DESIGN AND IMPLEMENTATION OF SECONDARY STORAGE SYSTEM

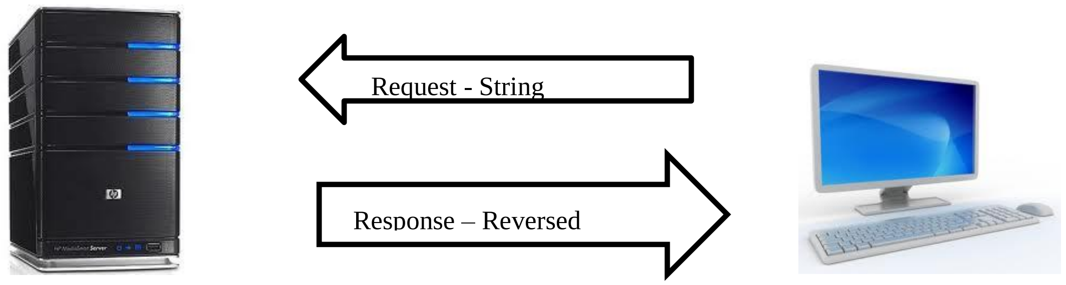
OBJECTIVE:

The project objective was to learn about the Network API and socket programming and to use them to implement a basic disk like secondary storage. Also, design and implement a basic file system that acts as a client to access the disk server and use the services provided by it.

ABSTRACT:

This project is the final course project in **CSE 521 Operating System Concepts**. This project was developed in five parts from a basic client server socket programming to design and implementation of file system that uses a secondary storage disk server with directory structure. The project was written using C programming on UNIX machine. **Part 1** is a **basic client server** that implements a string reverse program. **Part 2** is a **directory listing server** where the client sends ‘ls’ command to the server while the server executes and uses a dup2 to send the result back to client through sockets. **Part 3** is a **Basic disk server** that creates a file and performs various functions like read, write and send track and sector info based on the client request. **Part 4** is the **File System Server**, a higher implementation on the same disk server in part 3 where the disk server is contacted by the fileserver that acts as client to disk server and a file client that sends simple request like create read and write. The file server performs various operations on the requests and again sends appropriate request to the disk server based on the inputs it can take. The output is sent back to the file client. **Part 5** is the implementation of **Directory Structure** on the File System Server to create directories and sub directories on the file system.

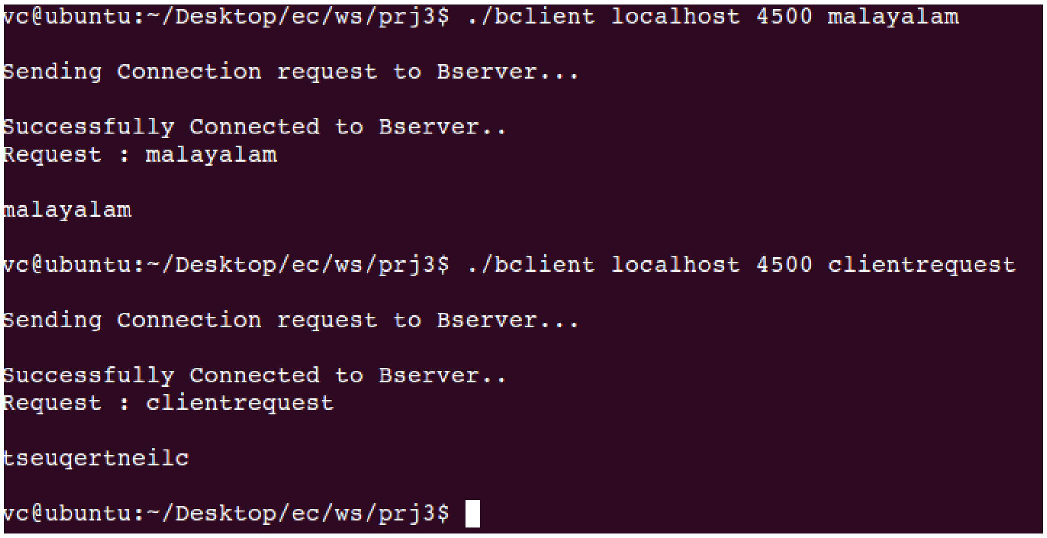
PROGRAM 1: Basic Client Server model to get the String Reverse



This is a basic client server program with a server that creates a socket, binds the socket and listens for incoming connections. When an incoming connection comes, it accepts the connection and keeps the connection open, till the client quits or server crashes. The server receives requests from the client by reading from the socket, and it sends response by writing to the socket.

The client, also creates a socket, and sends a request to the server by giving the IP address and port as command line arguments. This specific implementation of client gets the string to be reversed by the server as command line argument from the user and the client exits after printing the reversed string to the screen.

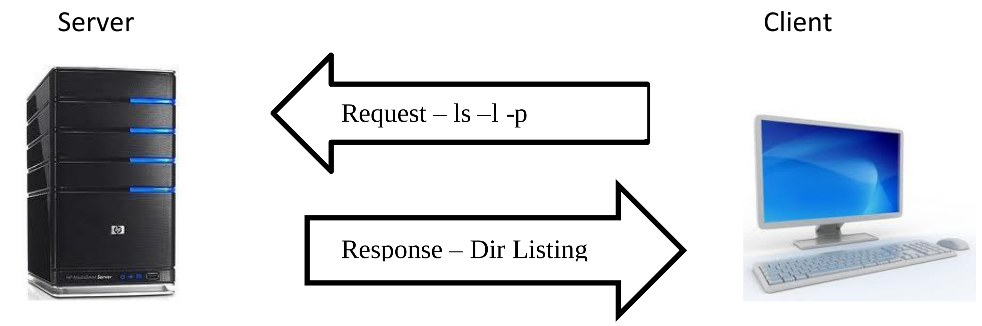
The server, when it receives the request, performs the string reverse, writes the output and closes the socket once the client exits and waits listening for other incoming connections. We quit the server by using ^C. The main advantage of server client implementation is many clients can access one server to reverse a string instead of creating their own program. Server Client programming comes handier for bigger program.



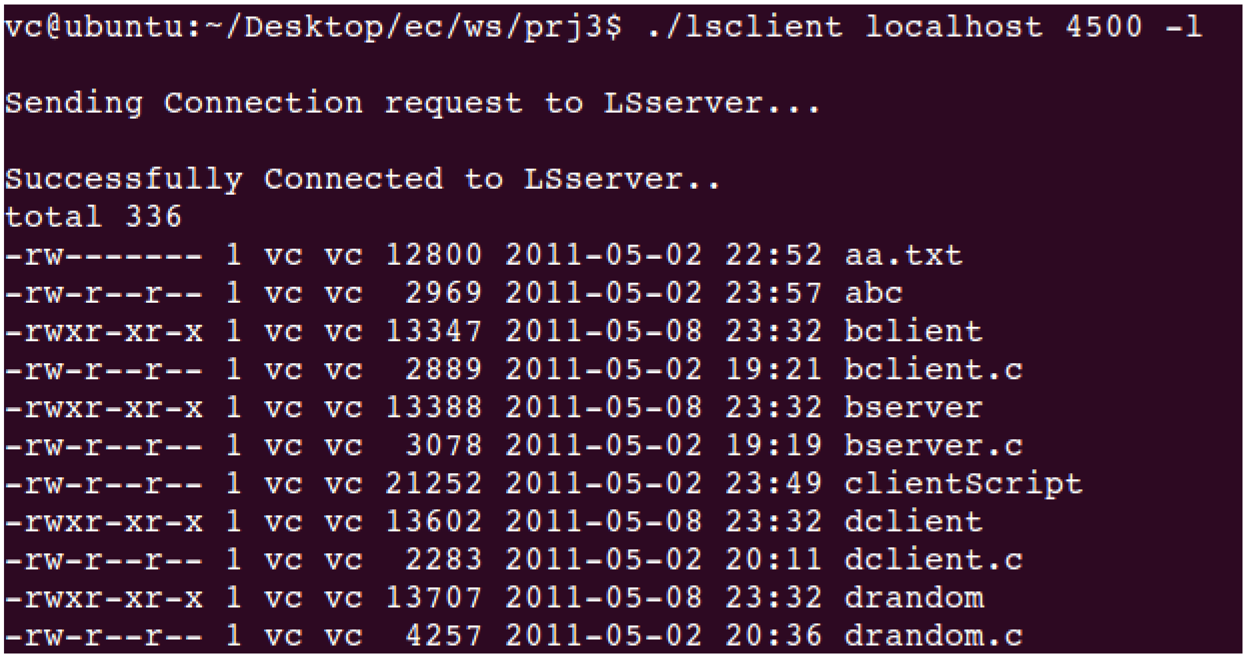
```
vc@ubuntu:~/Desktop/ec/ws/prj3$ ./bclient localhost 4500 malayalam

Sending Connection request to Bserver...

Successfully Connected to Bserver..
Request : malayalam

malayalam

vc@ubuntu:~/Desktop/ec/ws/prj3$ ./bclient localhost 4500 clientrequest

Sending Connection request to Bserver...

Successfully Connected to Bserver..
Request : clientrequest

tseuqertneilc

vc@ubuntu:~/Desktop/ec/ws/prj3$ 
```

Snapshot of Program1: Basic Client Server Program to Reverse a String

PROGRAM2: DIRECTORY LISTING SERVER



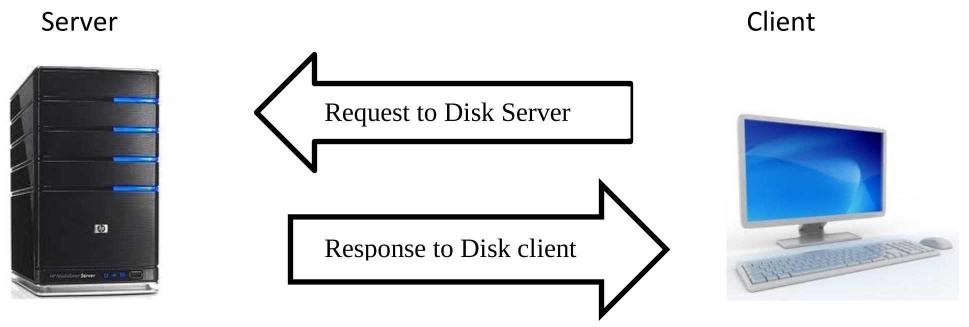
This is a similar implementation of Basic Client Server model explained in Program 1. Here the client sends the request parameters for listing server directories on the server from the client side with extra parameters. The server executes these commands with exec command and writes the output to the client socket with dup2 command. This output is read by the client on the client socket and printed to the console. Again here the parameters are received as command line arguments.



```
vc@ubuntu:~/Desktop/ec/ws/prj3$ ./lsclient localhost 4500 -l

Sending Connection request to LSserver...

Successfully Connected to LSserver..
total 336
-rw------- 1 vc vc 12800 2011-05-02 22:52 aa.txt
-rw-r--r-- 1 vc vc  2969 2011-05-02 23:57 abc
-rwxr-xr-x 1 vc vc 13347 2011-05-08 23:32 bclient
-rw-r--r-- 1 vc vc  2889 2011-05-02 19:21 bclient.c
-rwxr-xr-x 1 vc vc 13388 2011-05-08 23:32 bserver
-rw-r--r-- 1 vc vc  3078 2011-05-02 19:19 bserver.c
-rw-r--r-- 1 vc vc 21252 2011-05-02 23:49 clientScript
-rwxr-xr-x 1 vc vc 13602 2011-05-08 23:32 dclient
-rw-r--r-- 1 vc vc  2283 2011-05-02 20:11 dclient.c
-rwxr-xr-x 1 vc vc 13707 2011-05-08 23:32 drandom
-rw-r--r-- 1 vc vc  4257 2011-05-02 20:36 drandom.c
```

Snapshot of Program2: Directory listing server

PROGRAM3: BASIC DISK SERVER



Program 3 is the design and implementation of a Basic Disk Server Protocol. This Program followed the Disk Server Protocol that is listed below. It reads the client request from the socket and performs the necessary operation and writes the response to the client socket. The client keeps running till it is stopped manually by an ‘Exit’ Command or by ^C. The various commands given by the client are explained in the Disk Protocol.

The Disk server is started with the number of tracks and sectors for the disk. Here a file acts as the secondary storage and is of the size track * sector * 128, where 128 is the block size. The basic disk client commands are to write and read from a specific track and sector, or the command I to get the info of the number of track and sectors in the disk. Whatever written to the file are written to a 128 byte block and it is persistent. A read command reads the 128 byte with a 1 or 0 for success or failure with 128 bytes of data where null are replaced by 0’s. The basic disk server uses mmap to map the memory of 128 bytes.

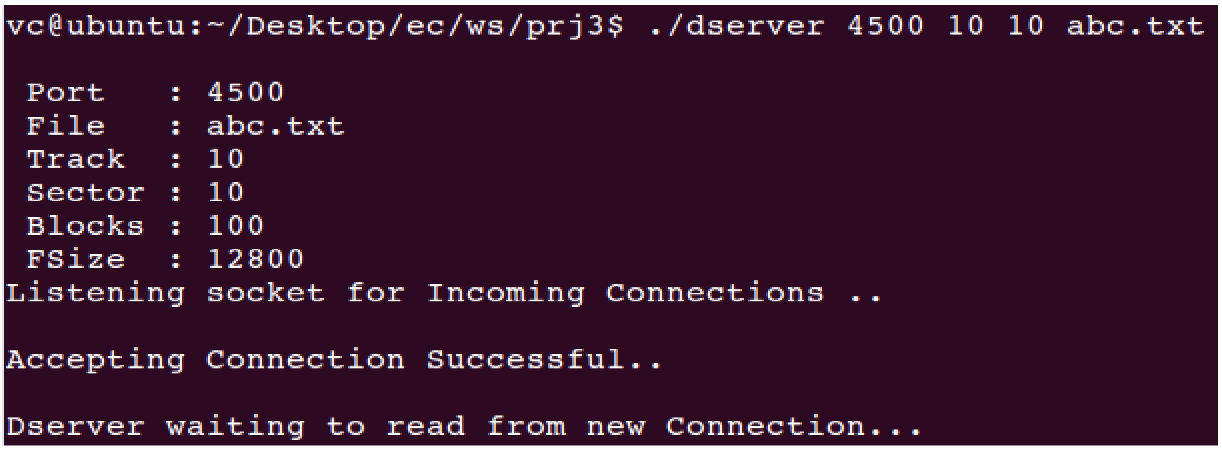
The program 3 also has a random client that generates a random value to read and write to work the testing of the server. The number of iterations is given as command line argument to random client.

The Disk Protocol:

The server must understand the following commands, and give the following responses:

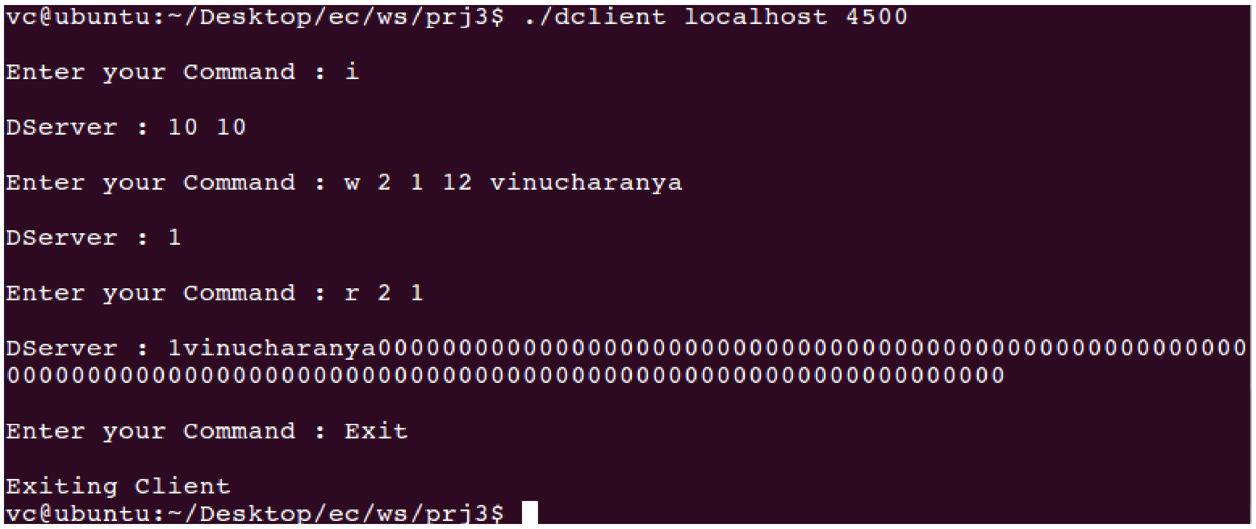
- ✓ I: information request. The disk returns two integers representing the disk geometry: the number of cylinders, and the number of sectors per cylinder.
- ✓ R c s: read request for the contents of cylinder c sector s. The disk returns ’1’ followed by those 128 bytes of information, or ’0’ if no such block exists. (This will return whatever data happens to be on the disk in a given sector, even if nothing has ever been explicitly written there before.)
- ✓ W c s l data: write request for cylinder c sector s. l is the number of bytes being provided, with a maximum of 128. The data is those l bytes of data. The disk returns ’1’ to the client if it is a valid write request (legal values of c, s and l), or returns a ’0’ otherwise. At what point in time this return code is sent, depends on the setting of the S synchronization mode.

Snapshots for Program3



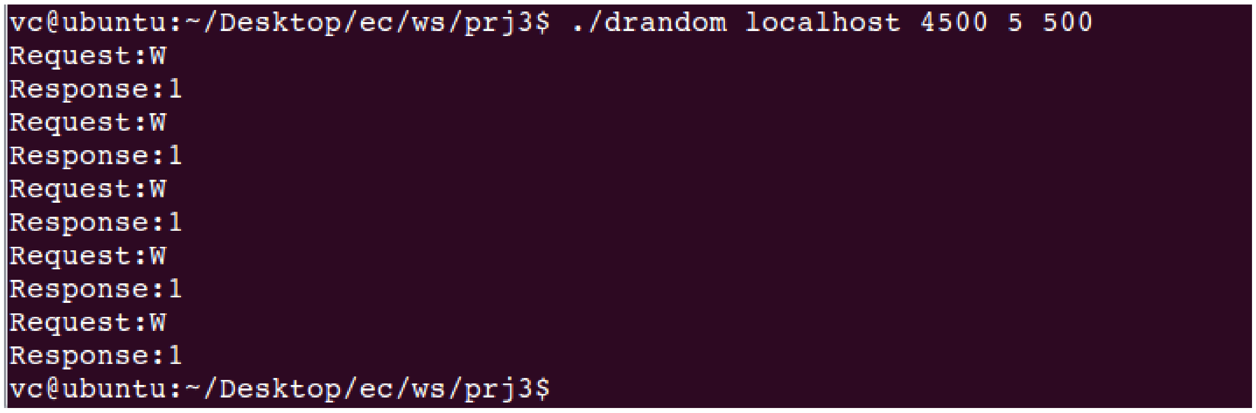
```
vc@ubuntu:~/Desktop/ec/ws/prj3$ ./dserver 4500 10 10 abc.txt

 Port   : 4500
 File   : abc.txt
 Track  : 10
 Sector : 10
 Blocks : 100
 FSize  : 12800
Listening socket for Incoming Connections ..

Accepting Connection Successful..

Dserver waiting to read from new Connection...
```

DSERVER SNAPSHOT



```
vc@ubuntu:~/Desktop/ec/ws/prj3$ ./dclient localhost 4500

Enter your Command : i

DServer : 10 10

Enter your Command : w 2 1 12 vinucharanya

DServer : 1

Enter your Command : r 2 1

DServer : 1vinucharanya000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000

Enter your Command : Exit

Exiting Client
vc@ubuntu:~/Desktop/ec/ws/prj3$
```

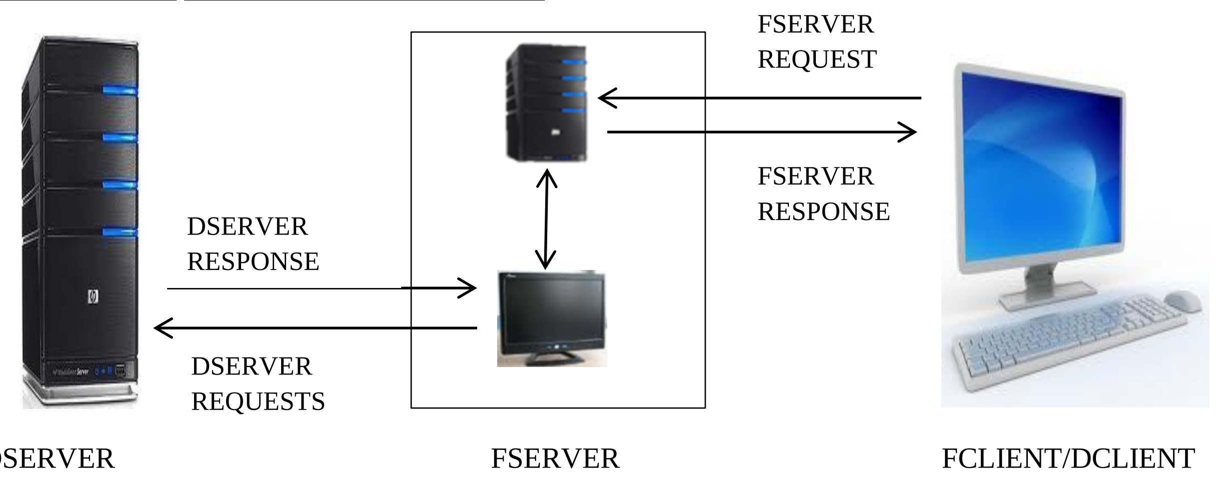
DCLIENT SNAPSHOT



```
vc@ubuntu:~/Desktop/ec/ws/prj3$ ./drandom localhost 4500 5 500
Request:W
Response:1
Request:W
Response:1
Request:W
Response:1
Request:W
Response:1
Request:W
Response:1
vc@ubuntu:~/Desktop/ec/ws/prj3$
```

DRANDOM CLIENT SNAPSHOT

PROGRAM4: FILE SYSTEM SERVER



Program 4 is the design and implementation of File System Server that follows the file system server protocol given below. This program uses the same dserver created in program 3 and same dclient or fclient can be used. The file system server is implemented in Fserver. In this program, unlike program3 where dclient contacts the server, fserver sends requests to the server and fclient has no contact with the server. Fserver serves as the server to fclient which processes requests from fclient, changes it into appropriate commands for dserver, sends the request and retrieved the result from the dserver again does the necessary operations and sends the user needed output to the fclient.

Basically the fserver is implementing a file system server. In order to accomplish this, the fserver uses two data structures, a Fat table and a File table. The user can create, write, read, list and delete files. Every time any operation is performed, the File table and Fat table are checked and modified accordingly. He can also format the entire table that erases all the entries from the table.

The various commands and the operations are as follows:

F: This command is used to format the file table. When this command is issued, all the entries in the table are set to null and zero's accordingly.
C: This command is used to create a file name according to the protocol. Whenever a file is attempted to create, the Fat table and file table are checked for free blocks and then they are allocated and the file is created.
W: Writes the entered length of data to the given filename. The data is separated into 128 bytes and a W command is sent in appropriate format to the dserver which writes the data to the file according to the free blocks. Everytime free blocks are taken while sending the command.
R: Reads the contents of the filename. Again the fileserver on receiving the command, checks the file table and fat table, and sends the necessary number of multiple read commands to the dserver, concatenates the necessary data and sends it to the fclient.
D: Deletes the file entry from the file table and frees all the blocks for the associated file in the Fat table.
L: Lists all the file names or the file name along with the size based on the L parameter given.

Help: There is also a help command to help the user about the format of each command to be given and possible commands.

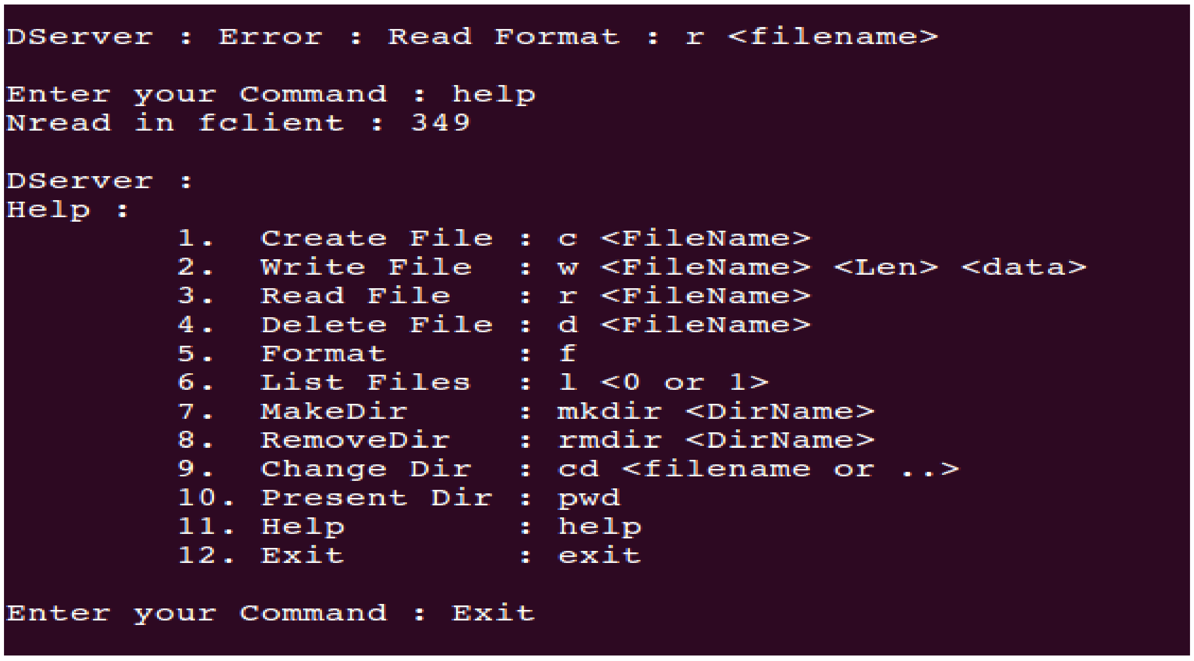
Exit: Exits the client gracefully

Whenever a user types a wrong command, the command format is given. Whenever an error occurs, the cause of the error is given and help list is available for more details.

The Filesystem Protocol

The server must understand the following commands, and give the following responses.

- ✓ ***F:*** format. Formats the filesystem on the disk, by initializing any/all of tables that the filesystem relies on.
- ✓ ***C f:*** create file. This will create a file named f in the filesystem. Possible return codes: 0 = successfully created the file; 1 = a file of this name already existed; 2 = some other failure (such as no space left, etc.).
- ✓ ***D f:*** delete file. This will delete the file named f from the filesystem. Possible return codes: 0 = successfully deleted the file; 1 = a file of this name did not exist; 2 = some other failure.
- ✓ ***L b:*** directory listing. This will return a listing of the files in the filesystems. b is a boolean flag: if ’0’ it lists just the names of all the files, one per line; if ’1’ it includes other information about each file, such as file length, plus anything else your filesystem might store about it.
- ✓ ***R f:*** read file. This will read the entire contents of the file named f, and return the data that came from it. The message sent back to the client is, in order: a return code, the number of bytes in the file (in ASCII), a white-space, and finally the data from the file. Possible return codes: 0 = successfully read file; 1 = no such filename exists; 2 = some other failure.
- ✓ ***W f l data:*** write file. This will overwrite the contents of the file named f with the l bytes of data. If the new data is longer than the data previous in the file, the file will be made longer. If the new data is shorter than the data previously in the file, the file will be truncated to the new length. A return code is sent back to the client. Possible return codes: 0 = successfully written file; 1 = no such filename exists; 2 = some other failure.



```
DServer : Error : Read Format : r <filename>

Enter your Command : help
Nread in fclient : 349

DServer :
Help :
        1.  Create File : c <FileName>
        2.  Write File  : w <FileName> <Len> <data>
        3.  Read File   : r <FileName>
        4.  Delete File : d <FileName>
        5.  Format      : f
        6.  List Files  : l <0 or 1>
        7.  MakeDir     : mkdir <DirName>
        8.  RemoveDir   : rmdir <DirName>
        9.  Change Dir  : cd <filename or ..>
        10. Present Dir : pwd
        11. Help        : help
        12. Exit        : exit

Enter your Command : Exit
```

Snapshot for Help Command and Read Format Error in Program 4

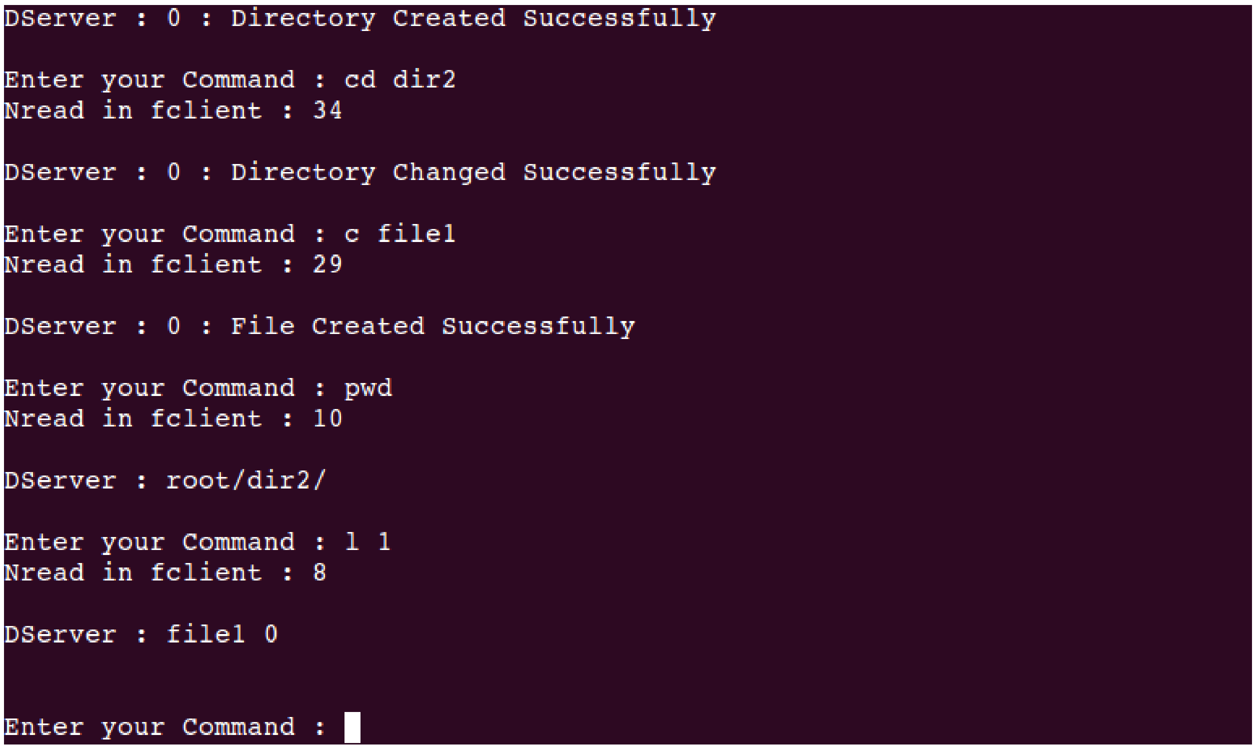
PROGRAM5: DIRECTORY STRUCTURE

Program 5 is extra design and implementation on program 4 which add the functionality of a directory structure to file system Server. The user can create, change and delete directories from fclient. He can also check the present working directory by the following commands.

Directory Structure Commands:

- ✓ mkdir dirname : create a directory of name dirname.
- ✓ cd dirname : change current working directory to dirname
- ✓ pwd : print the working directory name.
- ✓ rmdir dirname: remove the directory given. Throw an error if it is not present.

Whenever a new directory is created, its directory status is stored in the file table and also its parent directory index is stored while creating sub-directories. Any error that occurred during the process is accordingly reported to the fclient. All operations done on a file system can be done from within the directories.



```
DServer : 0 : Directory Created Successfully

Enter your Command : cd dir2
Nread in fclient : 34

DServer : 0 : Directory Changed Successfully

Enter your Command : c file1
Nread in fclient : 29

DServer : 0 : File Created Successfully

Enter your Command : pwd
Nread in fclient : 10

DServer : root/dir2/

Enter your Command : l 1
Nread in fclient : 8

DServer : file1 0


Enter your Command :
```

Snapshot for Program 5 Directory Structure