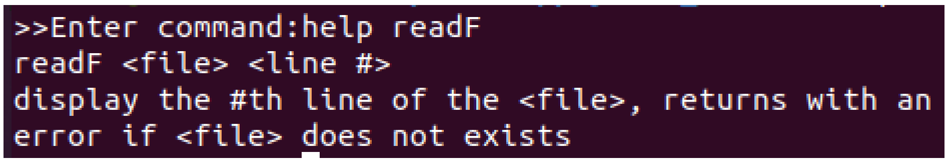
Client which has PID 8257 connected to server and server has 2 client now. So, server reaches the maximum number of clients.

4- help Command



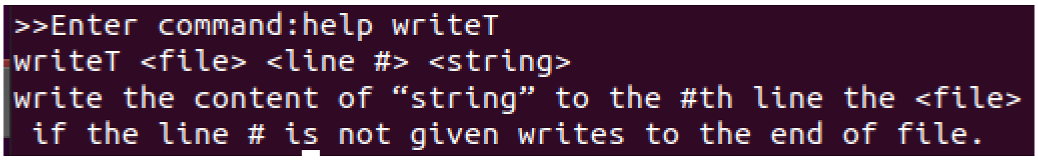
```
>>Enter command:help
help
Available comments are :
help, list, readF, writeT, upload, download, quit, killServer
```

5- help readF Command



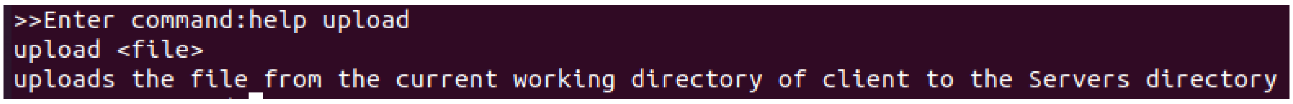
```
>>Enter command:help readF
readF <file> <line #>
display the #th line of the <file>, returns with an
error if <file> does not exists
```

6- help writeT Command



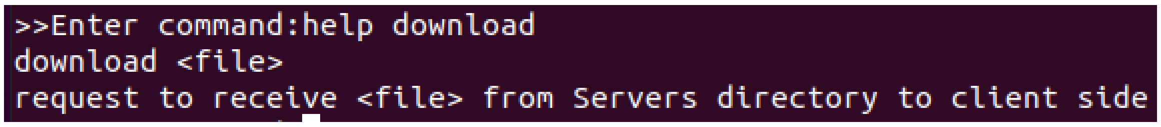
```
>>Enter command:help writeT
writeT <file> <line #> <string>
write the content of “string” to the #th line the <file>
 if the line # is not given writes to the end of file.
```

7- help upload Command



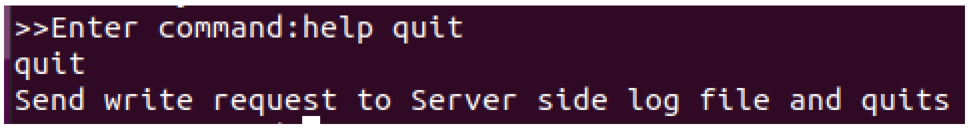
```
>>Enter command:help upload
upload <file>
uploads the file from the current working directory of client to the Servers directory
```

8- help download Command



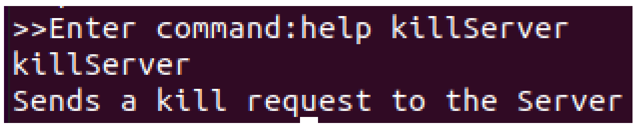
```
>>Enter command:help download
download <file>
request to receive <file> from Servers directory to client side
```

9- help quit



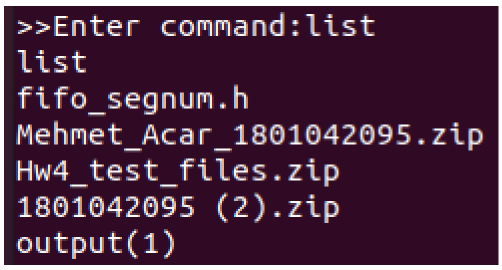
```
>>Enter command:help quit
quit
Send write request to Server side log file and quits
```

10- help killServer



```
>>Enter command:help killServer
killServer
Sends a kill request to the Server
```

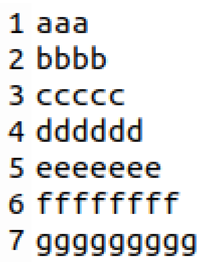
11- list Command



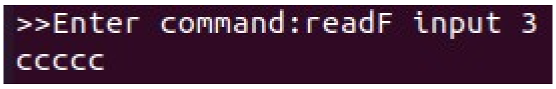
```
>>Enter command:list
list
fifo_segnum.h
Mehmet_Acar_1801042095.zip
Hw4_test_files.zip
1801042095 (2).zip
output(1)
```

12- readF command with given line number and except line number

Before I show the results this command, I want to share the contents of input file.

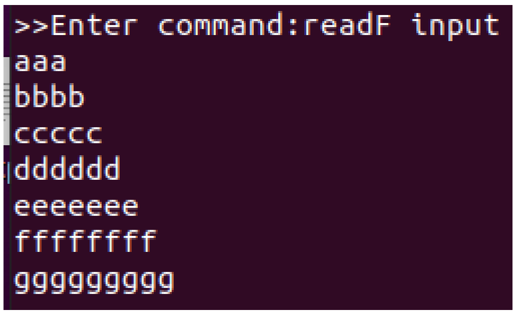


```
1 aaa
2 bbbb
3 ccccc
4 dddddd
5 eeeeeee
6 ffffffff
7 ggggggggg
```



```
>>Enter command:readF input 3
ccccc
```

If client gives input filename and line number which are “input” and 3 , third line of “input” file is printed as a “ccccc”.

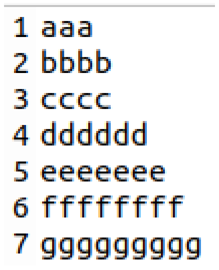


```
>>Enter command:readF input
aaa
bbbb
ccccc
dddddd
eeeeeee
ffffffff
ggggggggg
```

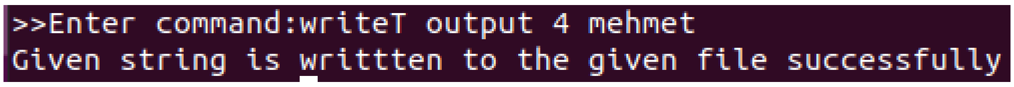
If client gives only input filename which is “input” and not give line number, whole contents of “input” file is printed.

13- write T command with given line number and except line number

Before I show the results this command, I want to share the contents of output file.



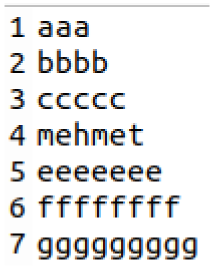
```
1 aaa
2 bbbb
3 cccc
4 dddddd
5 eeeeeee
6 ffffffff
7 ggggggggg
```



```
>>Enter command:writeT output 4 mehmet
Given string is writtten to the given file successfully
```

If client gives output filename ,line number and written string which are “output” , 4 and “mehmet”, “mehmet” string is written to the fourth line of output file.

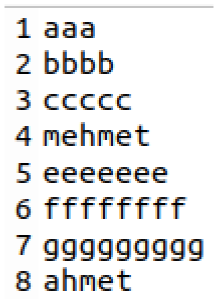
In order to show that, I share the current contents of output file.



```
1 aaa
2 bbbb
3 ccccc
4 mehmet
5 eeeeeee
6 ffffffff
7 ggggggggg
```

If client gives output filename and written string which are “output” and “ahmet”, “ahmet” string is written to the end of the output file.

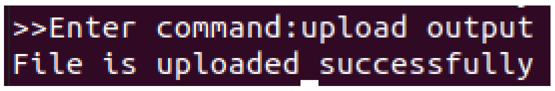
In order to show that, I share the current contents of output file.



```
1 aaa
2 bbbb
3 ccccc
4 mehmet
5 eeeeeee
6 ffffffff
7 ggggggggg
8 ahmet
```

14- Upload Command

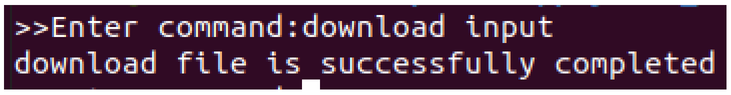
This command uploads the given file from client working directory to the server working directory. You can check the uploaded file in server working directory.



```
>>Enter command:upload output
File is uploaded successfully
```

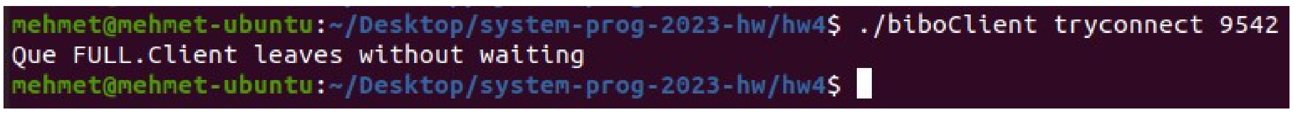
15 – Download Command

This command downloads the given file from server working directory to the client working directory. You can check the downloaded file in client working directory.



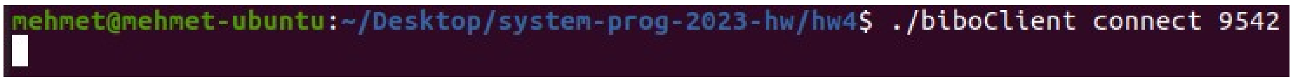
```
>>Enter command:download input
download file is successfully completed
```

16- Client wants to connect server when server has maximum client number



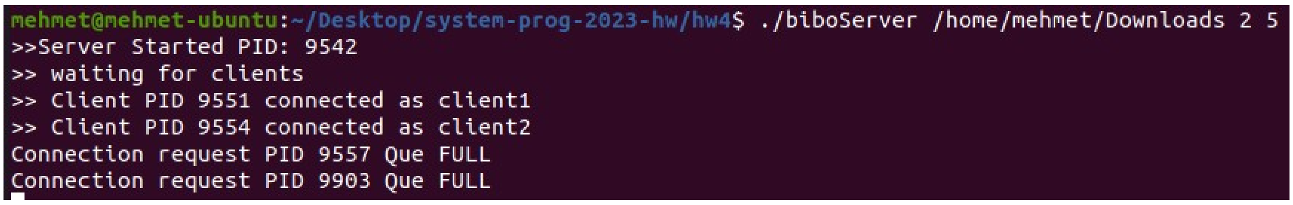
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboClient tryconnect 9542
Que FULL.Client leaves without waiting
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$
```

If client wants to connect to the server with tryconnect option, at that time, if server has maximum client number, this client leaves without waiting and server prints the message “Que Full” message with pid of this client.



```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboClient connect 9542
```

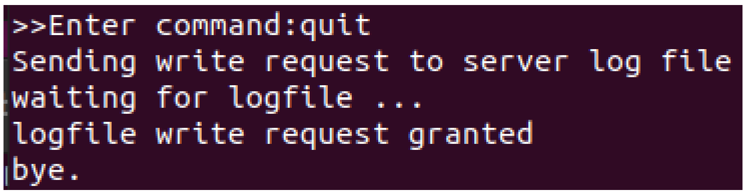
If client wants to connect to the server with connect option, at that time, if server has maximum client number, this client waits until a spot becomes available and server prints the message “Que Full” message with pid of this client.



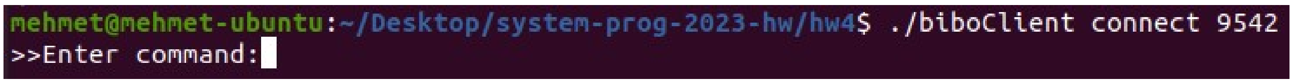
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 9542
>> waiting for clients
>> Client PID 9551 connected as client1
>> Client PID 9554 connected as client2
Connection request PID 9557 Que FULL
Connection request PID 9903 Que FULL
```

17 – quit Command

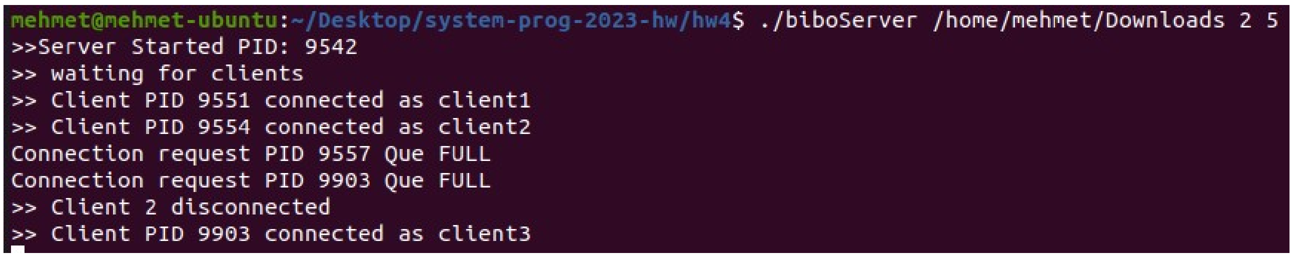
If server has maximum client number and there is a client waits for spot becomes available, if one client quits from the server or one client enters SIGINT(Ctrl+C) , SIGTSTP(Ctrl+Z) or SIGQUIT(Ctrl+\) signal, waiting process can connect to the server and give a command to the server. Also, server prints disconnected client number and connected client’s pid and client number.



```
>>Enter command:quit
Sending write request to server log file
waiting for logfile ...
logfile write request granted
bye.
```



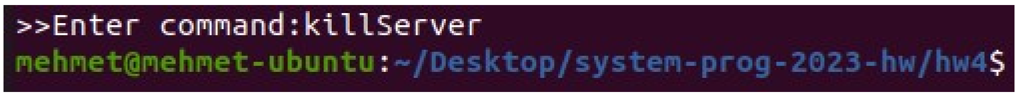
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboClient connect 9542
>>Enter command:
```



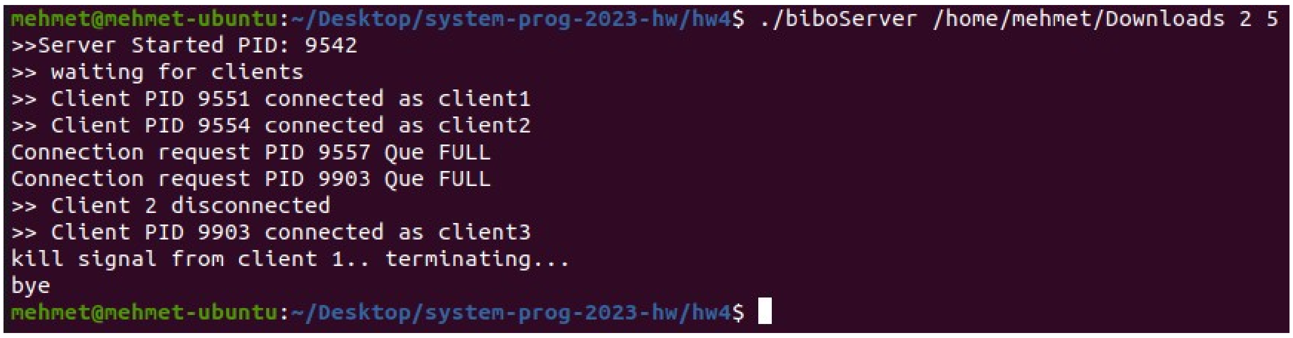
```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 9542
>> waiting for clients
>> Client PID 9551 connected as client1
>> Client PID 9554 connected as client2
Connection request PID 9557 Que FULL
Connection request PID 9903 Que FULL
>> Client 2 disconnected
>> Client PID 9903 connected as client3
```

18- killServer Command

This command kills the server and called client. Server prints the terminating message.

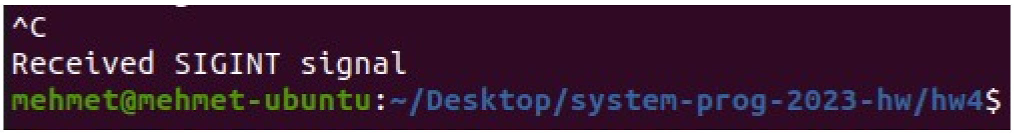


```
>>Enter command:killServer
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$
```



```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$ ./biboServer /home/mehmet/Downloads 2 5
>>Server Started PID: 9542
>> waiting for clients
>> Client PID 9551 connected as client1
>> Client PID 9554 connected as client2
Connection request PID 9557 Que FULL
Connection request PID 9903 Que FULL
>> Client 2 disconnected
>> Client PID 9903 connected as client3
kill signal from client 1.. terminating...
bye
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$
```

19- If User Enters SIGINT(Ctrl+C) , SIGTSTP(Ctrl+Z) or SIGQUIT(Ctrl+\) signal in Server Terminal



```
^C
Received SIGINT signal
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw4$
```

If server takes SIGINT(Ctrl+C) , SIGTSTP(Ctrl+Z) or SIGQUIT(Ctrl+\) signal, server is terminating.

20- Prevent Race Condition With File Locking

I also prevent that multiple processes accessing files simultaneously. For example, if two or more processes enters the file at the same time in writeT command, in order to prevent race condition, I used lock file operation. I used lock file operation for writeT , readF , upload and download commands. With this locking, two or more processes can not enter the critical region at the same time.