Gebze Technical University

Departman of Computer Engineering

Spring 2023 - CSE 344

Midterm Project Report

GONCA EZGİ ÇAKIR

151044054

1. Solution Approach

In summary, this program is implemented as a basic server client architecture. Communication between client and server provided by *server FIFO*s. Client processes handled by *children processes* on the server side by communicating through *client FIFOs* and the synchronization between processes are provided by s*hared memory, counting semaphores* and *binary semaphores (mutex)*. In order to prevent mutual exclusion and race condition *reader - writer paradigm* is implemented by binary semaphores. *Signals* are handled on both client and server sides.

1.1. Client Side

Client connects to server by **./biboClient Connect/tryConnect serverPid** arguments. If the connection succeeds, client *sends request* to server such as help, list, upload, download, readF, writeT, quit, killServer and gets response till quited or terminated by SIGINT.

Client program implementation contents are listed below.

- Stores the argument in a struct name as *BiboClient*. Connect or tryConnect as *connectInfo*, client process id as *clientPid*, server process id as *serverPid and creates the client's fifo.*
- Client opens the server fifo by using serverPid variable and sends the biboClient struct to server side, then waits for response to start commanding.
- *If there is no space* in the client queue and the connecInfo is "*Connect"* then client *waits for another client to quit,* till then it can send any request to server.

  If *there is no space* in the client queue and the connecInfo is "*tryConnect*" then server sends SIGUSR1 signal in order to *terminate that client.*

  If t*here is a space* in the client queue on the server side, client receives an information on the read end side of the server fifo. Then s*tarts to write requests to server* fifo's write side and wait for response again on the server fifo's read side.
- While client waits for a response server firstly sends the response size, client reads that information and allocates space for the response then server writes the response then client reads this response and stores to print to screen. All these read and write processes are provided on server fifo.
- If the client quits frees all its recources and terminates. If it receives SIGINT signal by Ctrl-C it writes to server fifo for the last time but changes the connectInfo to "kill" in order to handle closing all clients and server on the server side; frees all its recources and terminates.

Error Handling

- Given serverPid checked, if there is no fifo then the client receives and SIGUSR1 and terminates.
- If there is more or less argument than three client program prints an error message on screen and terminates.

1.2. Server Side

Client connects to server by **./biboServer directoryName maxClient** arguments. Server gets client connections and handles them by children processes, communicates through client fifos and sends releated responses to each client. It synchoronizes all resources and children processes till receives killServer request or terminated by SIGINT .

Server program implementation contents are listed below.

- Stores the argument in a struct name as *BiboServer*. Directory name as dirName and max client number as *maxClient also creates the server fifo.*
- Server directory created/opened then shared memory for children process counting semaphores, binary semaphores (mutex), counter variables is created and initalized. Client queue is created as seperate shared memory and initialized.
- Server starts to wait for client connection if *there is a space* in the client queue and the then *client connects and starts to send requests*.

  If *there* i*s no space* in the client queue and the connecInfo is "*tryConnect*" then client can't connect, server sends SIGUSR1 signal in order to t*erminate that client*.

  If *there is no space* in the client queue and the connecInfo is "*Connect*" then client *waits for another client to quit*, till then it can send any request to server.
- When a client connects the client counter is incremented and the client id created in order to print server results.

Semaphore Usage

- Counting semaphore *sem_clients* is set to maxClient amount at the beginning; called *sem_wait()* each time a client connected and called *sem_post()* when a client disconnected.
- Binary semaphore *sem_queue* is set to 1 amount at the beginning; called *sem_wait()* each time before a queue is manupulated and called *sem_post()* after.
- Binary semaphore *sem_logfile* is set to 1 amount at the beginning; called *sem_wait()* each time before a logfile is manupulated and called *sem_post()* after.

- Binary semaphores *readTry, rmutex, rsc, wmutex* set to 1 amount at the beginning; and used in the reader writer paradigm when a file is readed or writed. For the *upload* and *download* commands *writer* is applied; for the *readF reader* is applied.

Command Details

- **help:** sets the response with a list of commands as a string and returns the string.
- **list:** lists the current directory content by using dirent struct and adds to a string till it reaches directory '.' or '..' and returns the string.
- **help + command:** sets the response with the related command's explanation as a string and returns the string
- **upload**: sets the source to client directory and destination to server directory. Reads the source file and creates the file in the destination and writes into it. If the file is already exist at the destination or not exist at the source it sets an error message as response.
- **download**: sets the source to server directory and destination to client directory. Reads the source file and creates the file in the destination and writes into it. If the file is already exist at the destination or not exist at the source it sets an error message as response.
- **readF**: reads the given file's content, sets into a string and returns the string. If there a line number given it only sets that line's content to string and retuns the string. NOT SUPPORTED FOR LARGE FILES.
- **writeT**: NOT SUPPORTED.
- **quit**: details written to log file and child process ends, clients end when it requested to quit.
- **killServer**: calls signal handler in order to terminate all client processes, children processes and server process itself. All the chilren and client process pids previously stored in an array, so they are terminated according to these content.

Error Handling

- If there is more or less argument than three server program prints an error message on screen and terminates.
- Double instance is control by creating a temp file and removing at the end of the server process. So this temp file occurency at the beginning of the program; if it occurs that means server process already started; otherwise it is the only instance.

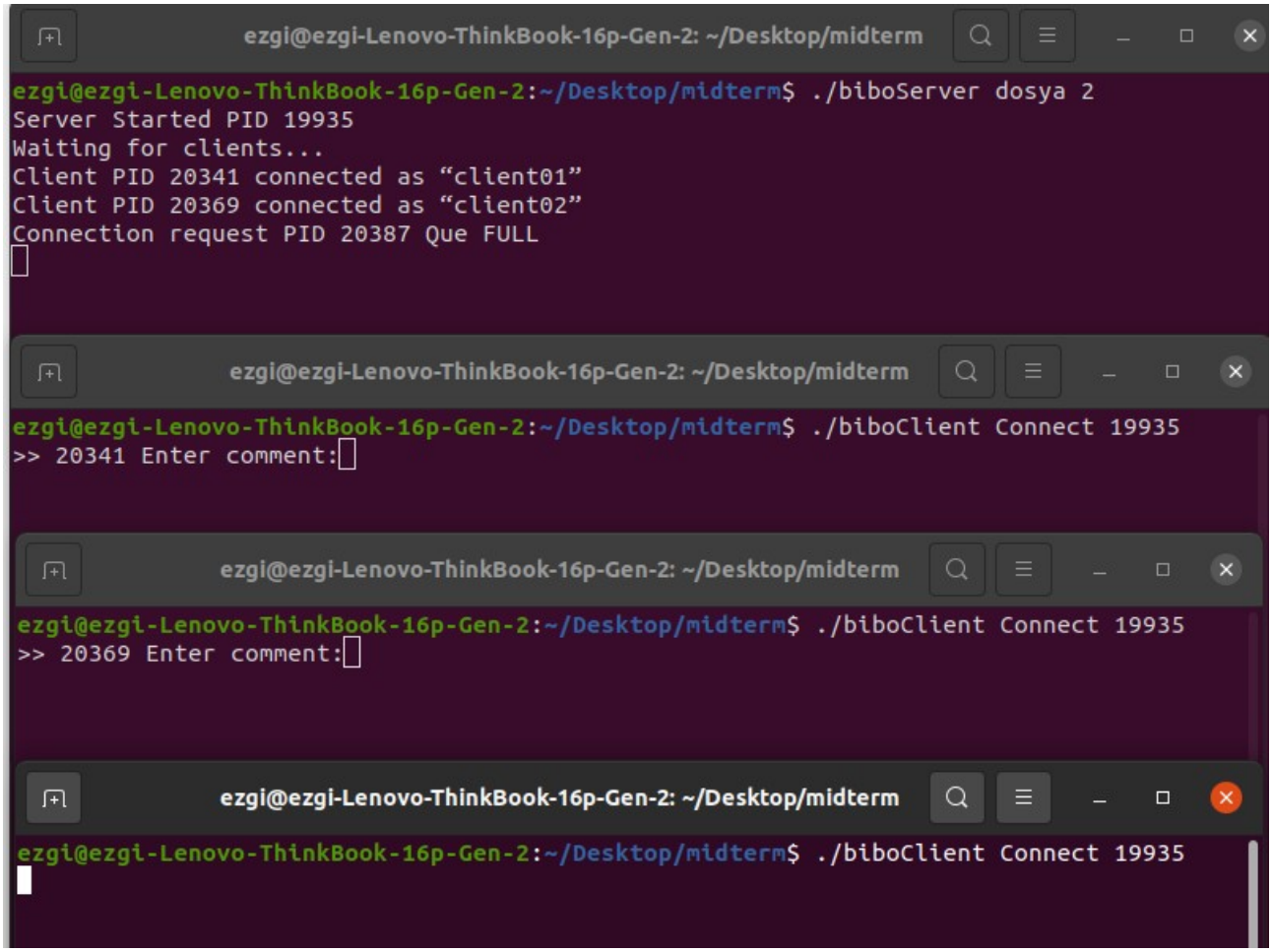
- Log file access is restricted fro clients by checking the filename argument at the beginning of the upload, download or readF commands. If the filename equals to log file name then request receives and related error message.

2. Test Results

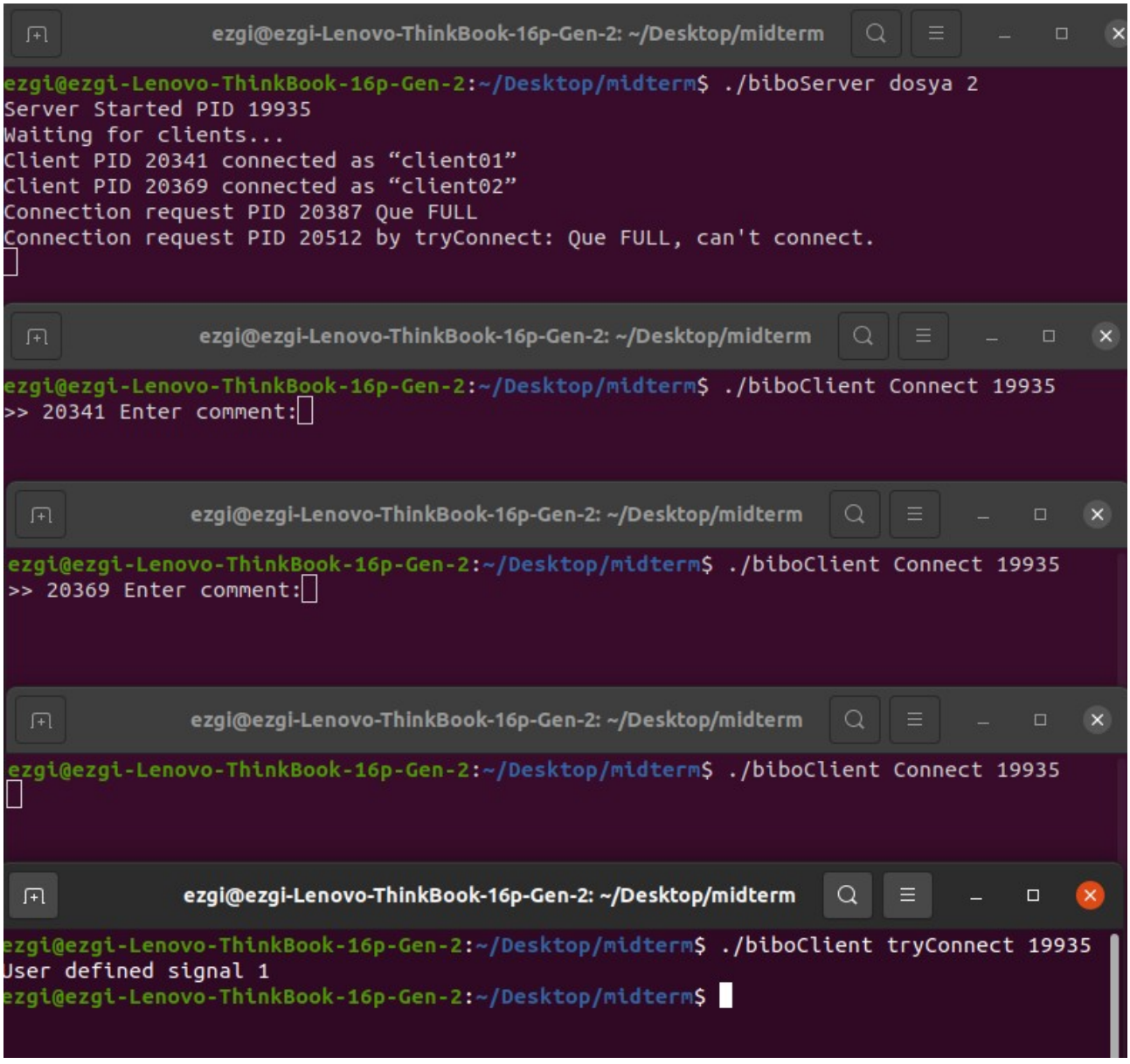
Multiple test case senarios applied and screenshots are added. Below you will see some program input result.

NOTE: Client Pids printed before 'Enter comment' line just to show the test result much clearly.

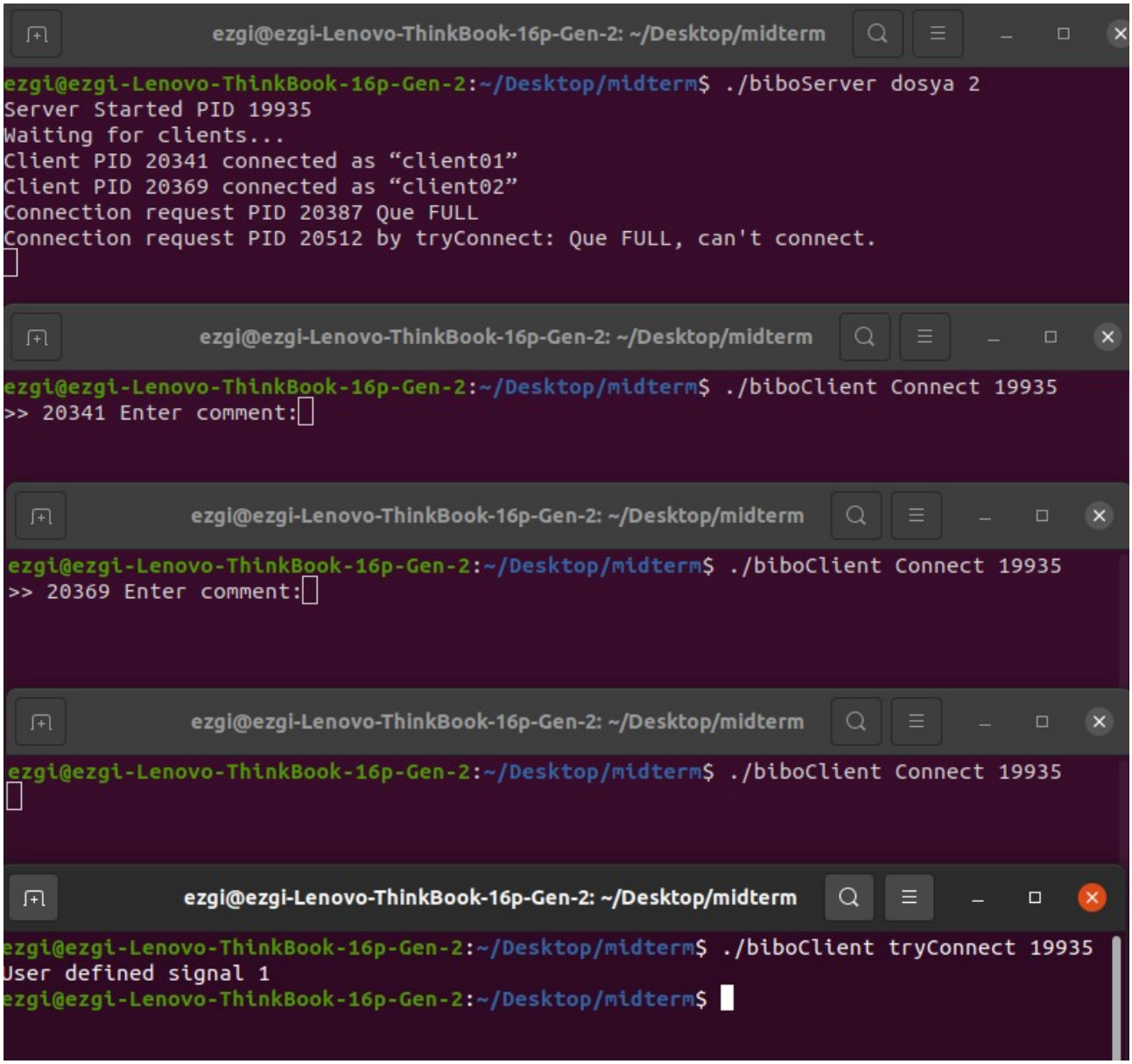
2.1. Client Queue is Full, New Client Comes With 'Connect'



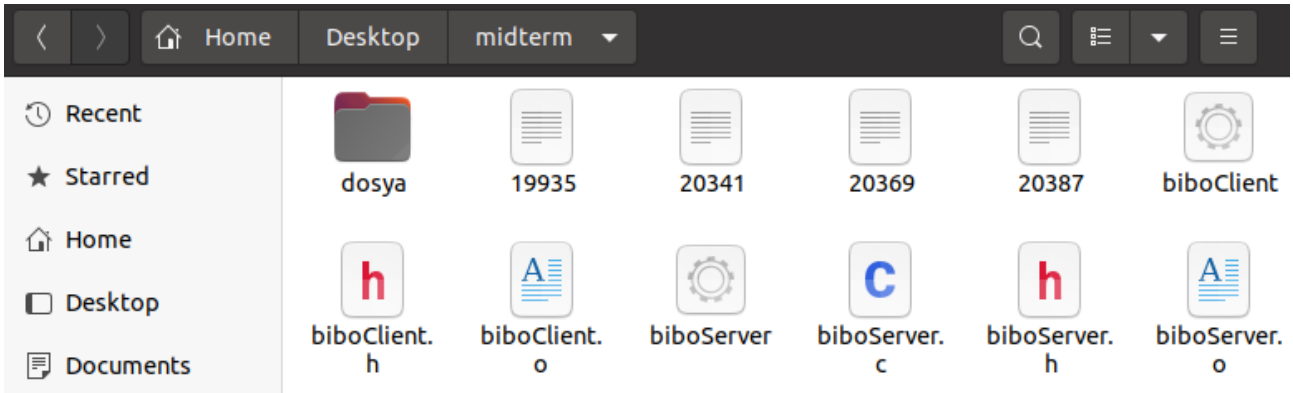
2.2. Client Queue is Full, New Client Comes With ‘tryConnect’

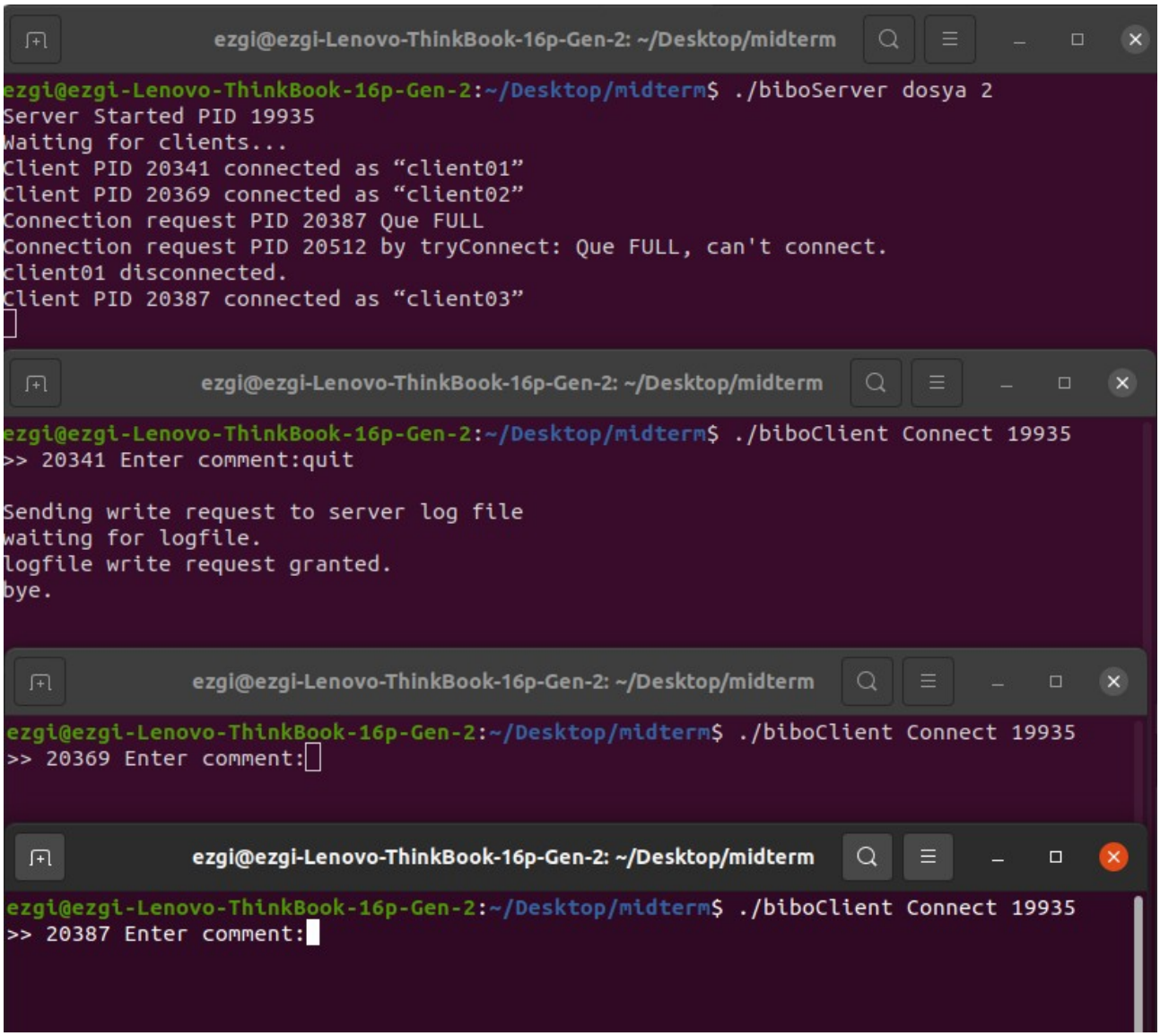


2.3. When a Client Quits, Another Client From Queue is Started

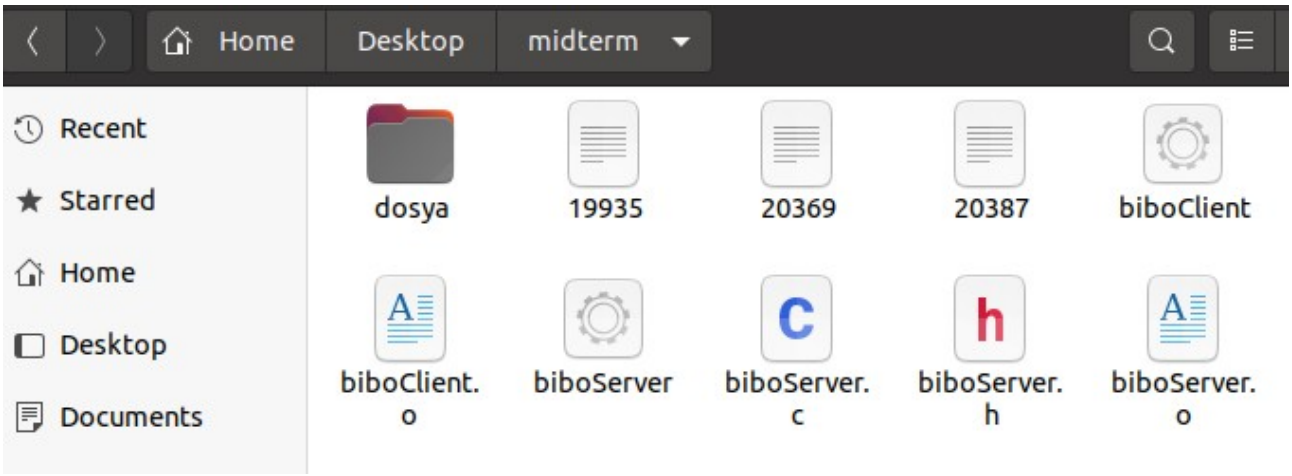


FIFOs Before Terminating



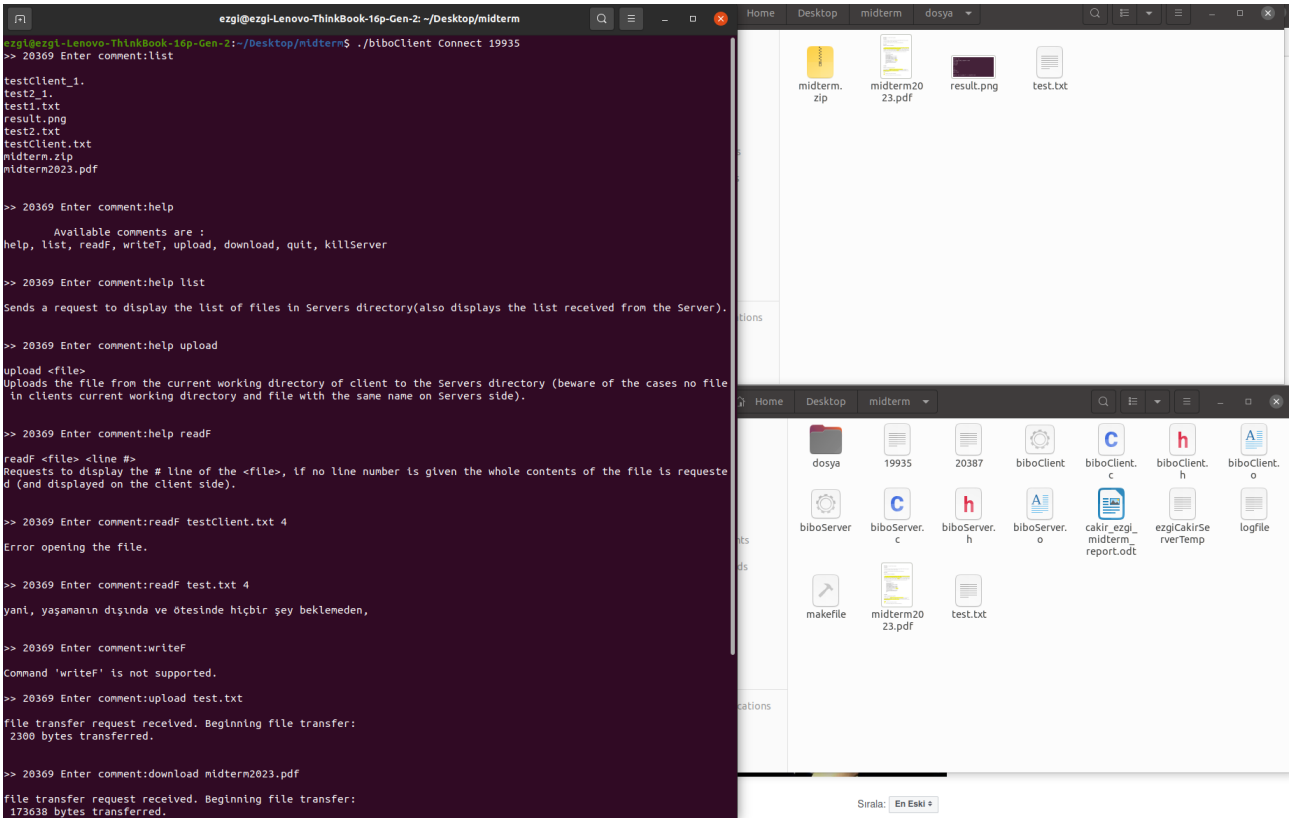


FIFOs After Terminating One Client

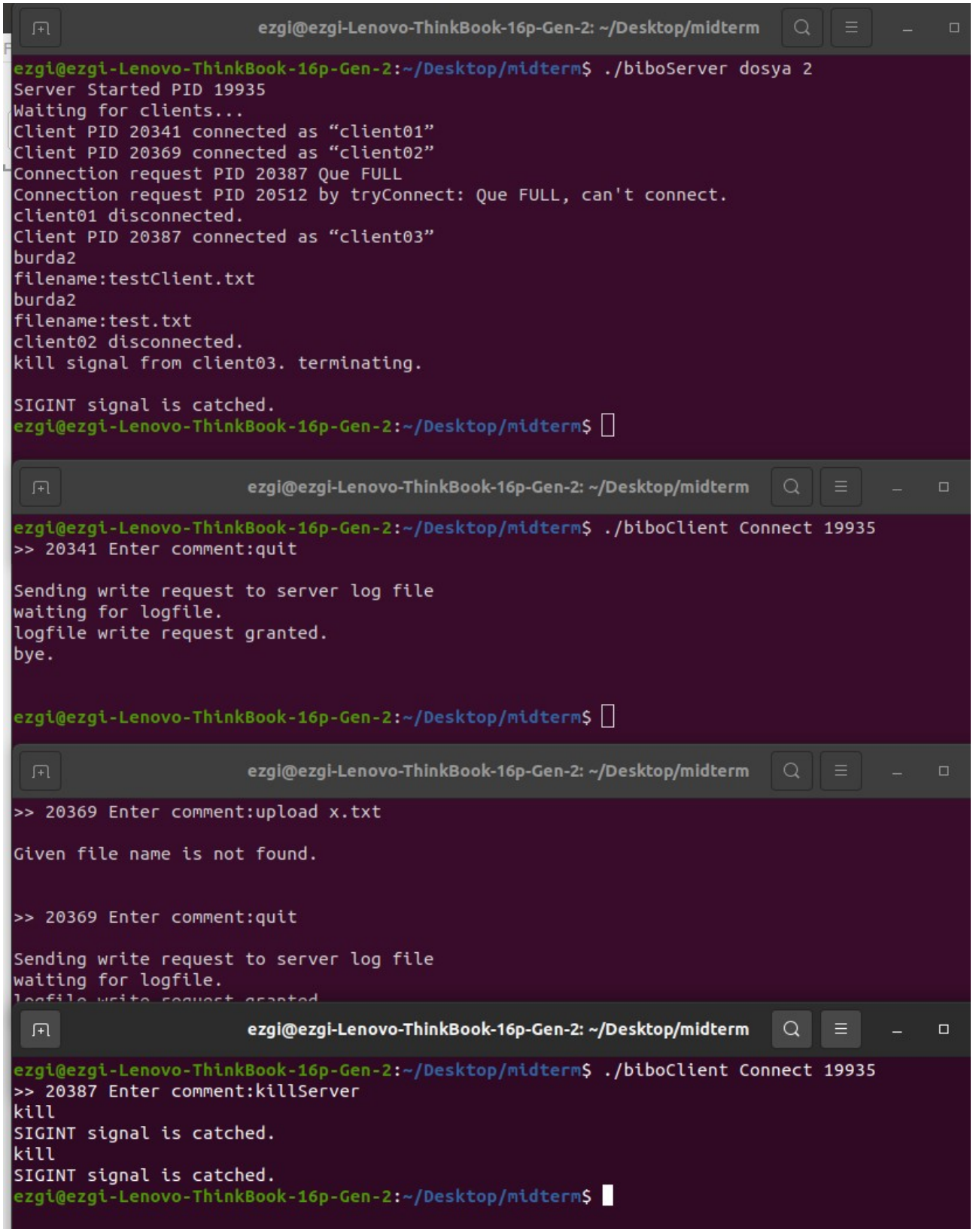


2.4. Client Sends Requests

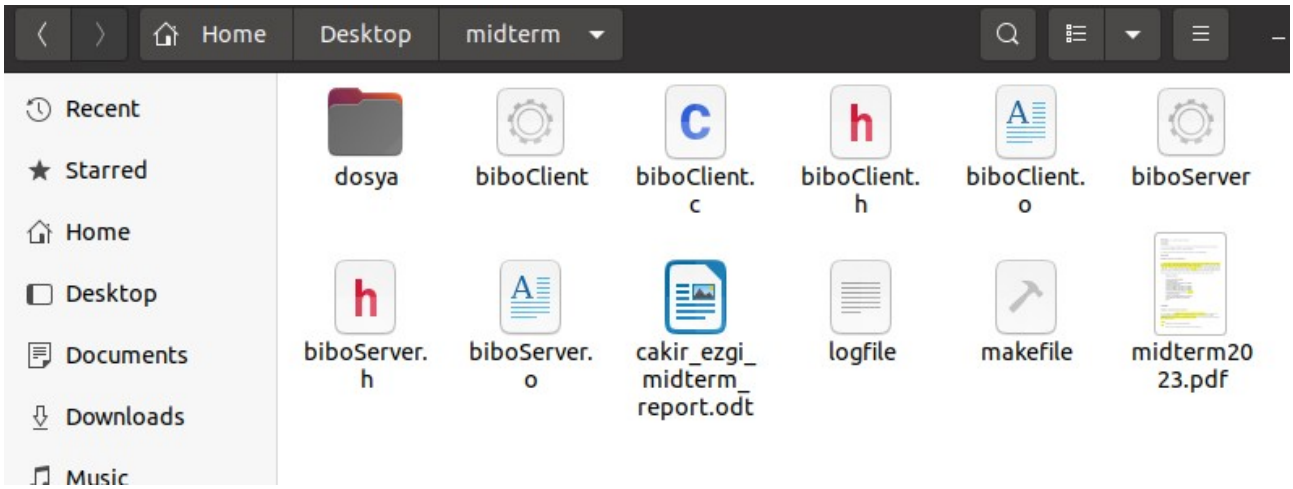
(Some cases are handled as error handlings)



2.5. Client Sends killServer



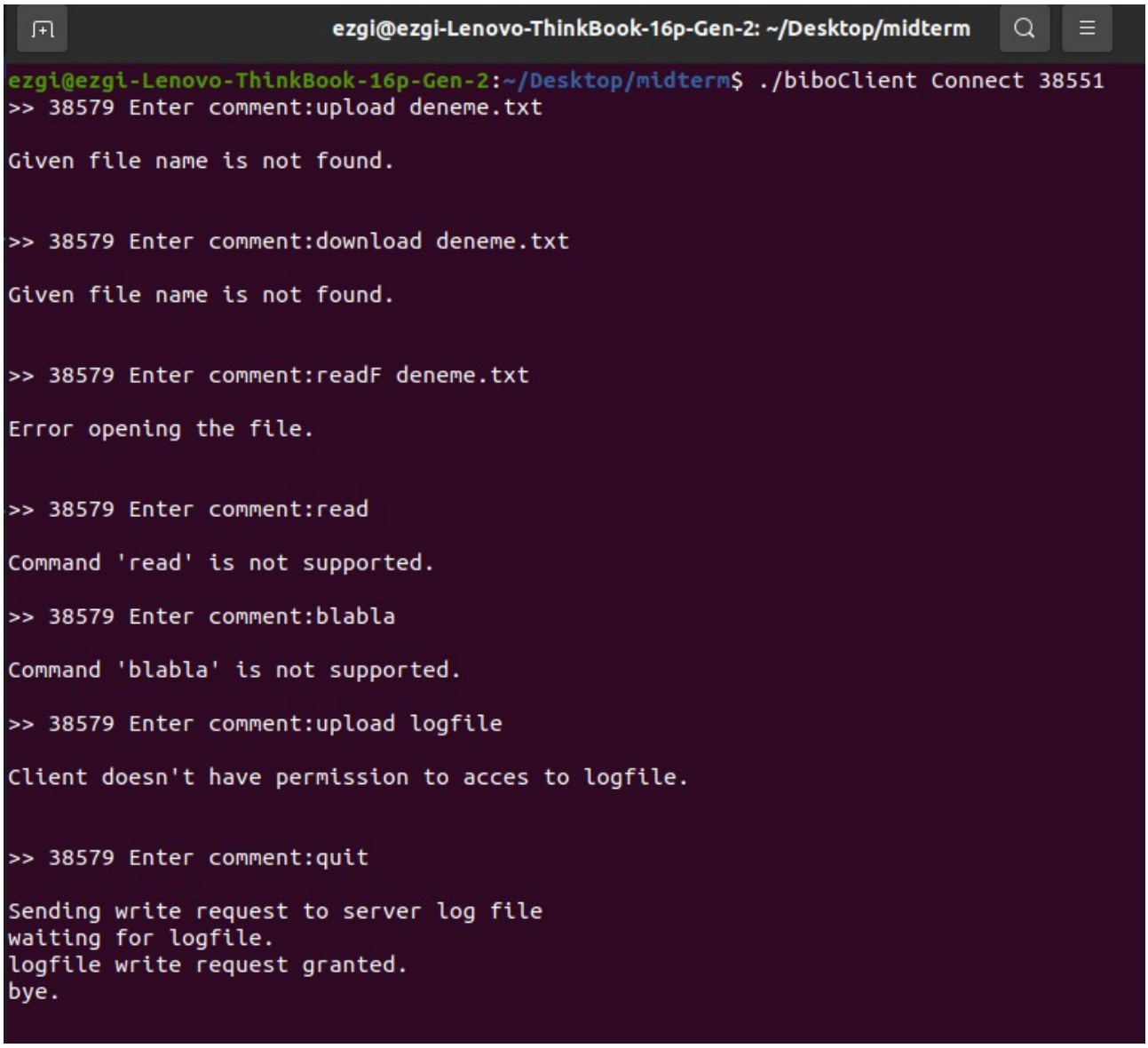
FIFOs After Terminating Server



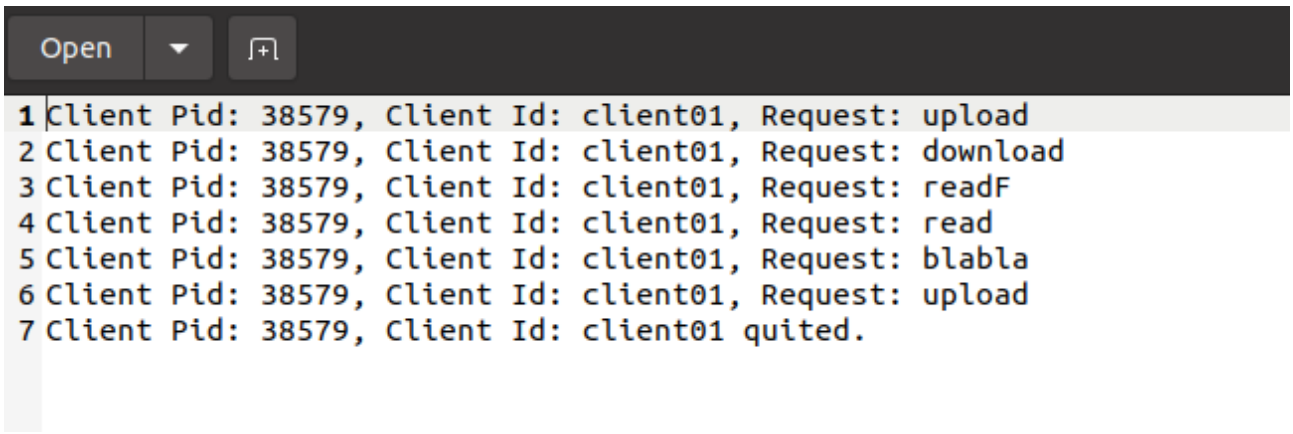
ps aux - No Zombies

```
root       36142  0.0  0.0      0     0 ?        I<   06:23   0:00 [xfs_mru_cache]
root       36143  0.0  0.0      0     0 ?        I    06:23   0:00 [kworker/5:1-events]
root       36147  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsIO]
root       36148  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36149  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36150  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36151  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36152  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36153  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36154  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36155  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36156  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36157  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36158  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36159  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36160  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36161  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36162  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36163  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsCommit]
root       36164  0.0  0.0      0     0 ?        S    06:23   0:00 [jfsSync]
root       37430  0.1  0.0      0     0 ?        I    06:24   0:00 [kworker/2:0-events]
root       38032  0.0  0.0      0     0 ?        I    06:24   0:00 [kworker/3:1-events]
root       38090  0.0  0.0      0     0 ?        I    06:24   0:00 [kworker/0:0-events]
root       38456  0.0  0.0      0     0 ?        I    06:27   0:00 [kworker/10:2-events]
root       38476  0.1  0.0      0     0 ?        I    06:27   0:00 [kworker/6:1-events]
root       38596  0.0  0.0      0     0 ?        I    06:28   0:00 [kworker/4:1-events]
root       38608  0.0  0.0      0     0 ?        I    06:29   0:00 [kworker/14:1-events]
root       38609  0.0  0.0      0     0 ?        I    06:29   0:00 [kworker/8:1]
ezgi       38637  1.1  0.1 1028480 50616 ?       Sl   06:29   0:01 /usr/bin/gnome-screenshot --g
root       38664  0.0  0.0      0     0 ?        I    06:29   0:00 [kworker/5:0]
root       38668  0.0  0.0      0     0 ?        I    06:29   0:00 [kworker/2:1]
root       38673  0.0  0.0      0     0 ?        I    06:30   0:00 [kworker/3:0-events]
root       38686  0.0  0.0      0     0 ?        I    06:30   0:00 [kworker/13:1-mm_percpu_wq]
ezgi       38690  1.0  0.1 823980 60240 ?        Sl   06:30   0:00 /usr/bin/gedit --gapplication
root       38756  0.0  0.0      0     0 ?        I    06:31   0:00 [kworker/11:0-events]
root       38757  0.0  0.0      0     0 ?        I    06:31   0:00 [kworker/15:0-events]
root       38760  0.0  0.0      0     0 ?        I    06:31   0:00 [kworker/9:1]
ezgi       38769  0.0  0.0  14232  3588 pts/0    R+   06:31   0:00 ps aux
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/midterm$
```

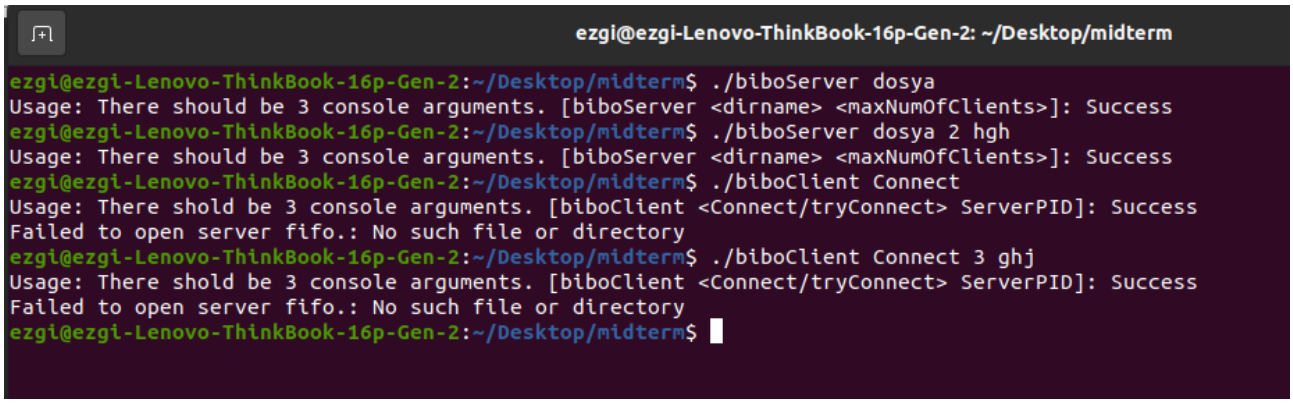
2.6. Error Handling Cases – 1 (File Access, Invalid Command)



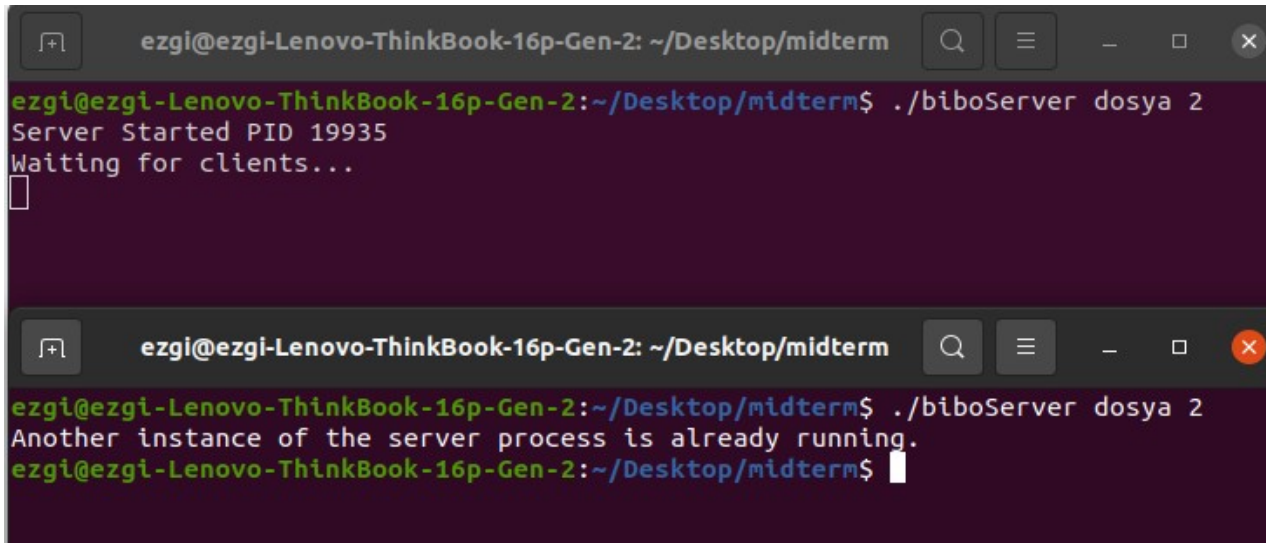
Log



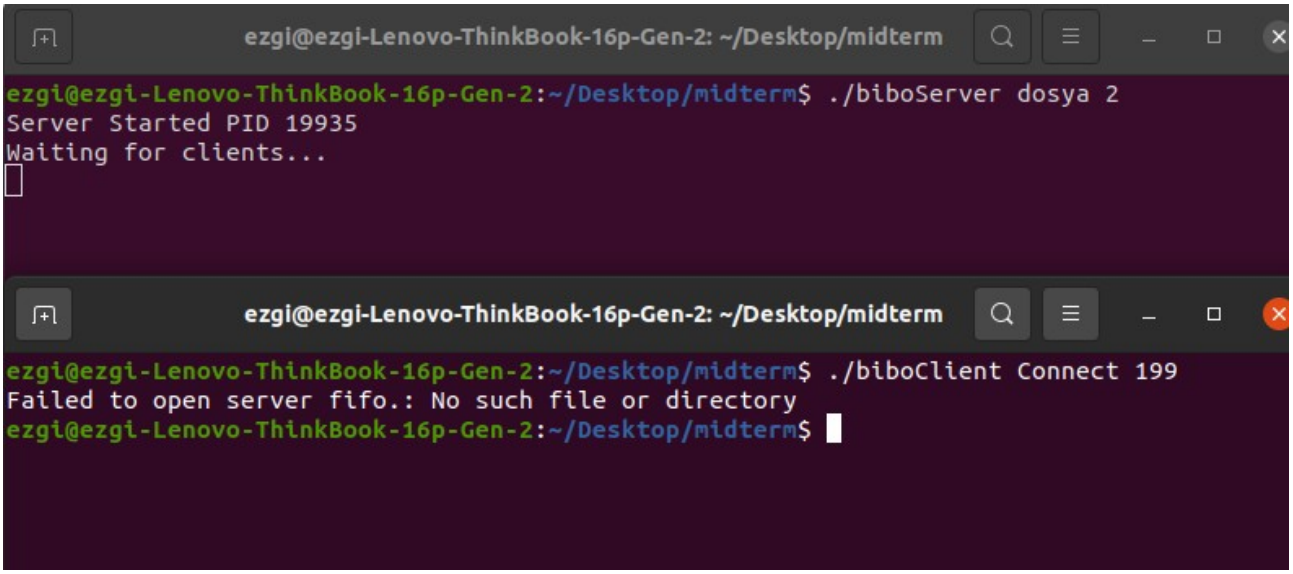
2.7. Error Handling Cases – 2 (Invalid Number of Console Arguments)



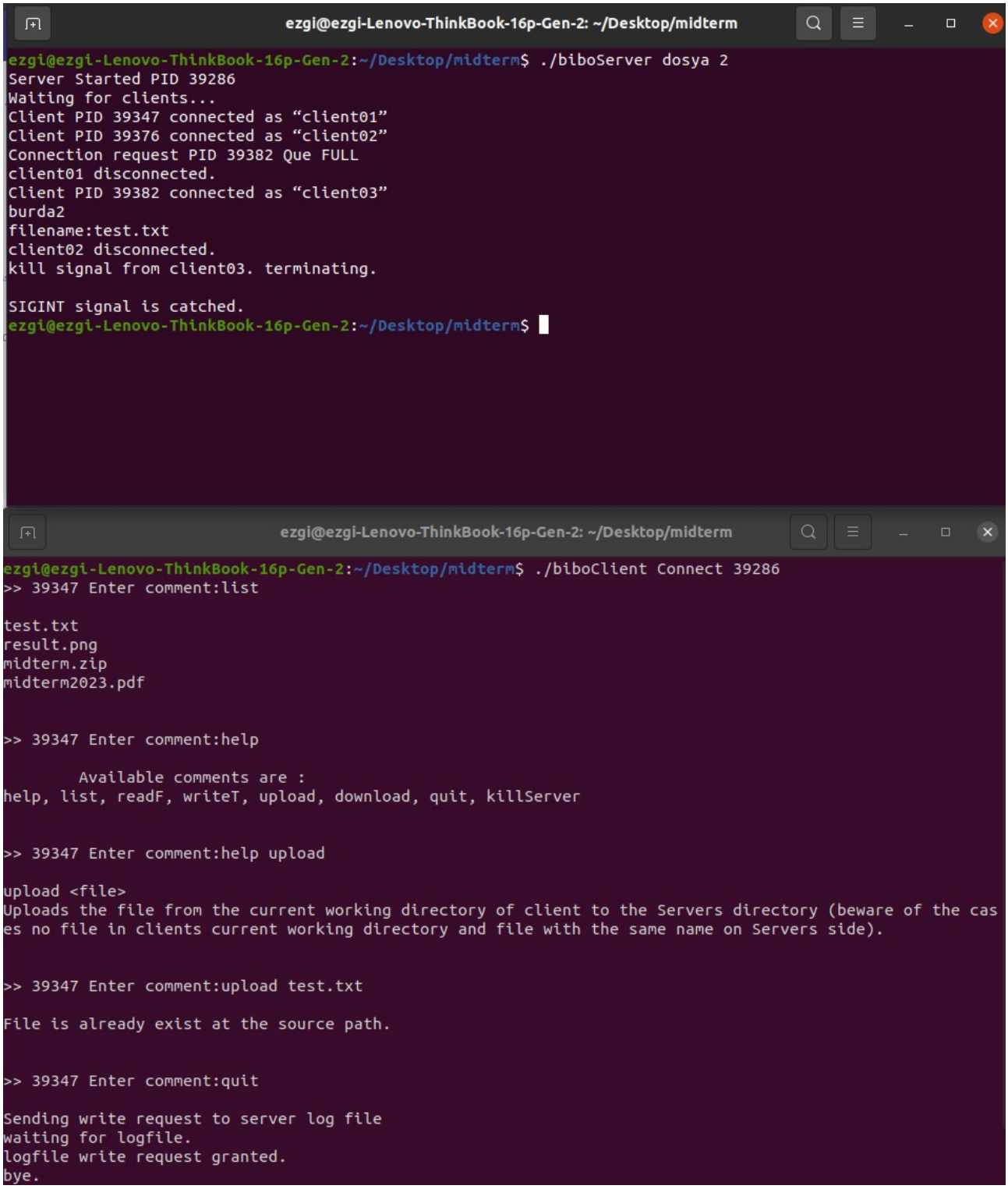
2.8. Error Handling Cases – 3 (Double Instance of Server)

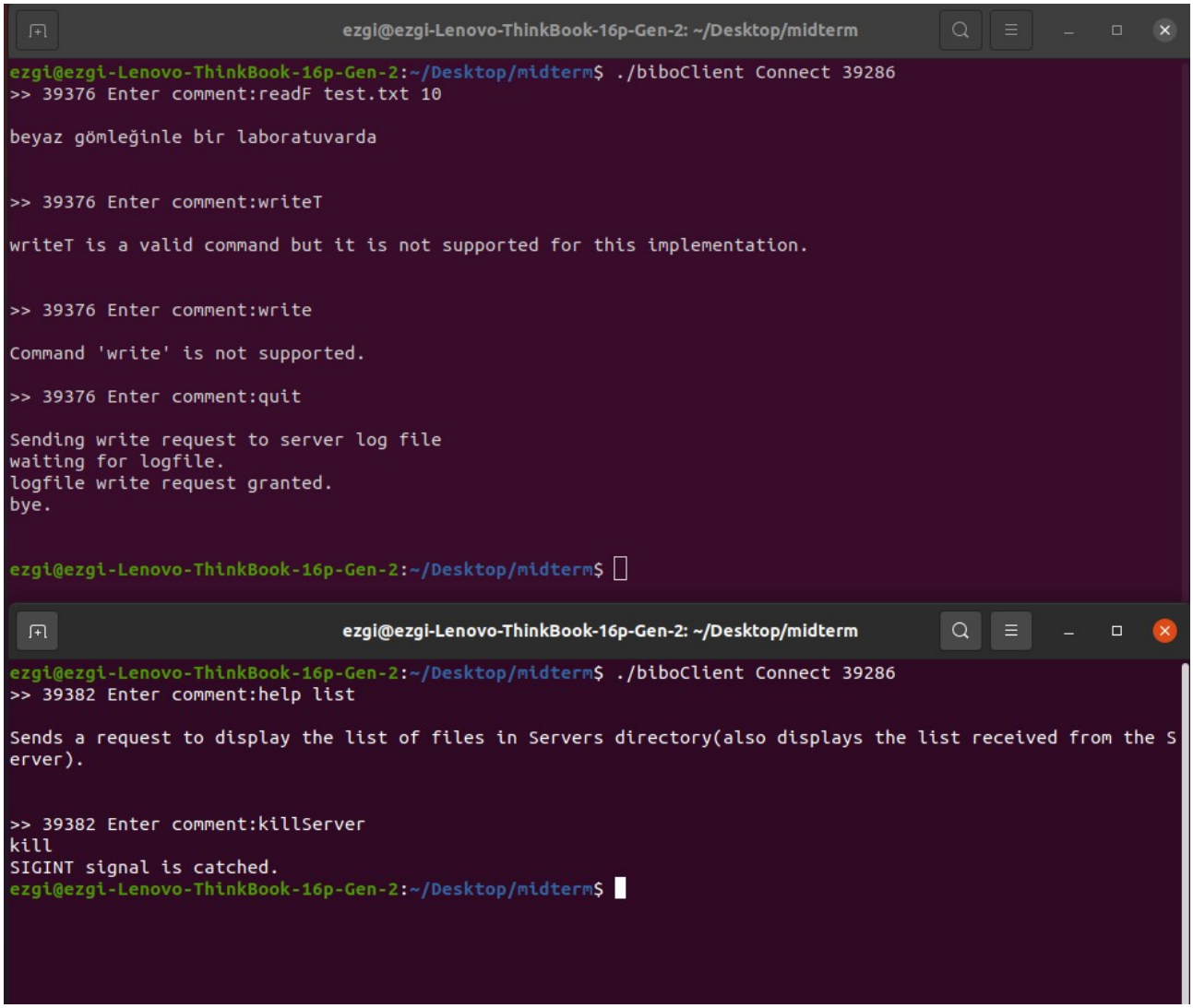


2.9. Error Handling Cases – 4 (Wrong Server Pid on Client Side)

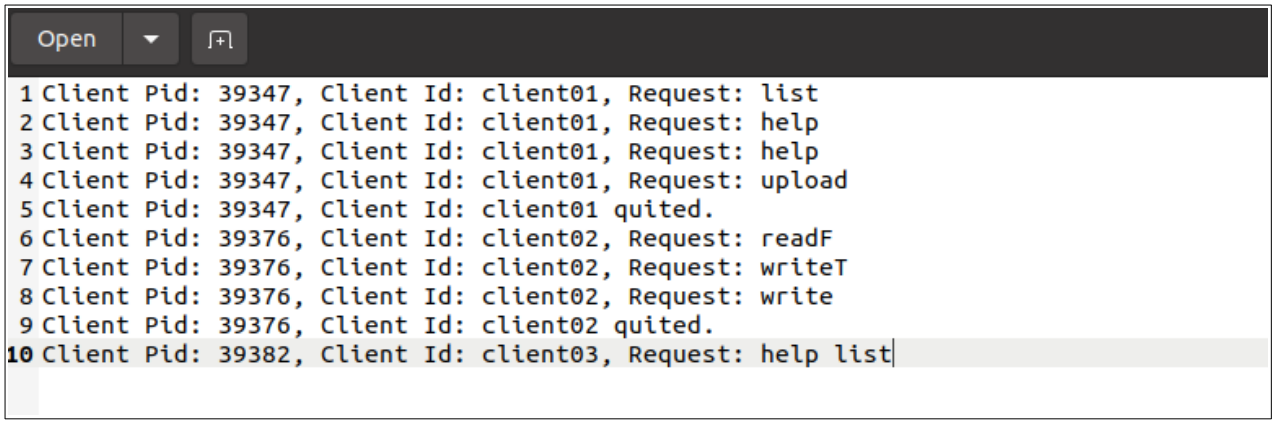


2.10. Log File Example




ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2: ~/Desktop/midterm
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/midterm$ ./biboClient Connect 39286
>> 39376 Enter comment:readF test.txt 10
beyaz gömleğinle bir laboratuvarda
>> 39376 Enter comment:writeT
writeT is a valid command but it is not supported for this implementation.
>> 39376 Enter comment:write
Command 'write' is not supported.
>> 39376 Enter comment:quit
Sending write request to server log file
waiting for logfile.
logfile write request granted.
bye.
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/midterm$
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2: ~/Desktop/midterm
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/midterm$ ./biboClient Connect 39286
>> 39382 Enter comment:help list
Sends a request to display the list of files in Servers directory(also displays the list received from the S
erver).
>> 39382 Enter comment:killServer
kill
SIGINT signal is catched.
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/midterm$

Log File


Open
1 Client Pid: 39347, Client Id: client01, Request: list
2 Client Pid: 39347, Client Id: client01, Request: help
3 Client Pid: 39347, Client Id: client01, Request: help
4 Client Pid: 39347, Client Id: client01, Request: upload
5 Client Pid: 39347, Client Id: client01 quited.
6 Client Pid: 39376, Client Id: client02, Request: readF
7 Client Pid: 39376, Client Id: client02, Request: writeT
8 Client Pid: 39376, Client Id: client02, Request: write
9 Client Pid: 39376, Client Id: client02 quited.
10 Client Pid: 39382, Client Id: client03, Request: help list

3. Makefile

You can see makefile content down below. It only complies the codes with warning flags by “make” command and cleans .o files with “make clean” command.