

SPL-1 Project Report, 2019

File Sender FTP

SE 305: Software Project Lab 1

Submitted by
Radowan Mahmud Redoy

Roll No. : BSSE 1002

BSSE Session: 2017-18

Supervised by
Dr. B M Mainul Hossain

Designation: Associate Professor
Institute of Information Technology



Institute of Information Technology

University of Dhaka

[29-05-2019]

Table of Contents

1. Introduction 1

1.1. Background Study----- 1-3

1.2. Challenges 4

2. Project Overview----- 4-7

3. User Manual----- 7-10

4. Conclusion----- 11

5. Appendix----- 11

References----- 11

Table of Figures

1. Fig 1: Socket Basic Diagram-----	1
2. Fig 2: Sample FTP session diagram-----	2
3. Fig 3: Use of fork-----	3
4. Fig 4: server part running and waiting for client request-----	6
5. Fig 5: pwd-----	8
6. Fig 6: ls-----	8
7. Fig 7: cd-----	8
8. Fig 8: Get-----	9
9. Fig 9: put-----	9
10. Fig 10 quit-----	10
11. Fig 11 Multiple Client-----	11

1. Introduction

The File Sender Ftp is mainly a file sharing application. It's main goal is to transfer files from machine to machine using TCP/IP socket and FTP protocol. It mainly uses server client model to connect machines where the session takes places. The client operates the session. During any session client is able to retrieve ,store files with in the directory of the server and if it is not the desired directory than the client can change the directory according to their needs

For the server, it usually hosts the session and acts on the commands given by the client , Here the commands are mainly FTP commands . The server can host multiple clients at a time and the during each session other sessions are not hampered and all of them are not known to the server or each other.

1.1. Background Study

Socket

One of the core part of this software is TCP socket. And a strong knowledge about TCP socket was needed. A TCP socket works on the basis of 4 points, server ip and port and client ip and port. The server creates the socket and wait for connection. So over all socket programming is a way of connection between two machines.

Now here our server firsts creates the socket that means it binds ip and port and that waits for clients request. Than the client sends request to the server and the server listens and accepts the connection. That is how our socket is implemented here.

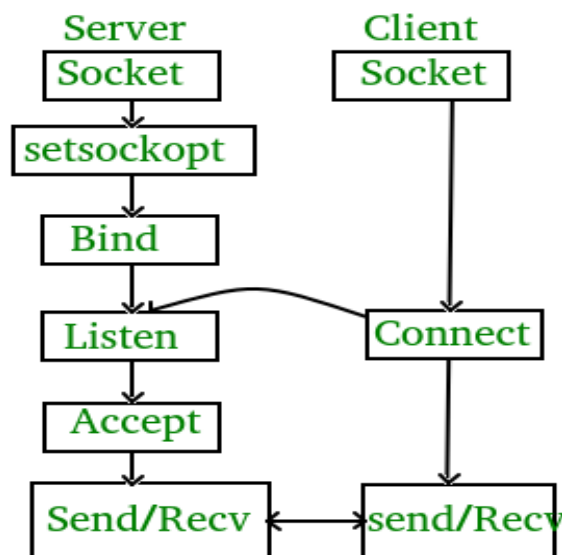


Fig 2 : Socket basic diagram (geeksforgeeks.com)

Reading Files

To send files to socket I have used a different technique . The files have been read in binary mood and than all have been included in an array and than the array is sent. Besides for sending text files sendfileoversocket has been implemented.

FTP

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network.

FTP is built on a client-server model architecture using separate control and data connections between the client and the server. FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

The first FTP client applications were command-line programs developed before operating systems had graphical user interfaces, and are still shipped with most Windows, Unix, and Linux operating systems

When a FTP session is started between a client and a server, the client initiates a control TCP connection with the server side. The client sends control information over this. When the server receives this, it initiates a data connection to the client side. Only one file can be sent over one data connection. But the control connection remains active throughout the user session. As we know HTTP is stateless i.e. it does not have to keep track of any user state. But FTP needs to maintain a state about its user throughout the session.

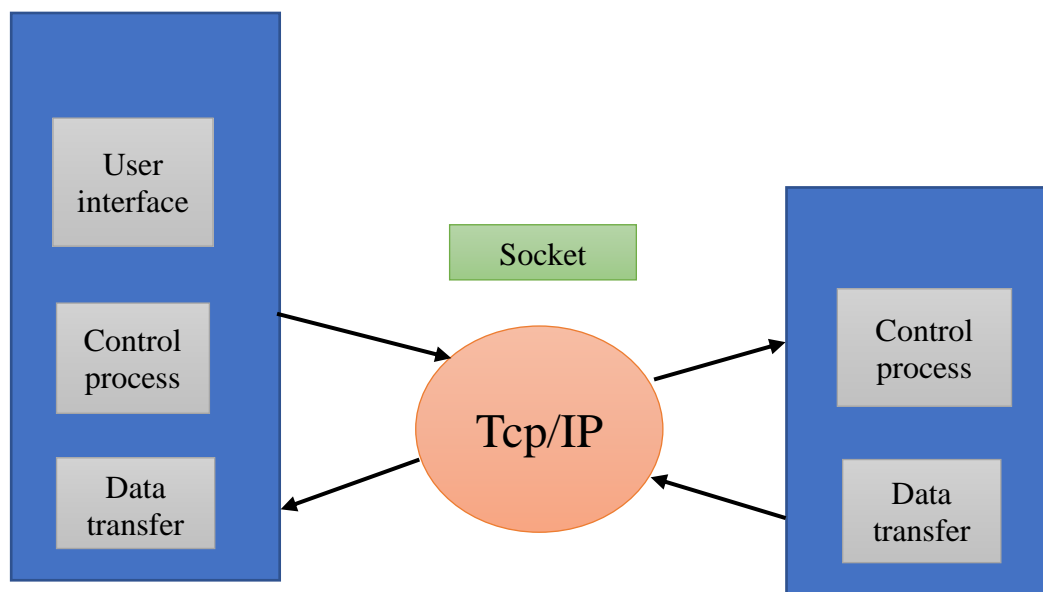


Fig 2: Sample FTP session diagram

Fork

The server can add multiple clients at a time . But as we have used c language there was no scope for thread there fore we have implemented fork. Fork is a function in Unix that is used to generate a duplicate of particular process by creating two simultaneous executing processes of a program

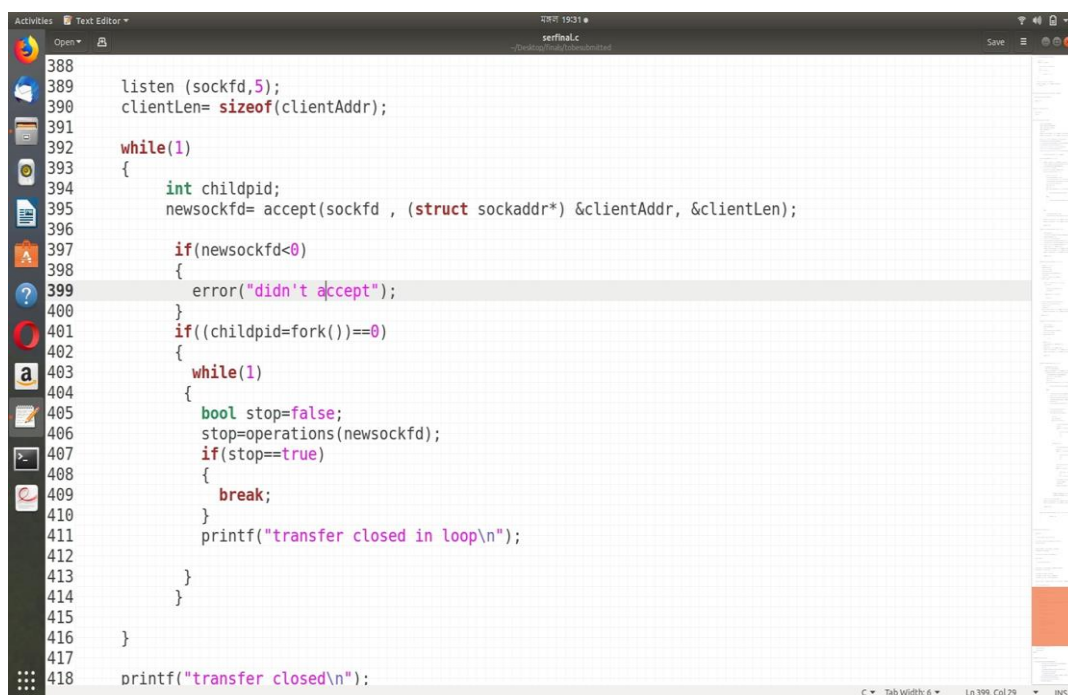
Fork system call use for creates a new process, which is called child process, which runs concurrently with process (which process called system call fork) and this process is called parent process. After a new child process created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

Fork takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive value: Returned to parent or caller. The value contains process ID of newly created child process.



```

388
389 listen(sockfd,5);
390 clientLen= sizeof(clientAddr);
391
392 while(1)
393 {
394     int childpid;
395     newsockfd= accept(sockfd , (struct sockaddr*) &clientAddr, &clientLen);
396
397     if(newsockfd<0)
398     {
399         error("didn't accept");
400     }
401     if((childpid=fork())==0)
402     {
403         while(1)
404         {
405             bool stop=false;
406             stop=operations(newsockfd);
407             if(stop==true)
408             {
409                 break;
410             }
411             printf("transfer closed in loop\n");
412         }
413     }
414 }
415
416 }
417
418 printf("transfer closed\n");

```

Fig 3 : use of fork

1.2. Challenges

My main challenge was to send various files . Such as text , jpg, video etc. And for these I have tried quit a few process and most of them appeared with different problems. . Finally I have solved these problems by sending them as binary form.

Besides adding ,multiple clients was a challenge her. I have used fork to accomplish these challenge.. Besides maintaining standard format changing directory or getting the current file list , I have solved these challenges. So to point out the key challenges were

- ⇒ Understanding concept of Socket
- ⇒ Read all files in binary and send them
- ⇒ Getting list of files from server
- ⇒ Use standard protocols
- ⇒ Make the server handle multiple clients

2. Project Overview

The software I have designed has two parts

- ⇒ Server part
- ⇒ Client part

The main operations are run by the server part where the client send different FTP commands . The client is mainly the driver of the session . The client actually give the commands. Multiple client are able to run multiple session and all of them are different from Each other. The software is mainly linux based and runs on linux operating system.

Server part

My server part works as the main server here which receives the FTP commands. Such as The client wants to change the directory than he initially gives the command to the client interface and from there the client sends the command to the server. The server gets the command matches with its standard and than operate the operation

Our server mainly acts on commands such as

- ⇒ Get Filename
- ⇒ Put Filename
- ⇒ cd
- ⇒ ls
- ⇒ pwd
- ⇒ quit

our server accepts these commands.

Get Filename

Here Get Filename command is used to retrieve any kind of file which is in the server. The server executes the command checks if the file is in the current directory of the server. And if it is there than first sends a message ok for the client part to confirm that server has got the file and it will start the transfer.

While going to transfer the server uses two ways. If the file is txt than it is sent at a whole. If not than it is sent as small chunks. First if it is not text than the file size is sent. Then the total file size is divided by 100. The number of results is also the number of segments which are to be sent and again the file size is modded by 100 and this time the remainder is sent as another small chunk. Here first the chunks are copied into an array and the array is sent to the server. Thus the whole file is sent to the server.

Put Filename

The put Filename command is used to store file in the current directory of the server. Here first the server gets the command put filename and extracts the filename and creates the corresponding file of that name. If again it is not txt than the file is sent as chunks. Here the server also receives the file size from the client and uses malloc to create a certain buffer of that size and then using fwrite the buffer is written down in the corresponding file.

Ls

The server also supports the command ls which stands for list of files. Often the client needs to know which files are in the current directory of the server. First the server receives the command and then it runs system function for ls and stores the result in a txt file and then that txt file is sent to the client. And the txt file is by default removed from the server.

Pwd

Now pwd stands for present working directory. If the current working directory of the server is not known to the server than the client can easily send the command to the server and the server again runs the system function and gives the present directory along with confirmation.

Cd

The cd command is for change of current directory. After the ls and pwd commands or may be any part of the session and if the client thinks that the current directory does not hold the files than the cd command is given. First the server receives the cd command and sends the confirmation that server is ready and then the client gives his desired path and the server then changes the directory and sends confirmation.

quit

If the client sends the command quit that means that the client is wanting to end the session and the process stops.

Our server accepts multiple client at a time. But all the connections are not disturbed by others. I have used fork here to run different process peralally. So even if a client gives the command quit only his/her certain session stops . The server is still alive .

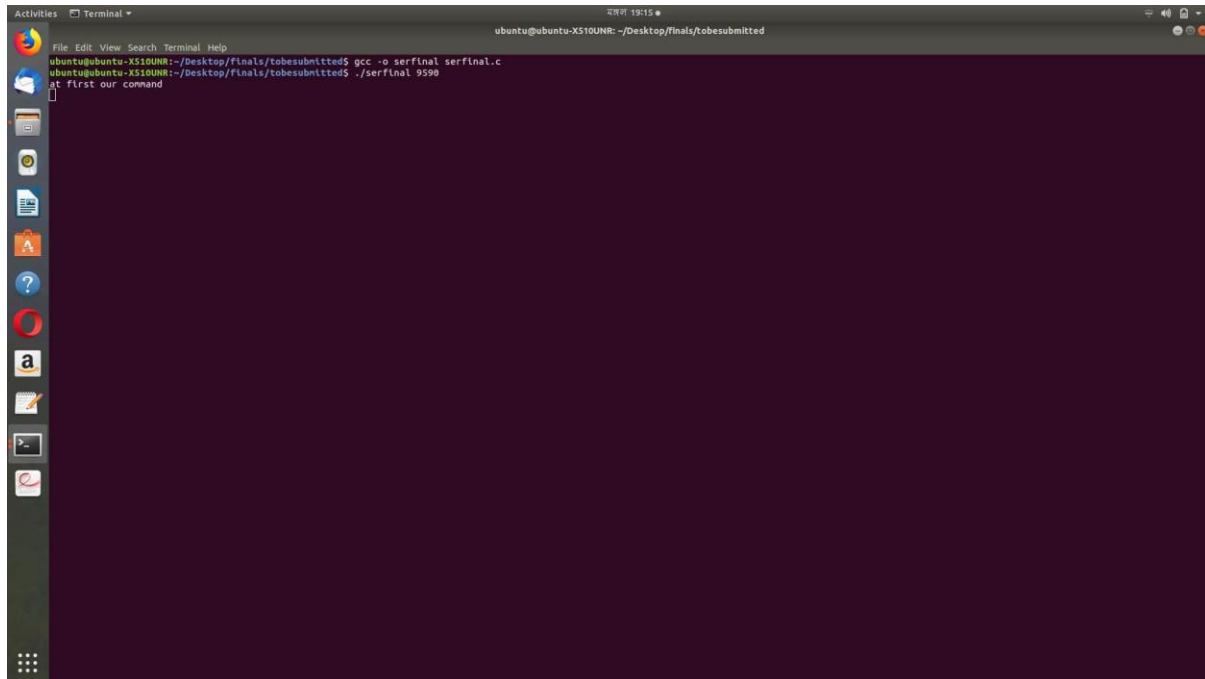


Fig 4: server part running and waiting for client request

Client part

The client part is the driver of our session. The client gives the command to the server. Here the user gives the command as he needs. Besides the client part has a few commands for itself as well.

- ⇒ !ls
- ⇒ !cd
- ⇒ !pwd

!ls

If the client wants know the list of files in his current directory than this !ls command can be used by the client.

!cd

If the client wants change his own directory than !cd command is used. Than if all is right than the client provides his/her desired directory.

!pwd

The !pwd stands for present directory of the client

Again the client must end the process by giving the command quit. The file receiving pattern for the client is first the size of the file is received and using malloc a file of same size is created. If the file is txt then we have used fopen to create the file. And then using recv() the file is received.

And if the file is not such then we have used the same process as of sending just only here we are receiving the chunk arrays and then after receiving all the array they are concatenated and using fwrite they are written on the file.

And for put or for sending files first the name is sent to the server and then the file size and the size of the file. And then depending on the type of the file the file is sent.

Over all, these are my works for the server and the client part.

Using these process two machine can connect between them and operate a session.

3. User Manual

First we need to set up the server. Our application is mainly a linux based application. Runs on linux operating system. First we need to set up the server. The server must run a particular port. Generally for FTP the port is 21. But for our custom built software random ports are to be used. So to initialize the server first we need to go to the current directory of the server and run commands

```
~$ gcc -o serfinal serfinal.c
~$ ./serfinal 9590
```

These commands will activate our server.

NOW ANY CLIENT WILL BE ABLE TO CONNECT THE SERVER ON THE CERTAIN PORT.

Now to run our client part

```
~$ gcc -o clifinal clifinal.c
~$ ./clifinal {IP of server} {port of server}
```

And thus our session begins.

Now moving on to the commands

```

ubuntu@ubuntu-XS10UNR: ~/Desktop/finals/tobesubmitted/client
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ gcc -o clifinal clifinal.c
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ ./clifinal 127.0.0.1 9590
Give command
pwd
3
our request msg is pwd
first reply msg
41
/home/ubuntu/Desktop/finals/tobesubmitted
loop is breaking
Give command

```

Fig 5: pwd

The client just have to write the command pwd on the prompt Give command

```

ubuntu@ubuntu-XS10UNR: ~/Desktop/finals/tobesubmitted/client
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ gcc -o clifinal clifinal.c
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ ./clifinal 127.0.0.1 9590
Give command
ls
2
our request msg is ls
first reply msg
for ls is
for ls 2
The remote directory listing is as follows:
client
serfinal
serfinal.c
t.txt
loop is breaking
Give command

```

Fig 6: ls

The client just gives the command ls

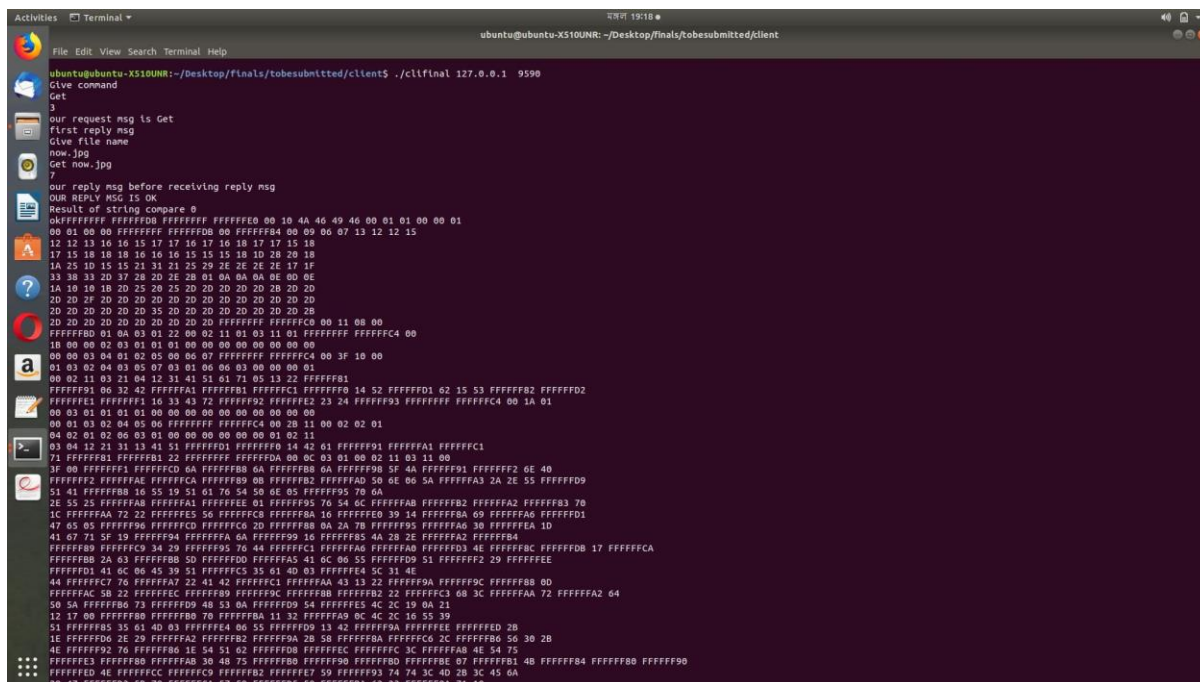
```

ubuntu@ubuntu-XS10UNR: ~/Desktop/finals/tobesubmitted/client
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ gcc -o clifinal clifinal.c
ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ ./clifinal 127.0.0.1 9590
Give command
cd
2
our request msg is cd
first reply msg
enter your choiceable path
/home/ubuntu/Desktop
20
changed successfully
loop is breaking
Give command

```

Fig 7:cd

The client first needs to give the command `cd`. And then the full path of the new directory has to be given.



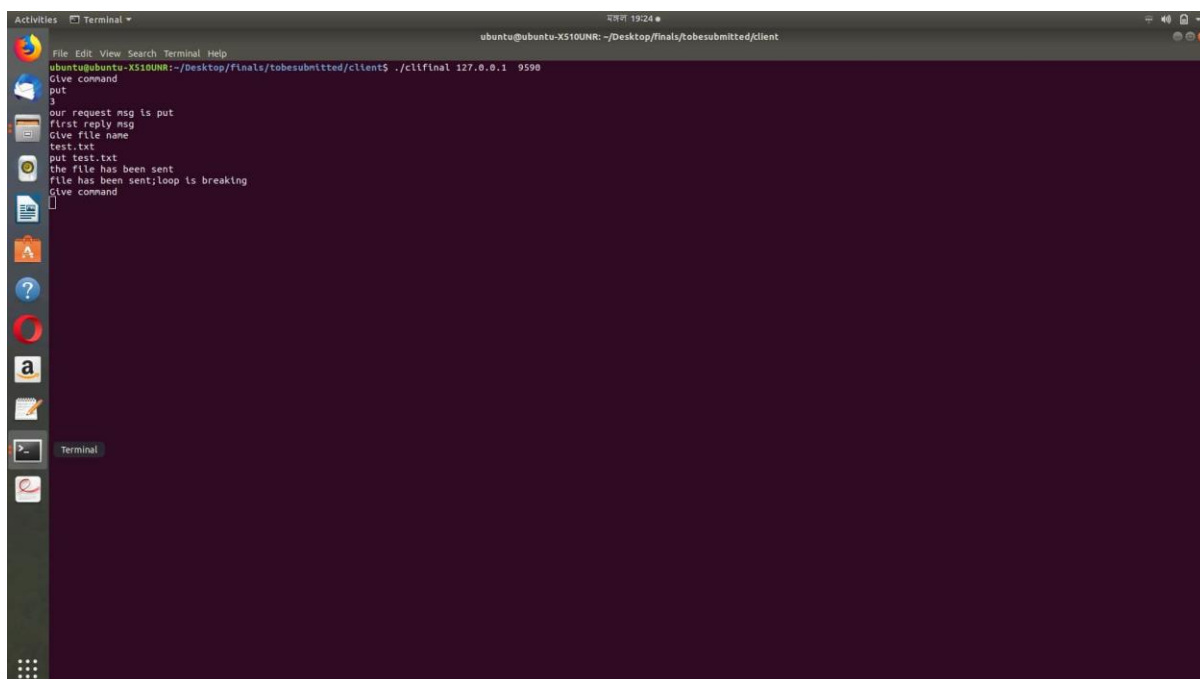
```

ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ ./clifinal 127.0.0.1 9590
Give command
get
3
our request msg is Get
first reply msg
Give file name
now.jpg
Get now.jpg
7
our reply msg before receiving reply msg
OUR REPLY MSG IS OK
Result of string compare 0
00 01 00 00 FFFFFFFF FFFFFFFB 00 FFFFFFFB 00 09 06 07 13 12 12 15
12 13 16 15 17 17 16 17 16 17 17 17 15 18
17 15 18 18 16 16 16 15 15 15 18 10 28 20 18
1A 25 10 15 15 21 31 21 25 29 2E 2E 2E 17 1F
33 38 33 20 37 28 20 2E 01 0A 0A 0A 0E 00 0E
1A 10 18 20 25 20 25 20 20 20 20 2B 20 20
20 20 2F 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 35 20 20 20 20 20 20 20 2B
20 20 20 20 20 20 20 20 20 FFFFFFFF FFFFFFFC 00 11 08 00
FFFFFFFF 01 0A 03 22 00 02 11 01 03 11 01 FFFFFFFF FFFFFFFC 00
18 00 00 02 03 01 01 01 00 00 00 00 00 00 00
00 03 04 01 02 05 00 01 FFFFFFFF FFFFFFFC 00 3F 10 00
01 03 02 04 03 05 07 03 01 06 06 03 00 00 01
00 02 11 03 21 04 12 31 41 51 01 71 05 13 22 FFFFFFFB
FFFFFFFF 06 32 42 FFFFFFFF FFFFFFFB FFFFFFFC FFFFFFFD 14 52 FFFFFFFD 02 15 53 FFFFFFFB FFFFFFFD
FFFFFFFF 16 33 43 72 FFFFFFFC FFFFFFFE 23 24 FFFFFFFD FFFFFFFF FFFFFFFC 00 1A 01
00 03 01 01 01 01 00 00 00 00 00 00 00 00
00 01 03 02 04 05 06 FFFFFFFF FFFFFFFC 00 2B 11 00 02 02 01
04 02 01 02 06 03 01 00 00 00 00 00 01 02 11
03 04 12 21 31 13 41 51 FFFFFFFD FFFFFFFD 14 42 01 FFFFFFFD FFFFFFFC
71 FFFFFFFB FFFFFFFB 22 FFFFFFFF FFFFFFFD 00 0C 03 01 00 02 11 03 11 00
3F 00 FFFFFFFF FFFFFFFC 0A FFFFFFFB 0A FFFFFFFB 0A FFFFFFFB 5F 4A FFFFFFFD FFFFFFFE 6E 40
FFFFFFFF FFFFFFFF FFFFFFFC FFFFFFFB 0B FFFFFFFB FFFFFFFD 50 06 0A FFFFFFFA 2A 2E 55 FFFFFFFD
51 41 FFFFFFFB 16 55 19 51 01 70 54 50 06 05 FFFFFFFF 70 6A
2E 55 25 FFFFFFFB FFFFFFFF FFFFFFFE 01 FFFFFFFF 76 54 0C FFFFFFFB FFFFFFFB FFFFFFFA FFFFFFFB 70
1C FFFFFFFA 72 22 FFFFFFFE 50 FFFFFFFC FFFFFFFB 16 FFFFFFFD 39 14 FFFFFFFB 09 FFFFFFFA FFFFFFFD
47 05 05 FFFFFFFF FFFFFFFC FFFFFFFC 2D FFFFFFFB 0A 2A 7B FFFFFFFF FFFFFFFA 30 FFFFFFFE 1D
41 07 71 5F 19 FFFFFFFF FFFFFFFA 0A FFFFFFFF 16 FFFFFFFB 4A 2B 2E FFFFFFFA FFFFFFFB
FFFFFFFF FFFFFFFC 34 29 FFFFFFFF 70 4A FFFFFFFC FFFFFFFA FFFFFFFD 4E FFFFFFFB FFFFFFFD 17 FFFFFFFA
FFFFFFFF 2A 03 FFFFFFFB 5D FFFFFFFD FFFFFFFA 41 0C 06 55 FFFFFFFD 51 FFFFFFFE 29 FFFFFFFE
FFFFFFFF 41 0C 06 45 39 51 FFFFFFFC 35 01 40 03 FFFFFFFE 5C 31 4E
44 FFFFFFFC 76 FFFFFFFA 22 41 42 FFFFFFFC FFFFFFFA 43 13 22 FFFFFFFA FFFFFFFC FFFFFFFB 0D
FFFFFFFF 5D 22 FFFFFFFE FFFFFFFB FFFFFFFC FFFFFFFB FFFFFFFB 22 FFFFFFFC 08 3C FFFFFFFA 72 FFFFFFFA 64
50 5A FFFFFFFB 73 FFFFFFFD 40 53 0A FFFFFFFD 54 FFFFFFFE 4C 2C 19 0A 21
12 17 00 FFFFFFFB FFFFFFFB 70 FFFFFFFB 11 32 FFFFFFFA 0C 4C 2C 10 55 39
51 FFFFFFFB 35 61 40 03 FFFFFFFE 00 55 FFFFFFFD 13 42 FFFFFFFA FFFFFFFE FFFFFFFD 2B
1E FFFFFFFD 2E 29 FFFFFFFA FFFFFFFB FFFFFFFA 2B 58 FFFFFFFB FFFFFFFC 2C FFFFFFFB 56 30 2B
4E FFFFFFFD 76 FFFFFFFB 1E 54 51 02 FFFFFFFB FFFFFFFE FFFFFFFC 3C FFFFFFFB 4E 54 75
FFFFFFFF FFFFFFFB FFFFFFFB 30 40 75 FFFFFFFB FFFFFFFD FFFFFFFB FFFFFFFB 07 FFFFFFFB 40 FFFFFFFB FFFFFFFB
FFFFFFFF 4E FFFFFFFC FFFFFFFC FFFFFFFB FFFFFFFE 59 FFFFFFFD 74 74 3C 4D 2B 3C 45 6A
29 47 FFFFFFFD 5B 7B FFFFFFFC 57 50 FFFFFFFB 5B FFFFFFFD 61 23 FFFFFFFB 71 10

```

Fig 8 : Get

First the client gives the command `get` and then the file name. Than both the commands are attached and sent to the server.



```

ubuntu@ubuntu-XS10UNR:~/Desktop/finals/tobesubmitted/client$ ./clifinal 127.0.0.1 9590
Give command
put
3
our request msg is put
first reply msg
Give file name
test.txt
put test.txt
the file has been sent
file has been sent; loop is breaking
Give command

```

Fig 9: put

Same way. First `put` than filename and then they are attached and sent to server

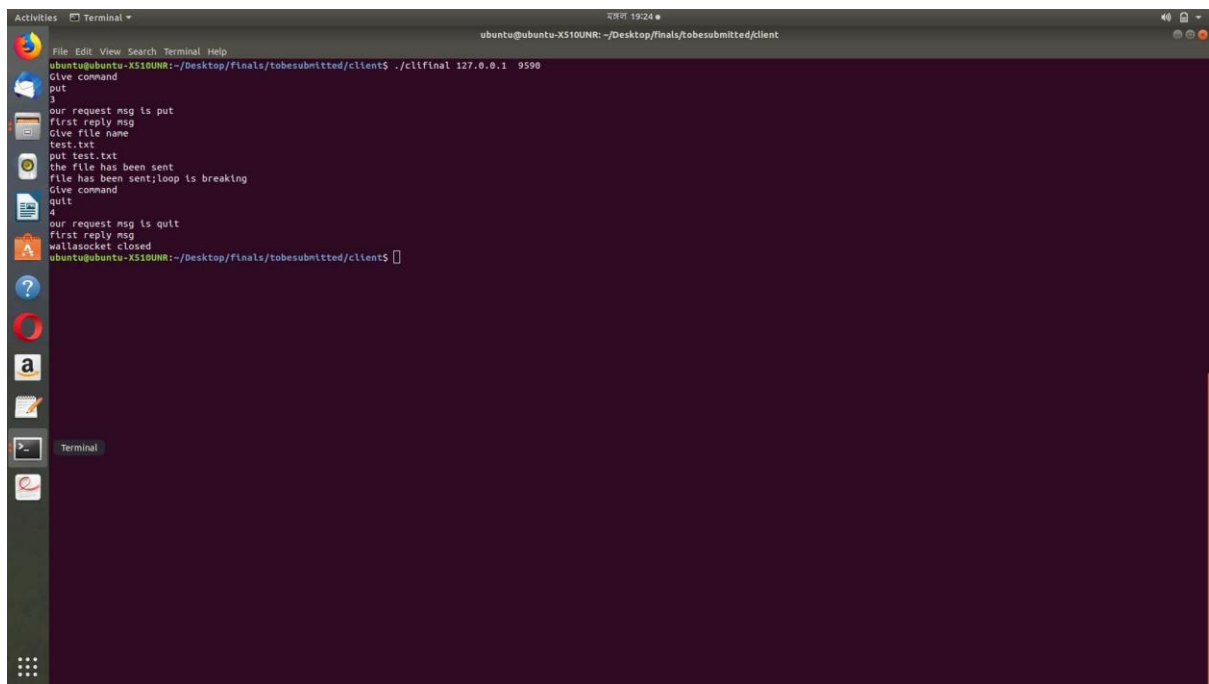


Fig 10 : quit

When the quit command is given the session ends

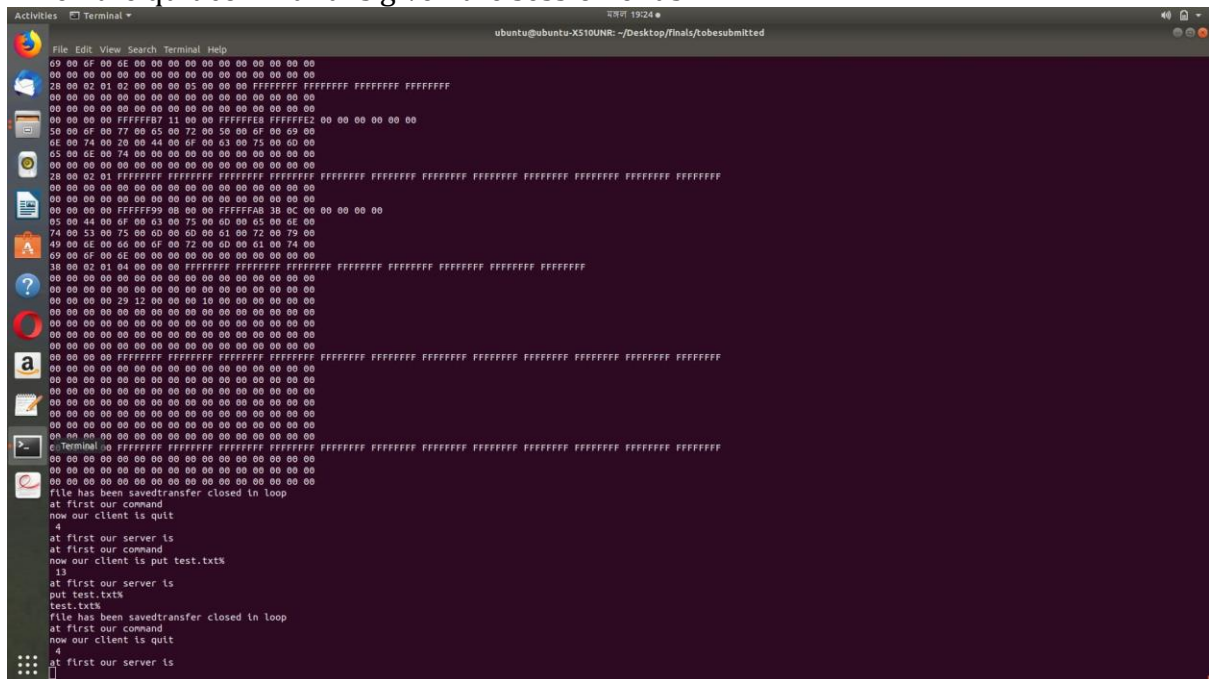


Fig 11: Multiple client

Here though one client has ended his/ her session still due to connections with other clients the server is still on.

4.Conclusion

The project is mainly based upon sockets Ftp which are really new topics to me . I had to do some background study in order to do the project . Besides I have handled a large amount of code and besides I have tried different formats to transfer so over all I have made a mark which was suggested to me for this software

5.Appendix

Over all I tried to make a file sharing application which will run on the basis of ftp command. I experienced hardships while doing this project but still there are few bugs or areas of improvement for me. But overall this is my software.

References

1

.<https://www.geeksforgeeks.org/computer-network-file-transfer-protocol-ftp/>

2.

http://www.ece.eng.wayne.edu/~gchen/ece5650/myftp.pdf?fbclid=IwAR2alqO766euloEBumXkeSS0r6xhKJz_56JSRfMrw6s30FzomWN0rVtQZF0

3.

<https://stackoverflow.com/questions/13762673/how-to-program-an-ftp-client-in-c?fbclid=IwAR2iB5aAwlrhSUWnc2qbIKiSv8abwCiOR1NSKUgvU25IUPDAQiKH48sv17Q\>

4.

https://www.wired.com/2010/02/ftp_for_beginners/?fbclid=IwAR09U1YWeIzWXFu2pUseX4YfyX5W4KDem0H77MRinn9n8MAtdmi7huqr9U0