



CSE344 – System Programming

Midterm

Report

 Seniha Sena Topkaya
 131044020

First of all i developed the program on macOS. I also tried it on ubuntu to test it. There are some differences. some operations work on macOS but not on ubuntu and there are also differences in the makefile.

I used Fifo, Shared memory and Semaphore in the program. I used Shared memory for communication such as flags, Semaphore for synchronization and Fifo for data transfer. To avoid process conflicts, I used Fifos with their own id numbers for each client. I used the Queue structure to wait for the client id sending a new connection request if the available spots are full.

Part1: biboClient

On the client side, I first checked the argument and if it was entered incorrectly, printed the usage appropriately. There are 3 shared memory and 2 Fifos in this section.

shmServerPid -> This is the shared memory I use to access the server pid number. I use this to test if the arguments are entered correctly. It ensures that the Client connects to the correct Server.

shm_freeSpot-> Shared memory that holds the number of available spots. It is manipulated by the server and controlled by the client. This value is only checked if the client connects with tryConnect. If there are no available spots (which is found by checking shm_freeSpot) the client terminates itself.

Then I added the necessary commands to capture signals, added signals to the signal mask and used sigaction inside the infinite loop.

shm_fd-> This shared memory, I check to make sure that two clients do not send requests at the same time. If the server side is available at that moment, this value is 0, if not, it is 1. If this value checked on the client side is 0, conectFifo writes its id number in the connection fifos and sends it to the server.

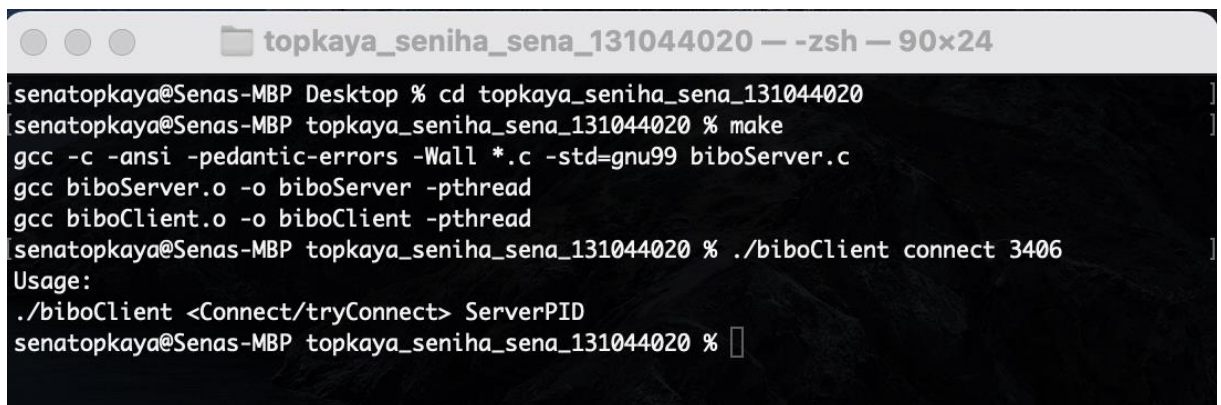
After the connection is established, the Client creates a Fifo containing its id number in order to send the commands to the Server. If the command entered is quit or killServer, the Server is informed and exits. If not, the command is split and sent to the server and the result of the

command is received and printed on the same fifo. At this stage there is a separate control only for readF. This is because if the whole file is to be read, the fifo must be read in an infinite loop. In addition, I implemented a signalHandler function to process the signals on the Client side.

Output:

- **If command line arguments are invalid, print usage information and exit:**

I check all the arguments and if they are invalid then I call printUsage function to print valid input format.



```
senatopkaya@Senas-MBP Desktop % cd topkaya_seniha_sena_131044020
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % make
gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 biboServer.c
gcc biboServer.o -o biboServer -pthread
gcc biboClient.o -o biboClient -pthread
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 3406
Usage:
./biboClient <Connect/tryConnect> ServerPID
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %
```

Part2: biboServer

On the server side, I created a Queue structure to keep the waiting Clients. I checked the argument on input and printed the usage in case of error. For signal operations, I added signals to the signal mask and tried to capture the signals with sigaction in an infinite loop. Here I used 3 shared memory and 2 semaphores.

my_semaphore-> This semaphore is there to synchronize when an element will be removed from the queue (dequeue).

semCh-> This semaphore is there to lock the system after receiving the client pid number in an infinite loop until a child process is created for that client.

shm_fd-> Assigned 0 to allow a connection to be established by the server, 1 if busy.

shm_freeSpot-> This shared memory holds the number of clients connected. It is added to the queue to connect by checking the client side, or if it is tryConnect, it exits without connecting.

shmServerPid-> This shared memory is used to publish the id number of the server.

In this section I used a few fifos for data transfer.

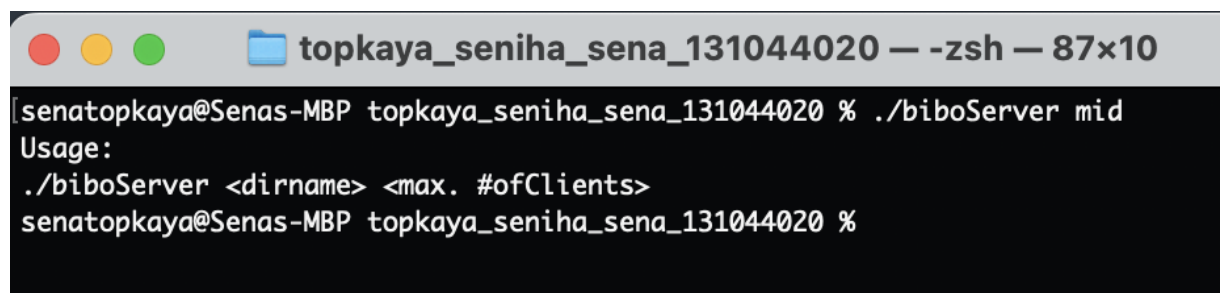
connectFifo-> I used it to get the id number of the client.

When the connection request comes, I wrote 1 in `shm_fd` memory and locked the system by waiting on `semCh` semaphore. If there is no available spot, the client is added to the queue. If there is an available spot, I first connected the client with `connectFifo`. I increased my `numClient` variable that holds the number of clients. I decreased the value in `shm_freeSpot` shared memory by 1. I did `fork()` for the client and created the process that will deal with that client. Then I post `semCh` and unlocked the system because now I can fork for another client. In the child process I first changed the `shm_id` memory to 0 to show that it is available. I created `Fifo (fifo_clientid)` using the client id. Inside the infinite loop I opened the fifo in read mode and received the commands. If the command was quit I increased the `freeSpot` by 1 and called the log file function and exited by writing "bye" to the client side. If the command was `killServer` I increased the `freeSpot` by 1 and called the `killServer` function and exited. If the command was something else I called the `splitCommand` function. Parent process keeps id numbers of child processes in global array. I did this to kill the child processes first in case of kill.

Output:

- If command line arguments are invalid, print usage information and exit:

I check all the arguments and if they are invalid then I call `printUsage` function to print valid input format.

A terminal window titled 'topkaya_seniha_sena_131044020 — -zsh — 87x10'. The prompt is 'senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %'. The user has entered './biboServer mid'. The terminal displays a usage message: 'Usage: ./biboServer <dirname> <max. #ofClients>'. The prompt returns to 'senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %'.

Functions:

`splitCommand():`

This function splits the command according to the space character and stores the arguments in the `str` array. It also makes a copy of the entered command to be used later for `writeT`.

If the first element of `str` is **help**, it calls the `help` function with the second argument.

If the first element of `str` is **list**, it calls the `list` function.

If the first element of str is **readF**, the number of arguments is checked. If there are 2 arguments, readF is called to read the entire file. If there are 3 arguments, readF is called to read a specific line. If the command is incorrect, usage is printed to the client fifo.

If the first element of str is **writeT**, it is split by quotes to get the string to write to the file. the rest of the arguments are also split by spaces. If there are 2 arguments, the given string is written to the end of the file. If there are 3 arguments, the given string is written to the desired line. The writeT function is called. If the command is incorrect, usage is printed to the client fifo.

If the first element of str is **upload**, the upload function is called with the server directory. If the command is incorrect, the usage is printed to the client fifo.

If the first element of str is **download**, the download function is called with the server directory. If the command is incorrect, the usage is printed to the client fifo.

If the command is not defined, an error message is printed to the client fifo.

help():

Writes the correct use of the queried command to the client fifo. If there is no specific command, it writes the supported commands.

list():

Since I cannot use system function, I used fork and exec. After forking, I first duplicated the stdout file descriptor in the child process with dup2. So I redirected the execl output end to the input end of the client fifo. Then I gave the execl function the ls command and got the result. I waited for the child to terminate in the parent process. I closed the fifo and exited.

readF():

First of all, I added the newline character to the last line of the file, since I will be doing line by line operations. I then checked to see if a specific line number was given. If the line number is given, I read the file up to the given line and copied the desired line into the result variable. Then I sent the result to the client fifo.

If there is no line number, I read the whole file in a loop. I sent line by line to client fifo. Here I tried to prevent the conflict using semaphore. I sent the last string "end" to let the client know that the file content is finished.

writeT():

First of all, I created the file if the file does not exist. Then I read the file line by line and when I got to the desired line, I came to the beginning of the line and wrote the given string. If the line number is not given, I wrote the given string at the end of the file. I informed the client.

upload():

Since I cannot use system function, I used fork and exec. First of all, I read the whole file and found the size to give size information to the client fifo. After forking, I called the cp command with execl in the child process. I gave the directory of the Server as an argument to the Execl function. After the child finished its work, I sent information to the client fifo and logged out.

download():

Since I cannot use system function, I used fork and exec. First of all, I read the whole file and found the size to give size information to the client fifo. After forking, I called the cp command with execl in the child process. I gave the directory of the Server as a filepath and gave the current working directory(client directory) as an argument to the Execl function. After the child finished its work, I sent information to the client fifo and logged out.

killServer();

It kills all child processes and finally the parent process and terminates.

initializeQueue():

Function that initializes the queue.

isQueueEmpty():

Function that returns whether the queue is empty or not.

isQueueFull():

Function that returns whether the queue is full or not.

enqueue():

Function to add an element to the queue.

dequeue():

Function to remove elements from the queue.

printQueue():

Function to display the contents of the queue.

killHandler():

It kills all child processes and finally the parent process and terminates.

signalHandler():

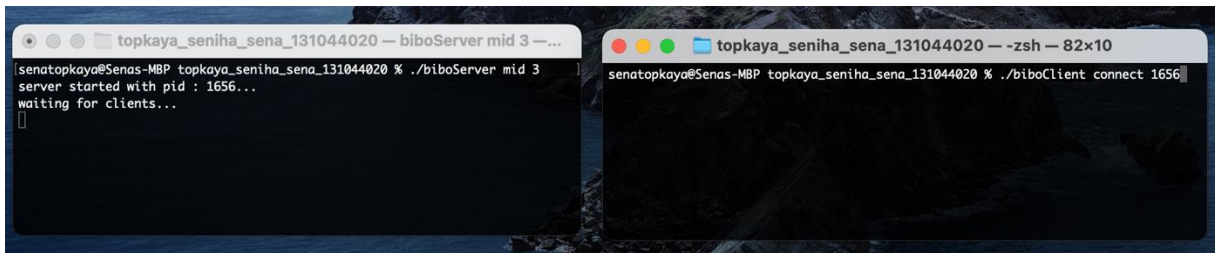
When the signal is caught, the **killHandler** function is called to do cleanup and exit.

logFile():

It is triggered when the quit command is received. Using the client id number and child process id number, it writes the client termination information to the log file.

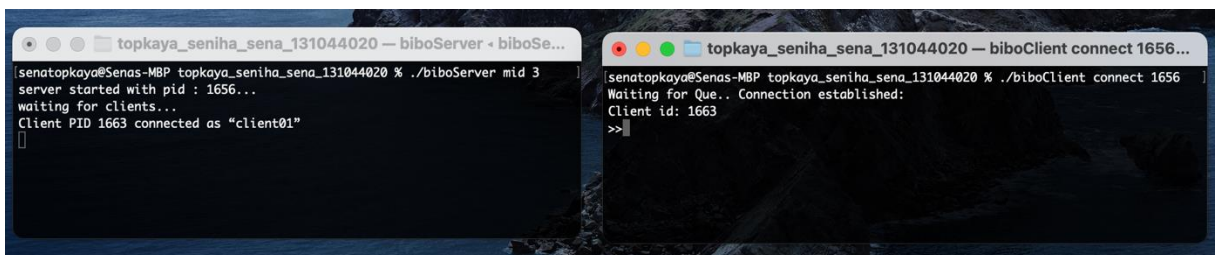
Test Cases:

- Server waiting for connection



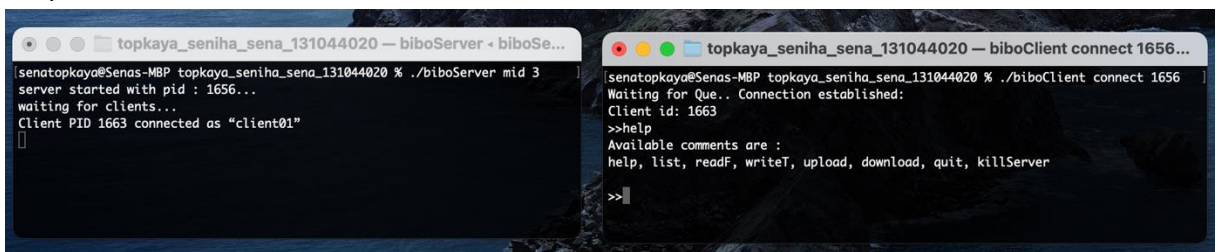
The image shows two terminal windows. The left window, titled 'topkaya_seniha_sena_131044020 — biboServer mid 3 —...', shows the server command './biboServer mid 3' being executed. The output is 'server started with pid : 1656...' followed by 'waiting for clients...' and a blank prompt. The right window, titled 'topkaya_seniha_sena_131044020 — -zsh — 82x10', shows the client command './biboClient connect 1656' being executed, with the output 'Waiting for Que..' and a blank prompt.

- Client connected



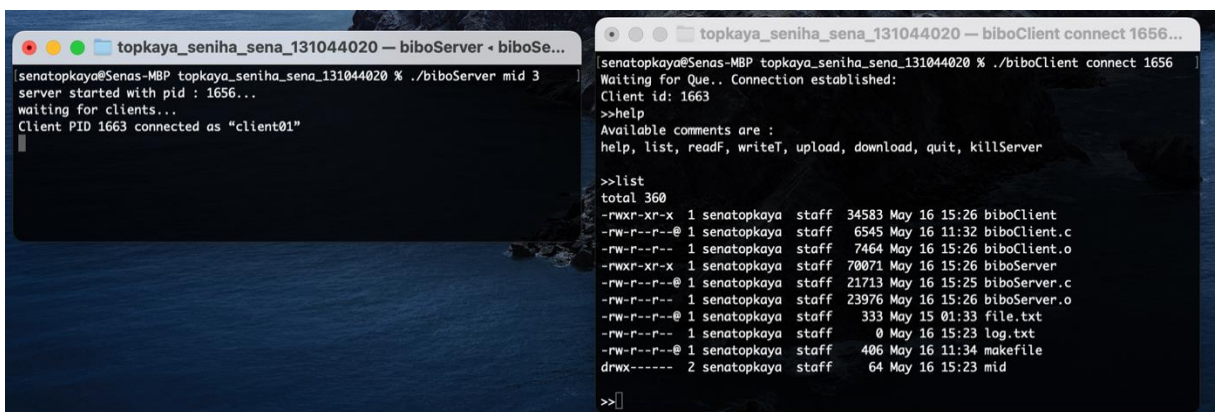
The image shows two terminal windows. The left window shows the server output: 'waiting for clients...' followed by 'Client PID 1663 connected as "client01"' and a blank prompt. The right window shows the client output: 'Waiting for Que.. Connection established: Client id: 1663' followed by a blank prompt.

- Help command test



The image shows two terminal windows. The left window shows the server output: 'waiting for clients...' followed by 'Client PID 1663 connected as "client01"' and a blank prompt. The right window shows the client output: 'Waiting for Que.. Connection established: Client id: 1663' followed by '>>help' and the output 'Available comments are : help, list, readF, writeT, upload, download, quit, killServer' followed by a blank prompt.

- List command test



The image shows two terminal windows. The left window shows the server output: 'waiting for clients...' followed by 'Client PID 1663 connected as "client01"' and a blank prompt. The right window shows the client output: 'Waiting for Que.. Connection established: Client id: 1663' followed by '>>help' and the output 'Available comments are : help, list, readF, writeT, upload, download, quit, killServer'. Then, the client enters '>>list' and the output shows a table of files and directories:

total	360
-rwxr-xr-x	1 senatopkaya staff 34583 May 16 15:26 biboClient
-rw-r--r--	@ 1 senatopkaya staff 6545 May 16 11:32 biboClient.c
-rw-r--r--	1 senatopkaya staff 7464 May 16 15:26 biboClient.o
-rwxr-xr-x	1 senatopkaya staff 70071 May 16 15:26 biboServer
-rw-r--r--	@ 1 senatopkaya staff 21713 May 16 15:25 biboServer.c
-rw-r--r--	1 senatopkaya staff 23976 May 16 15:26 biboServer.o
-rw-r--r--	@ 1 senatopkaya staff 333 May 15 01:33 file.txt
-rw-r--r--	1 senatopkaya staff 0 May 16 15:23 log.txt
-rw-r--r--	@ 1 senatopkaya staff 406 May 16 11:34 makefile
drwx-----	2 senatopkaya staff 64 May 16 15:23 mid

The client prompt is '>>'.

- Help with argument command test

The image shows two terminal windows. The left window is titled 'topkaya_seniha_sena_131044020 — biboServer • biboSe...' and shows the server starting with PID 1656. The right window is titled 'topkaya_seniha_sena_131044020 — biboClient connect 1656...' and shows the client's help command output:

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

>>help writeT
writeT <file> <line #> <string> :
request to write the content of "string" to the #th line the <file>,
if the line # is not given

>>help list
list
display the list of files in Servers directory

>>help upload
upload <file>
uploads the file from the current working directory of client to the Servers direc
tory

>>|
```

- Invalid command test

The image shows two terminal windows. The left window is titled 'topkaya_seniha_sena_131044020 — biboServer • biboSe...' and shows the server starting with PID 1656. The right window is titled 'topkaya_seniha_sena_131044020 — biboClient connect 1656...' and shows the client entering invalid commands:

```
>>readF
readF <file> <line>
>>writeT
writeT <file> <line> <string>
>>upload
upload <file>
>>download
download <file>
>>|
```

- readF command test with line argument

The image shows two terminal windows. The left window is titled 'topkaya_seniha_sena_131044020 — biboServer • biboSe...' and shows the server starting with PID 1927. The right window is titled 'topkaya_seniha_sena_131044020 — biboClient connect 1927...' and shows the client using the readF command:

```
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 1927
Waiting for Que.. Connection established:
>>readF file.txt 2
What about rain?

>>|
```

- readF command test

The image shows two terminal windows. The left window is titled 'topkaya_seniha_sena_131044020 — biboServer • biboSe...' and shows the server starting with PID 1966. The right window is titled 'topkaya_seniha_sena_131044020 — biboClient connect 1966...' and shows the client using the readF command:

```
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 1966
Waiting for Que.. Connection established:
>>readF file.txt
What about sunrise?
What about rain?
What about all the things
That you said we were to gain?
What about killing fields?
Is there a time?
What about all the things
That you said was yours and mine?
Did you ever stop to notice
All the blood we've shed before?
Did you ever stop to notice
This crying Earth, these weeping shores?

>>|
```

- writeT command test with and without line

The image shows two terminal windows. The left window is titled 'topkaya_seniha_sena_131044020 — biboServer • biboSe...' and shows the server starting with PID 1966. The right window is titled 'topkaya_seniha_sena_131044020 — biboClient connect 1966...' and shows the client using the writeT command:

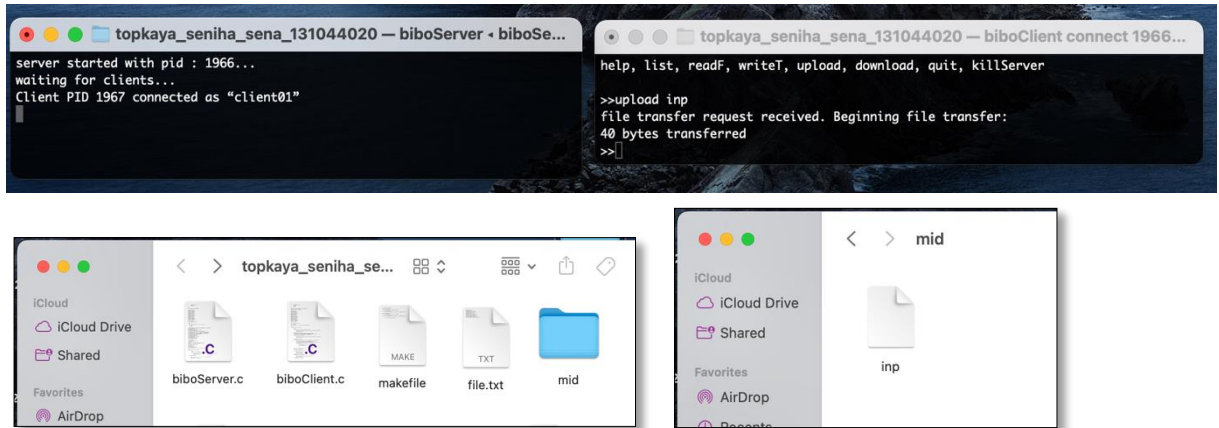
```
>>writeT inp 1 "this is a string"
this is a string <- that string was written to the given file.
>>writeT inp "here is another string"
here is another string <- that string was written to the given file.

>>|
```

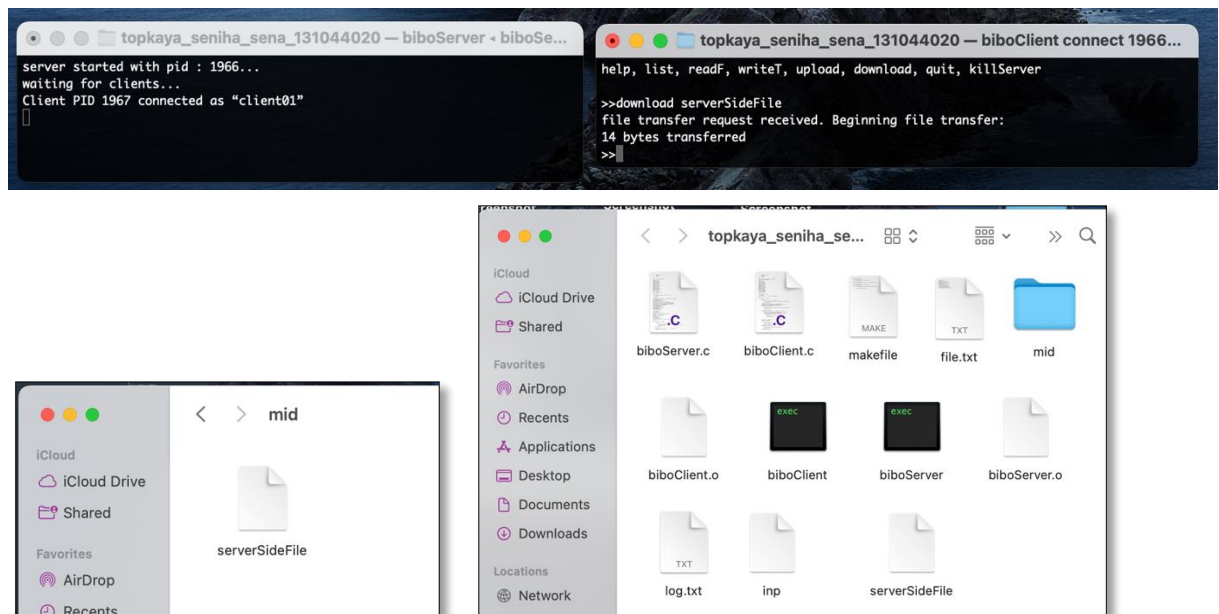
Below the terminal windows, there is a separate window titled 'inp' showing the contents of the file:

```
this is a string
here is another string
```

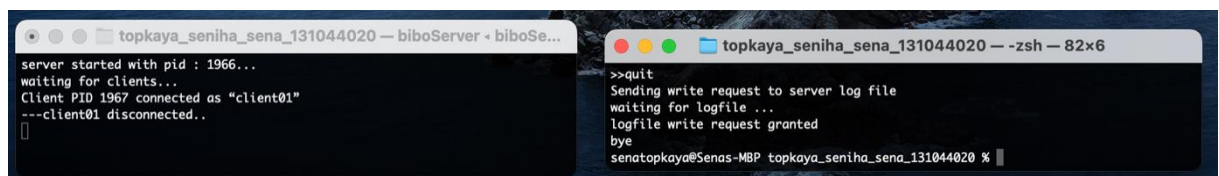

- Upload command test



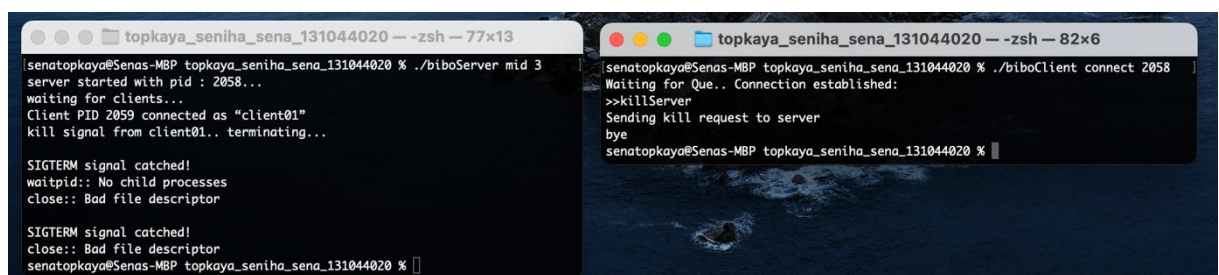
- Download command test



- Quit command test

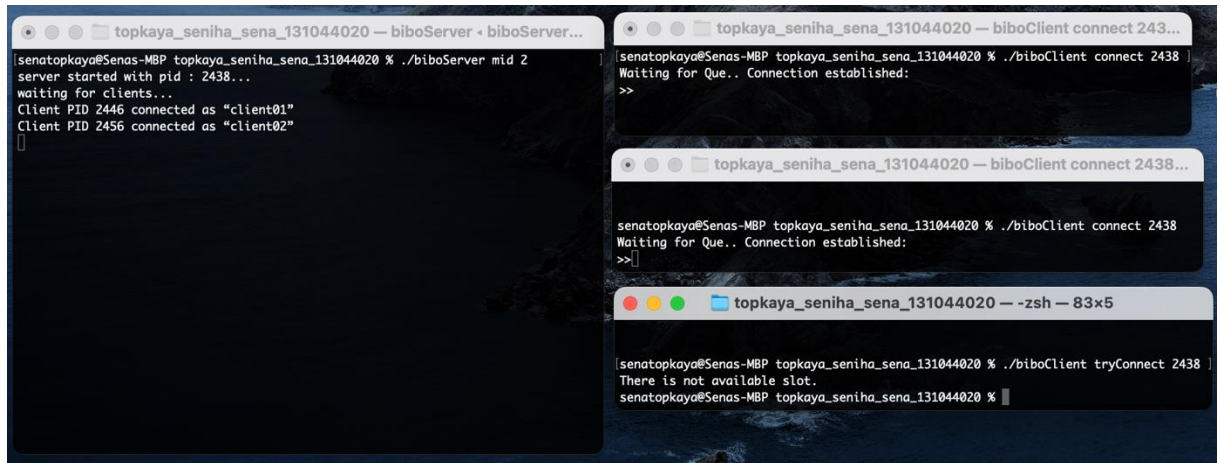


- killServer command test



Multiple connections

- tryConnect test



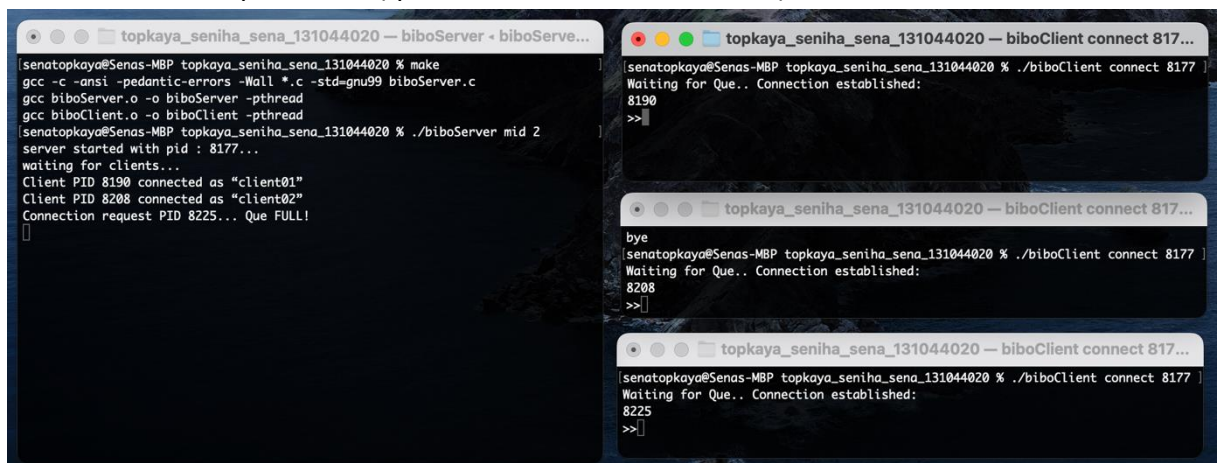
```
topkaya_seniha_sena_131044020 — biboServer • biboServer...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboServer mid 2
server started with pid : 2438...
waiting for clients...
Client PID 2446 connected as "client01"
Client PID 2456 connected as "client02"
]

topkaya_seniha_sena_131044020 — biboClient connect 2438...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 2438
Waiting for Que.. Connection established:
>>]

topkaya_seniha_sena_131044020 — biboClient connect 2438...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 2438
Waiting for Que.. Connection established:
>>]

topkaya_seniha_sena_131044020 — zsh — 83x5
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient tryConnect 2438
There is not available slot.
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ]
```

- The client added queue test (queue element does not work)



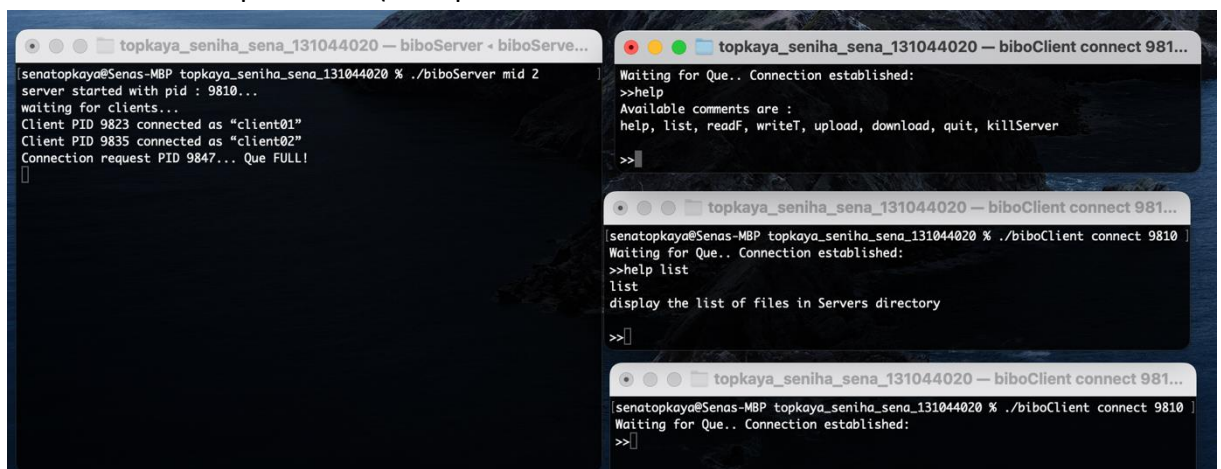
```
topkaya_seniha_sena_131044020 — biboServer • biboServe...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % make
gcc -c -ansi -pedantic-errors -Wall *.c -std-gnu99 biboServer.c
gcc biboServer.o -o biboServer -pthread
gcc biboClient.o -o biboClient -pthread
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboServer mid 2
server started with pid : 8177...
waiting for clients...
Client PID 8190 connected as "client01"
Client PID 8208 connected as "client02"
Connection request PID 8225... Que FULL!
]

topkaya_seniha_sena_131044020 — biboClient connect 8177...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 8177
Waiting for Que.. Connection established:
8190
>>]

topkaya_seniha_sena_131044020 — biboClient connect 8177...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 8177
Waiting for Que.. Connection established:
8208
>>]

topkaya_seniha_sena_131044020 — biboClient connect 8177...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 8177
Waiting for Que.. Connection established:
8225
>>]
```

- The client added queue test (multiple clients can work at the same time)



```
topkaya_seniha_sena_131044020 — biboServer • biboServe...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboServer mid 2
server started with pid : 9810...
waiting for clients...
Client PID 9823 connected as "client01"
Client PID 9835 connected as "client02"
Connection request PID 9847... Que FULL!
]

topkaya_seniha_sena_131044020 — biboClient connect 9810...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 9810
Waiting for Que.. Connection established:
>>help
Available comments are :
help, list, readF, writeT, upload, download, quit, killServer
>>]

topkaya_seniha_sena_131044020 — biboClient connect 9810...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 9810
Waiting for Que.. Connection established:
>>help list
list
display the list of files in Servers directory
>>]

topkaya_seniha_sena_131044020 — biboClient connect 9810...
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 9810
Waiting for Que.. Connection established:
>>]
```

- The client added queue test (one client disconnected and dequeue worked)

```

senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboServer mid 2
server started with pid : 9810...
waiting for clients...
Client PID 9823 connected as "client01"
Client PID 9835 connected as "client02"
Connection request PID 9847... Que FULL!
---client01 disconnected..
Client PID 9847 connected as "client02"

>>quit
Sending write request to server log file
waiting for logfile...
logfile write request granted
bye
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %

senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 9810
Waiting for Que.. Connection established:
>>help list
list
display the list of files in Servers directory
>>

senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % ./biboClient connect 9810
Waiting for Que.. Connection established:
>>help
Available comments are :
help, list, readF, writeT, upload, download, quit, killServer
>>

```

General rules:

- Signal handler

```

topkaya_seniha_sena_131044020 — -zsh — 82x5

readF <file> <line>
>>^C

SIGINT signal caught!
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %

```

- Makefile with make clean that just compile the program not run for all source files.
(macos does not accept -lrt but ubuntu needs it)

```

all: biboServer biboClient

biboServer: biboServer.o
    gcc biboServer.o -o biboServer -pthread

biboServer.o: biboServer.c
    gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 biboServer.c

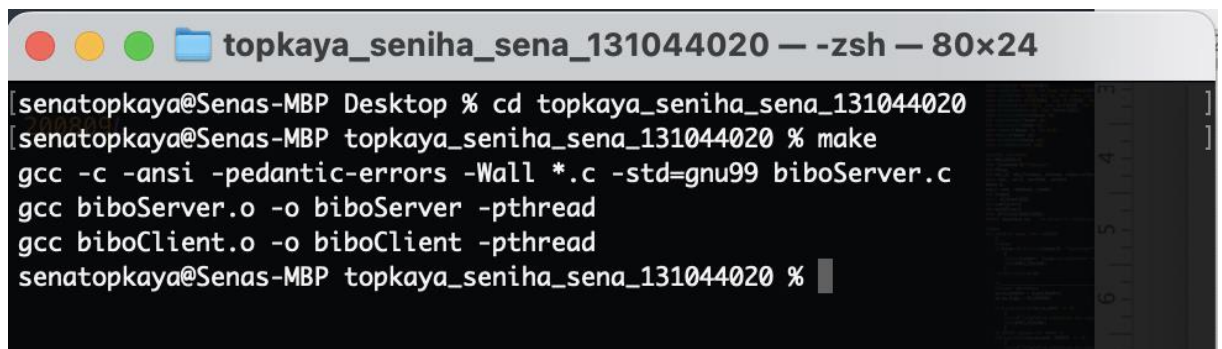
biboClient: biboClient.o
    gcc biboClient.o -o biboClient -pthread

biboClient.o: biboClient.c
    gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 biboClient.c

clean:
    rm -f biboServer biboClient *.o

```

- There is no compilation error



```
topkaya_seniha_sena_131044020 — -zsh — 80x24
[senatopkaya@Senas-MBP Desktop % cd topkaya_seniha_sena_131044020
[senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 % make
gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 biboServer.c
gcc biboServer.o -o biboServer -pthread
gcc biboClient.o -o biboClient -pthread
senatopkaya@Senas-MBP topkaya_seniha_sena_131044020 %
```

End of the report