CLIENT-SERVER BASED FILE SHARING SYSTEM

Rajni Krishna Gandhar, Viram Shah, J N V Naveen Teja
School of Engineering, Santa Clara University

ACKNOWLEDGEMENTS

We would like to offer our special thanks to Professor Mr. Ming-Hwa Wang for providing us the opportunity to design and execute this project.

LIST OF CONTENTS

LIST OF FIGURES:

LIST OF TABLES:

ABSTRACT

The main objective of our project “Modelling a prototype for Client-Server based File Sharing System” is to develop a high performance network, whose primary purpose is to build a storage devices which stores files and allow users to access, share and modify it from anywhere. Generally users create files and save that on their hard drives which consumes their memory and if they want to share that file with any other user then personally they have to send their files using some services which is tedious work. The other factor is security issue i.e. while sharing the file, user is not sure that whether it reaches to the correct destination without tempering file. If user wanted to modify the file which it had saved on its hard drive, then either he has to access same computer or has to take manually that file in any other hard drive.

II. Introduction

"Building a prototype model for client-server based file sharing system."

The main objective of our project "Modelling a prototype for Client-Server based File Sharing System" is to develop a high performance network, whose primary purpose is to build a storage devices which stores files and allow users to access, share and modify it from anywhere. Generally users create files and save that on their hard drives which consumes their memory and if they want to share that file with any other user then personally they have to send their files using some services which is tedious work. The other factor is security issue i.e. while sharing the file, user is not sure that whether it reaches to the correct destination without tempering file. If user wanted to modify the file which it had saved on its hard drive, then either he has to access same computer or has to take manually that file in any other hard drive.

"Modelling a prototype for Client-Server based File Sharing System" will allow us to have good understanding for the client-server model. Multithreading which is necessary model for any Server will be studied in detail. File creation on server side after it is uploaded by user will also be understood. After requesting for the file, how the file is deployed on respective client machine is also discussed. File Sharing between users and giving access rights to another user is another aspect which is discussed. and User Authentication using different Cryptographic algorithm is also implemented. Memory management is also important aspect which will be covered in project.

Following aspects of the prototype model can be modernized or optimised in terms of memory optimization, user perception and scalability:

1. Dynamic memory allocation on server: Dynamic memory allocation on server allows us to optimally use the unused user space when assigned statically. That is it can be managed between more users as a result the 10GB space can be allocated to more than 10 users which is the user limiting case when assigned statically like 1GB per user.
2. Graphical User Interface: To enrich our application we can take our desktop based application to web based application with user friendly interface which will allow users to run our application with much more ease.
3. Scalability

when users requests for more memory than the room allocated to him/her, they can purchase it from the server memory or from its peers at comparatively low cost by using safe transaction method.

III. Theoretical basis and literature review.

The main focus of this paper is to model a prototype of client-server based file sharing system. The earliest design uses a central server (or server cluster) to coordinate participating nodes and to maintain an index of all available files being shared. When a peer node joins the system, it contacts the central server and sends a list of the local files that are available for other peers to download (shared files). To locate a file, a peer sends a query to the central server, which performs a database lookup and responds with a list of peers that have the desired file. If a peer leaves the system, its list of shared files is removed from the central server. We will denote such architecture as a CIA (Centralized Indexing Architecture). An example of such a system is the Napster network. There are two other architectures eliminate the central server and distribute the indices of available files among participating nodes; they differ from each other primarily in the manner in which they distribute the file indices. But the paper proposed in this paper deals about fundamental prototype model of client-server based file sharing system. Broadly the paper can be divided into 3 major modules. Mainly, User authentication, file organization and sharing and Multi-user request handling. Let us look at each of these modules in discrete and the technical background associated to it.

Firstly, User authentication is a means of identifying the user and verifying that the user is allowed to access some restricted service. That is when user logs in to the server, you verify the authenticity of the server whether he is allowed to access the server resources or not. Once user gets through the authentication phase he gets the full privilege to access his files on the server.

Authentication can be accomplished in many ways. The importance of selecting an environment appropriate Authentication Method is perhaps the most crucial decision in designing secure systems. Authentication protocols are capable of simply authenticating the connecting party or authenticating the connecting party as well as authenticating itself to the connecting party. This overview will generalize several Authentication Methods and Authentication Protocols in hopes of better understanding a few options that are available when designing a security system. First let's look at passwords; passwords are the most widely used form of authentication. Users provide an identifier, a typed in word or phrase or perhaps a token card, along with a password. In many systems the passwords, on the host itself, are not stored as plain text but are encrypted. Password authentication does not normally require complicated or robust hardware since authentication of this type is in general simple and does not require much processing power. Password authentication has several vulnerabilities, some of the more obvious are: Password may be easy to guess and discovering passwords by eavesdropping or even social engineering. To avoid the problems associated with password reuse, one-time passwords were developed. There are two types of one-time passwords, a challenge-response password and a password list. It is important to keep in mind that Password systems only authenticate the connecting party. It does not provide the connecting party with any method of authenticating the system they are accessing, so it is vulnerable to spoofing or a man-in-middle attack. Public key cryptography is based on very complex mathematical problems that require very specialized knowledge. Public key

cryptography makes use of two keys, one private and the other public. The two keys are linked together by way of an extremely complex mathematical equation. The private key is used to decrypt and also to encrypt messages between the communicating machines. Both encryption and verification of signature is accomplished with the public key. In this paper we designed a novel user authentication algorithm to authenticate the clients accessing the server which will be dealt in detail coming paragraphs.

Secondly, the major functionality of our project is implemented in file organization and sharing module. Network file sharing is the process of copying files from one computer to another using a live network connection. This paragraph describes the different methods and networking technologies available to help you share files. Let's begin with FTP File Transfers, File Transfer Protocol (FTP) is an older but still popular method to share files on the Internet. A central computer called the FTP server holds all the files to be shared, while remote computers running FTP client software can log in to the server to obtain copies. Coming to P2P - Peer to Peer File sharing, it is an extremely popular method for swapping large files on the Internet, particularly music and videos. Unlike FTP, most P2P file sharing systems do not use any central servers but instead allow all computers on the network to function both as a client and a server. Next comes the famous email, files have been transferred from person to person over a network using email software. Emails can travel across the Internet or within a company intranet. Like FTP systems, email systems use a client/server model. The sender and receiver may use different email software programs, but the sender must know the recipient's email address, and that address must be configured to allow the incoming mail. Finally coming to Online sharing services, numerous Web sites built for community file sharing exist on the Internet. Members post or upload their files to the site using a Web browser, and others can then download copies of these files using their browser. Our project focuses on implementing a prototype of online sharing services on the basis of client server model. The functionality of this nodule is explained in detail in successive paragraphs.

In order to facilitate multiple clients accessing server at the same time, the third module focuses on handling multi-client requests and associated conflicts in accessing the server resources. This can be implemented using multithread concept of operating system. In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution. In the same multithreaded process in a shared-memory multiprocessor environment, each thread in the process can run on a separate processor at the same time, resulting in parallel execution. When the process has fewer or as many threads as there are processors, the threads support system in conjunction with the operating environment ensure that each thread runs on a different processor. For example, in a matrix multiplication that has the same number of threads and processors, each thread (and each processor) computes a row of the result. Traditional UNIX already supports the concept of threads--each process contains a single thread, so programming with multiple processes is programming with multiple threads. But a process is also an address space, and creating a process involves creating a new address space. Creating a thread is much less expensive when compared to creating a new process, because the newly created thread uses the current process address space. The time it takes to switch between threads is much less than the time it takes to switch between processes, partly because switching between threads does not involve switching between address spaces. Communicating

between the threads of one process is simple because the threads share everything--address space, in particular. So, data produced by one thread is immediately available to all the other threads. The interface to multithreading support is through a subroutine library, libpthread for POSIX threads, and libthread for Solaris threads. Multithreading provides flexibility by decoupling kernel-level and user-level resources.

The solution proposed in this paper is novel in terms of user authentication which differs from other papers. We tried to address the disadvantages of authentication methods such as plain password authentication are easily implemented but are in general weak and primitive. The fact that plain password authentication it is still by far the most widely used form of authentication, gives credence to the seriousness of the lack of security on both the Internet and within private networks. Also, we have designed our own protocol for client-server message interactions whose simplicity should is to be tested against real time performance indices.

IV. Methodology

We are designing and implementing a small model of file sharing system. This system uses a client - server architecture to make this thing happen and on top of client-server architecture we are using some concepts of file system to allocate some storage to users. Here mainly we have 6 tasks, these tasks are performed by the client and server to provide services to user.

Tasks are as follows:

1. Users can register to get the benefits of file sharing services offered by file sharing system.

2. Server will always does user authentication.
3. Users can upload files on the file sharing system (server).
4. User can download the file from the file sharing system(server).
5. Users can share their file with other registered users.
6. Users can delete the file from the file sharing system.

Workflow of all the above tasks:

In the workflow, we are going to explain briefly about all the steps our file sharing system will take in order to serve the requesting service.

1. **Registration Workflow:**
   - User sends registration request with register command, and gives username and password as the parameters of the register command, using file sharing client (client performs encryption on password before sending using the shared key between client and server) to file sharing system (file sharing server).
     For example "*register* rajni abc".
   - File sharing system (server) receives the registration request and makes an entry of this username in the database with the encrypted form of the password (for the security purpose), makes a directory named "userId" in the file system allocates some storage (in our case 1GB) to this user.

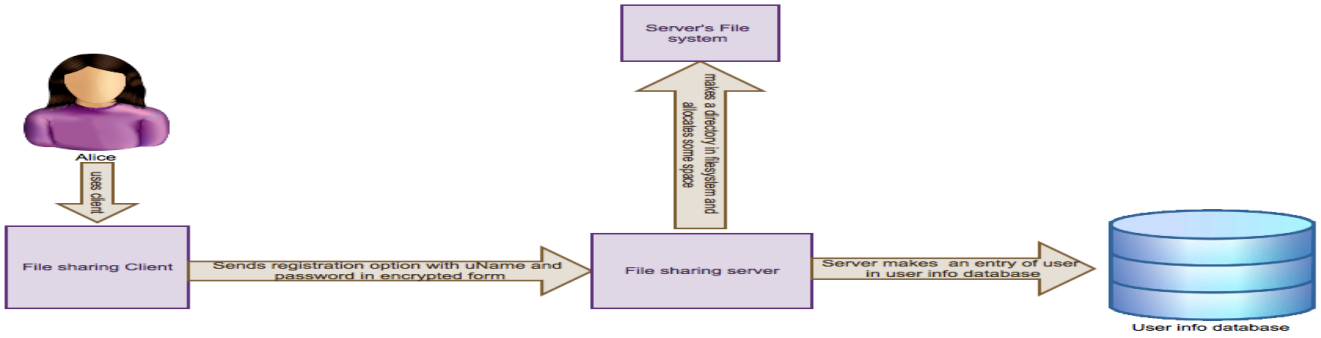


Fig. 1 Flow diagram of the registration workflow

2. **User Authentication Workflow:**
   - Before allowing any user to use the services of the file sharing system, system (server) first validates that the user requesting for the service is a registered user (valid) or not.
     To do that server follows the following steps:
   - Receives username and the password and performs a search in the user information database to find an entry of the user with the given username.
   - If server finds an entry in the database for the requesting user it further proceeds for the the password verification otherwise it sends a message to user that says username is not valid.

Server gives maximum three attempts to user to re-enter the username, if user fails to provide correct username in all the attempts then after 3rd attempt server immediately drops the connection.

- if server finds that the password provided by the requesting user is correct then only it allows user to use the services otherwise sends a message to user that password doesn't match. Server gives maximum three attempts to user to re-enter the password, if user fails to provide correct password in all the attempts then after 3rd attempt server immediately drops the connection.

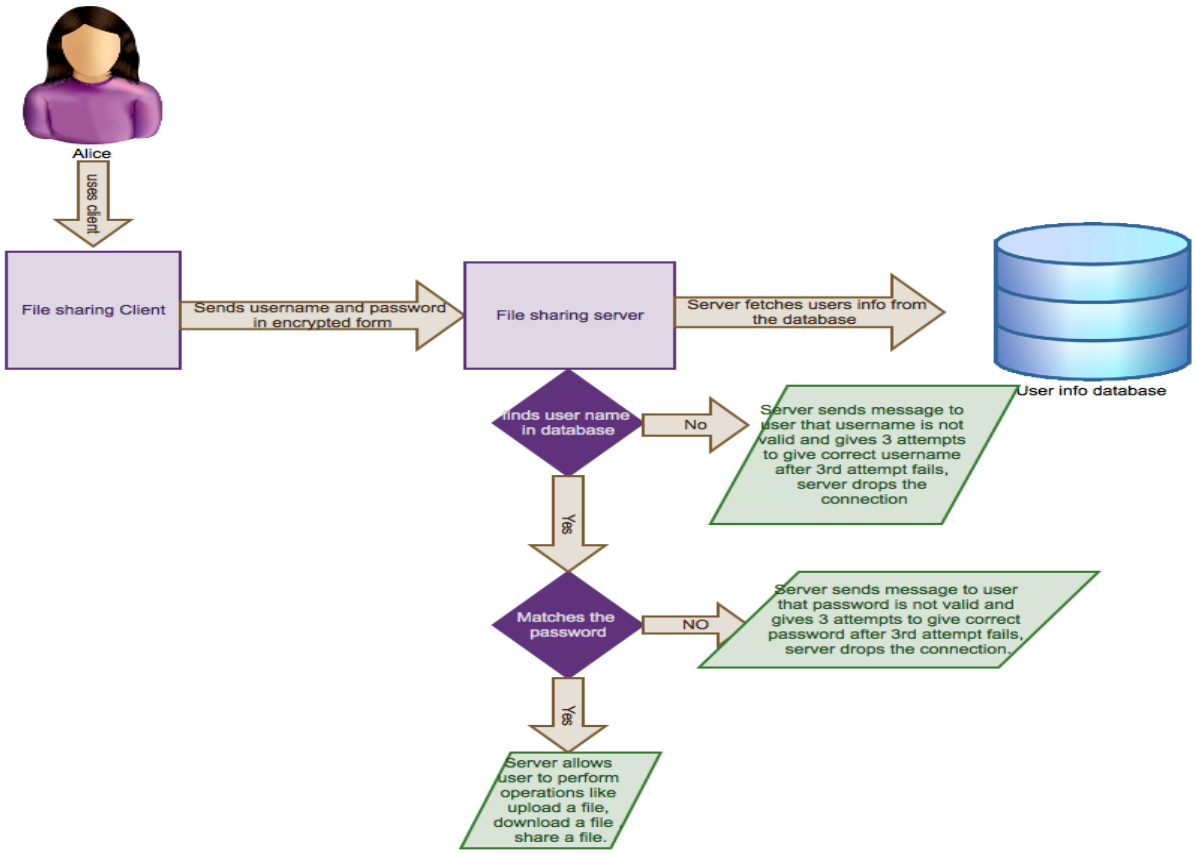


Fig. 2 Flow Diagram Of User Authentication WorkFlow

3. File Upload Workflow

- User sends file upload request with upload command, and gives filename (file path) as the parameter of the upload command, using file sharing client to file sharing system (file sharing server).

  For example "*upload* /home/rajni/xyz.txt".
- Server receives file upload request and checks that service requesting user has enough space available or not on the basis of the file size user wants to upload.
- If server finds that user has enough available space it puts user's file named "xyz.txt" in the user's directory named as "userId" of this user. Otherwise server sends a message to user that you don't have enough storage available to upload the file "xyz.txt".

- After uploading the file server updates the available storage of the user by subtracting the size of the uploaded file from the current available storage of the user.

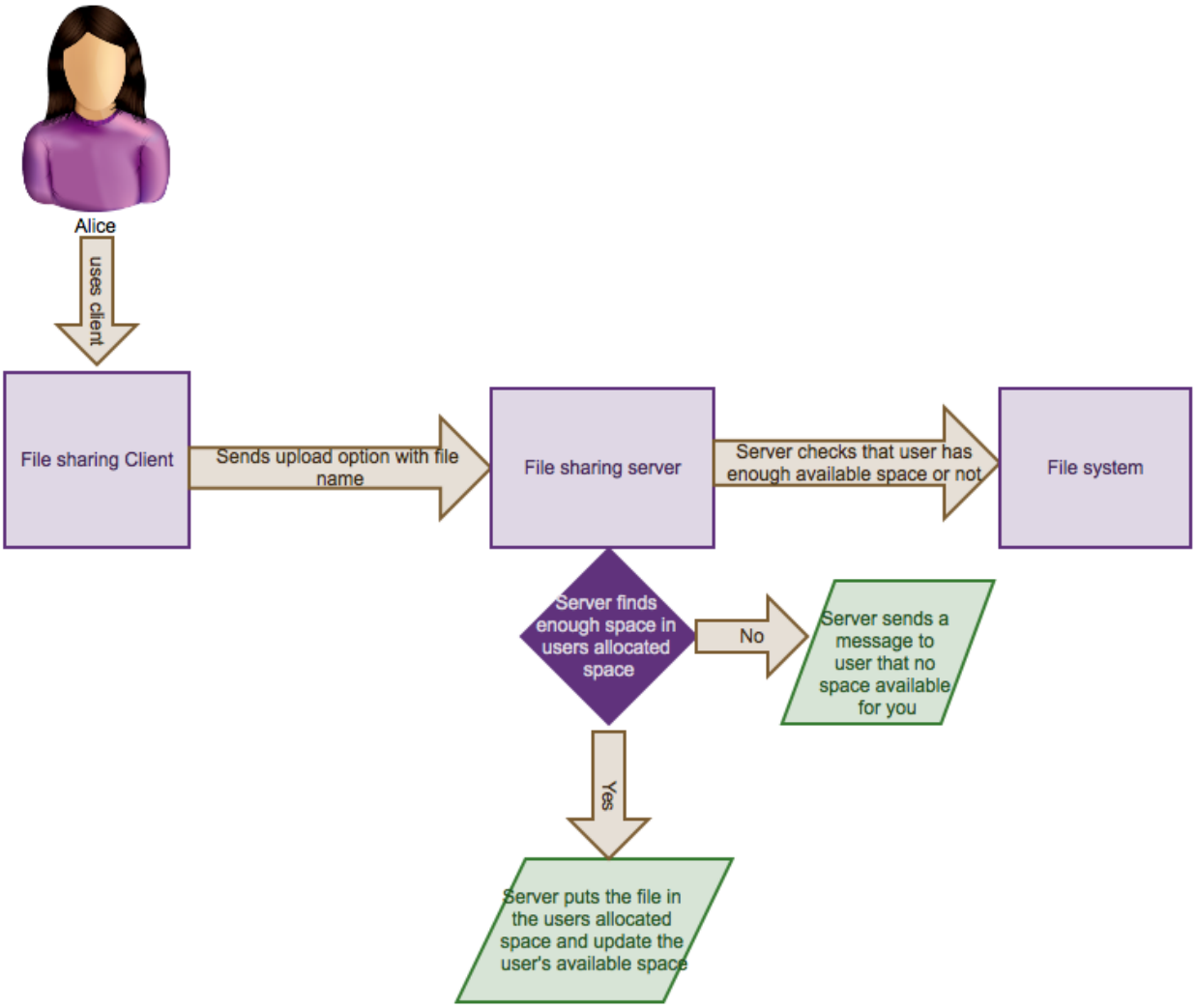


Fig.3 Flow Diagram Of File Upload WorkFlow

4. File Download WorkFlow:

- User sends file download request with download command, and gives filename and a path where user wants to save file in its local machine as the parameters of the download command, using file sharing client to file sharing system (file sharing server).
  For example "*download* xyz.txt /home/rajni/".
- Server receives the download request from the user and checks in the user's directory named as "userId" that the requesting file ("xyz.txt") for downloading resides in the requesting user's directory or not.
- If server finds the requesting file in the user's directory, it sends the file to the requesting client and then client saves the received file in the place specified by the user as the second parameter

of the command.

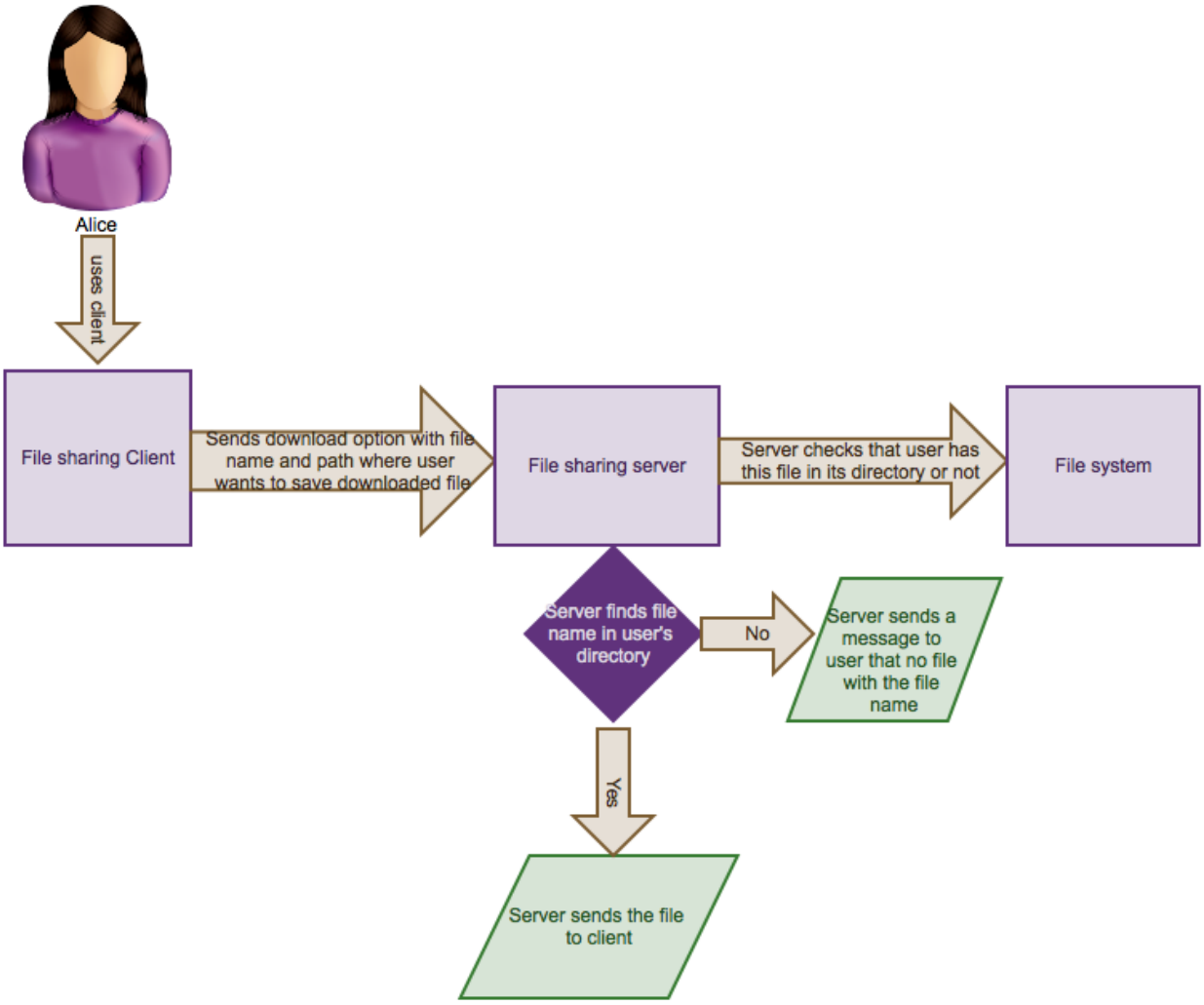


Fig.4 Flow Diagram Of File Download WorkFlow

5. File Share WorkFlow:

- User sends file sharing request with share command, and gives filename, sharing mode (read mode, write mode, executable mode) and other users' usernames, user wants to share file with as the parameters of the share command, using file sharing client to file sharing system (file sharing server).

  For example "*share* xyz.txt -r rajni".

  Note: number of usernames must be equal to the number of sharing modes.
- Server receives the sharing request and checks in the user's directory named as "userId" that the requesting file for sharing resides in the requesting user's directory or not.
- if server finds the file in the requesting user's directory then it proceeds for the given usernames'

verification for the sharing otherwise sends a message to the user that requesting file for sharing doesn't presents in your directory.

- In usernames' verification server checks for all the usernames that username has an entry in the database i.e. user you want to share file with is a valid registered user in file sharing system.
- If it finds users are valid then it changes the access permissions of the requesting file for the given usernames according to the sharing mode. Otherwise it sends a message to sharing requesting user that the users you want to share file with are not valid registered users on file sharing system.

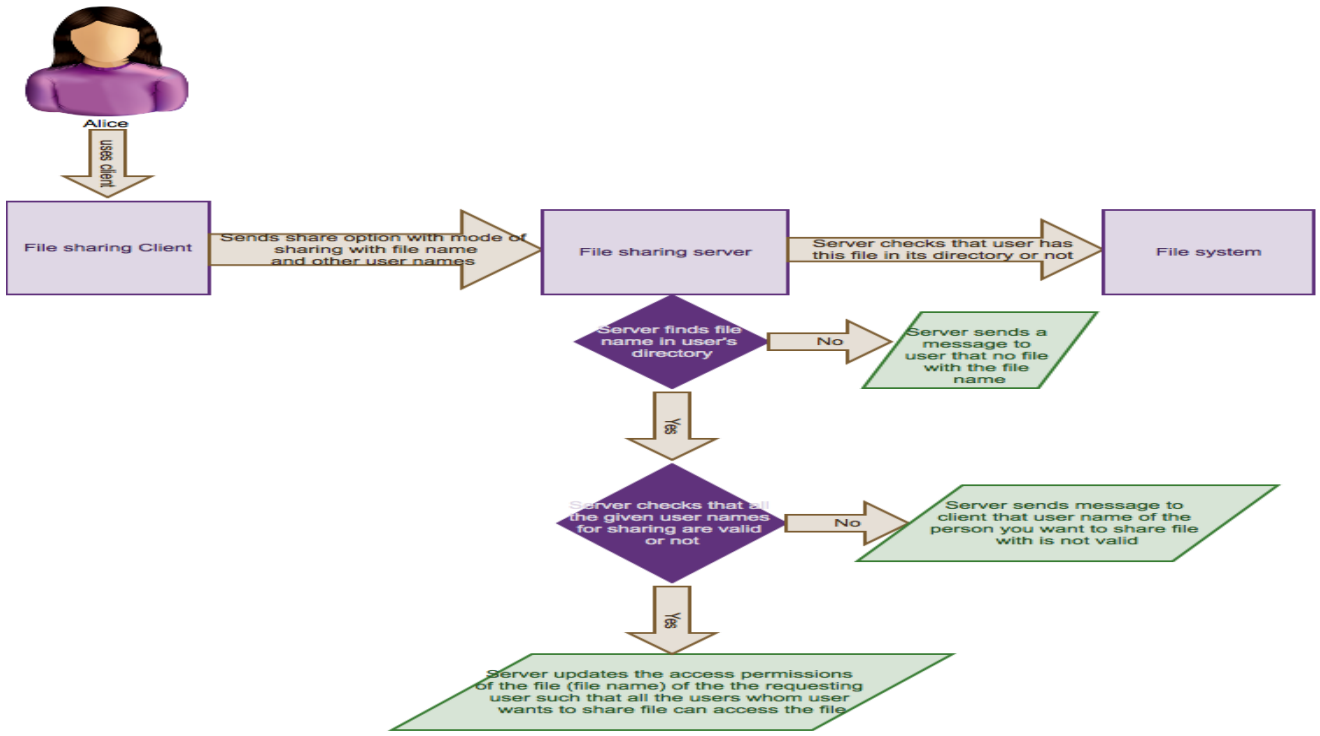


Fig. 5 Flow Diagram Of File Share Workflow

6. Delete File WorkFlow:

- User sends file delete request with delete command, and gives filename as the parameter of the upload command, using file sharing client to file sharing system (file sharing server).
  For example: delete xyz.txt
- Server receives the delete request from the user and checks in the user's directory named as "userId" that the requesting file ("xyz.txt") to delete resides in the requesting user's directory or not.
- If server finds the file in the user's directory then it removes the entry of the file from the user's directory and updates the current available space of user by doing addition of the file size in current available space. Otherwise sends a message to user that file you are requesting to delete doesn't exist in your directory.

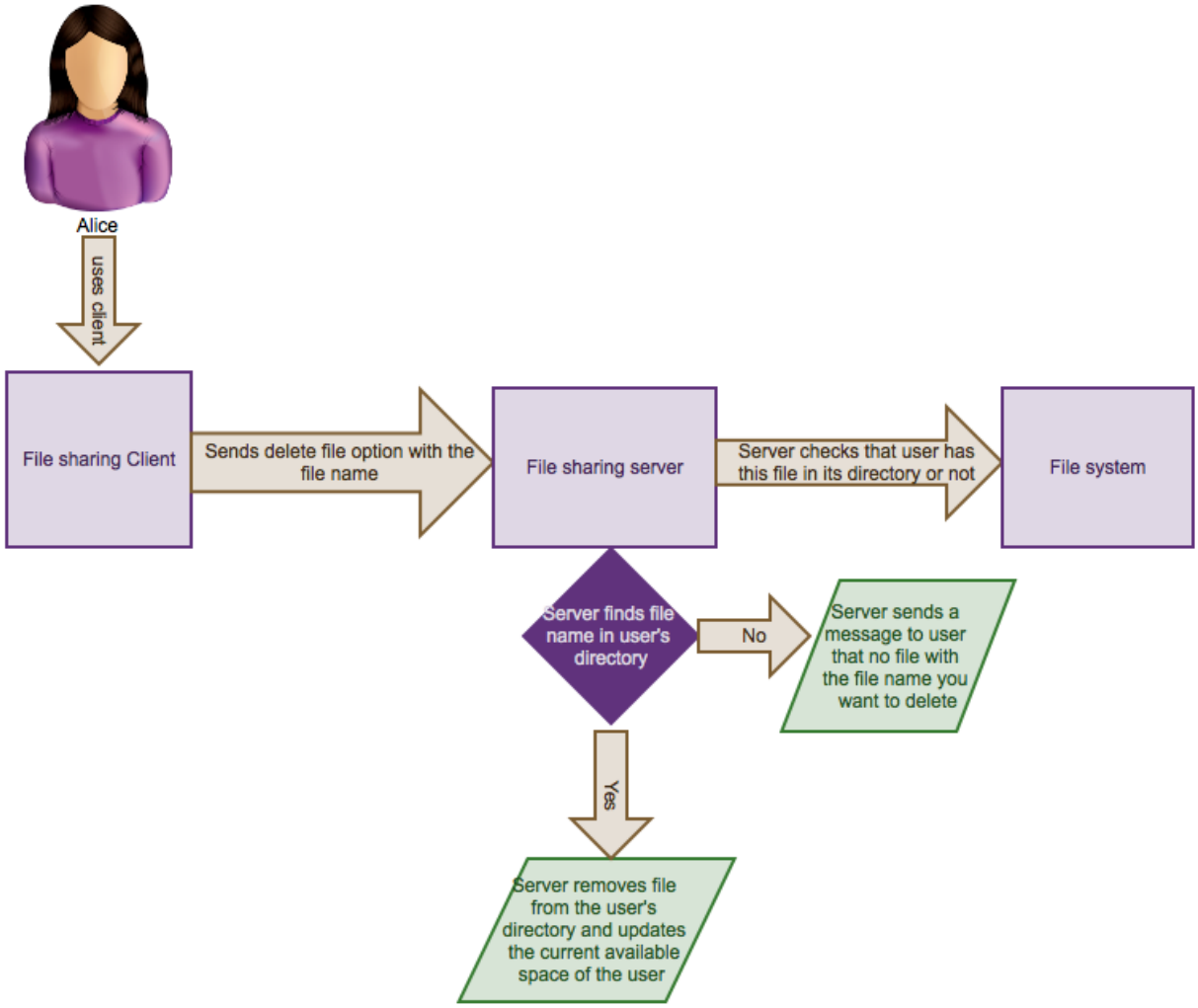


Fig.6 Flow Diagram Of The Delete WorkFlow

V. Implementation

Design Document:-

The prototype model designed has two client two versions GUI and command line. In either of the cases the server is programmed in C to ensure faster computation, while GUI version of client is programmed in Java and the command line version in coded in C. The current version of release supports limited commands. The Code is designed in such a way that its modular structure enables one to easily understand the implementation processes and and also offers more scalability to append new features in the long run. Broadly the code can be sectioned into three major categories Client , Server and Library. Client module holds all the associated code files and functions that contribute to client functionality, Server module has all the server functionality related code files while Common has code

files which depict the functionality common for both client and server. Lets look at each of these modules in separate, firstly Client

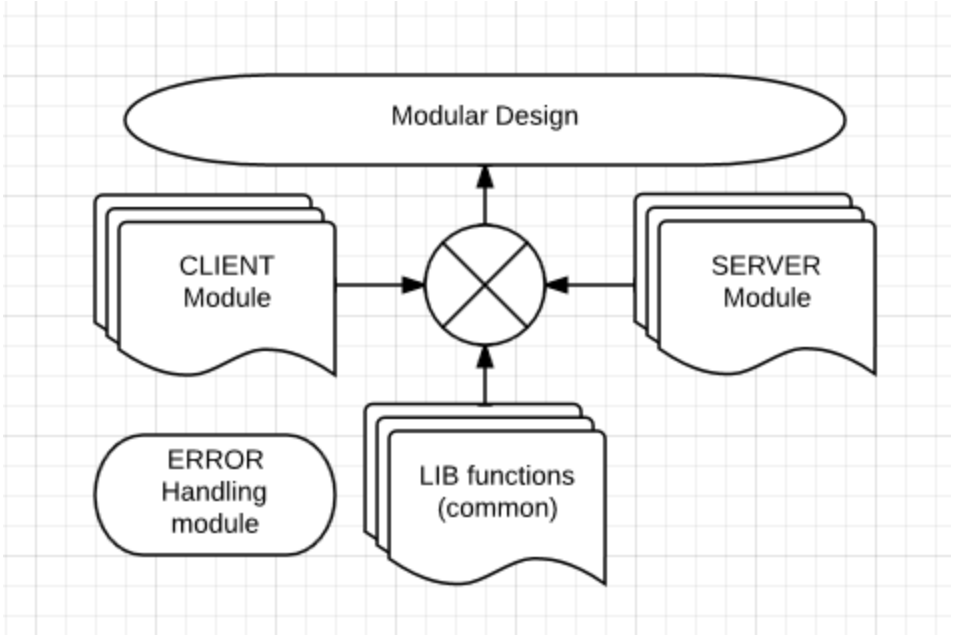


Fig 5.1 Modular design

Client Module:

1. client.c

All client operation will be initialized using this file. For secure connection with server, client.c sends Server's HOST ID and PORT NO. as parameters to connectivity.h. After connection is made between client and server i.e. after getting successful sockfd and key parameter from connectivity.h, Client keeps on executing in loop until there is quit command or there are no commands from client. If user enters "help" command then there will be output on the screen related all instructions about commands. The commands that are executed by the client is first getting fetched word by word and is being saved in buffer. According to command words that are being fetched and saved that in buffer client.c sends respective commands along with sockfd parameter to cmdHandler.h for executing commands. For all instruction there are no.of parameters that are required to execute in a correct state, if such parameters are not given properly or are invalid then client. c throws error like "file name too long" ,"invalid no.of arguments", "username too long", "Path too long", "Permission string too long" and "unknown command".

2. connectivity.h & connectivity.c

Connectivity.h is used for client server connection. It takes input from client.c as host and port no. of server and tries to connect with server. If connectivity.h gets input from server as true then connection is established and it saves sockfd and sends it to client. If connection is not established then accordingly it will show its error messages. When client.c sends quit as a parameter then connection with server is

closed using close function so that socket fd is closed.

3. cmdHandler.h & cmdHandler.c

This file handles the core functionality of dealing with commands to be issued to the server and also in deserializing the error messages send by server. Following are the commands that a user can issue to server in the current release of version:

Command	Description
client <host-name> <port-number>	To start the client.
acceptreq <file-name> <user-id>	To accept a pending request.
delete <file-name>	To delete a file which you own.
deleteaccount	To delete your account. All files will be deleted and all sharing will go away.
denyreq <file-name> <user-id>	To deny a pending request.
download <file-name> <file-path>	To download your own file.
downloadshared <file-name> <owner-id> <file-path>	To download a file which someone has shared with you.
editprofile	To update your profile details.
login	To login with an existing account.
logout	To logout from the currently logged in account.
listall	To list files all the files visible to you.
listownedbyme	To list files which are owned by you.
listsharedwithme	To list files which are shared with you by other users.
listsharedbyme	To list files which you have shared with others.
listsharereqs	To list all the pending sharing requests for you from others.
quit	To quit and shutdown the client.
register	To register/create a new account.
rename <old-file-name> <new-file-name>	To rename a file owned by you.
share <file-name> <permissions> <user-id>	To send a file sharing request to a user

showprofile	To show your profile details.
upload <file-name> <file-path>	To upload a file (owned by you) on to the server.
uploadshared <file-name> <owner-id> <file-path>	To upload a shared file (shared in RW mode).
unsharefrom <file-name> <user-id>	To remove an existing sharing or cancel the share request from a user.
unsharefromall <file-name>	To remove an existing sharing or cancel the share request from all the users.
help	To print this help message

Table 1: Commands and their Description

Common Module:

1. sendReceive.h & sendReceive.c

This file plays a role of mediator between client and server. All kind of communication between client and server is rightly done using this file. This file can be called as communication channel between client and server. Generally all files that are sent from either of the side are first sent to this file and then this file forward it accordingly. Before sending buffer or Byte integer i.e. file data to any of the side, sendReceive.h sends data to crypto.c for encryption after it receives encrypted text from crypto.c then it sends to client/server. This file is also used for receiving buffer or Byte integers, after receiving it sends data to crypto.c for decrypting it and then sends to client/server. Even messages are sent to the respective sides using sendReceive.h. sendReceive.h generally sends True/False to sender accordingly task performed.

2. misc.h & misc.c

These File is handles multiple cleanup and acknowledge functions like freeing of un-used buffer, it takes error message from sender and then sends to receiver, if some operations are not performed correctly and needs to abort instantaneously, generation of encryption key pair etc.

3. Crypto.h & Crypto.c

Basically this file generally deals with encryption and decryption process. Input are taken from sendReceive.h and then either encryption or decryption is processed as per the requirement. The output is either plaintext or ciphertext as per the requirement and is forwarded to sendReceive.h

Server Module:

1. server.c

This file generally deals with the response part. All the request made by client are processed and then its response is given. It also initializes database and even deals with multithreading concept. It checks whether server has enough access permission on database as well as root directory. when new client arrives it generally create new thread for it and return to main thread. It uses handle command to execute all instruction by a specified client. server.c will send these commands in a variable to cmdHandler.h. After client has been assigned thread it even generate public key which it will be using for that specific client only. Input and Output both are taken and return to sendReceive.h

2. cmdHandler.h & cmdHandler.c

This file deals with handling of commands sent by different clients. Main functions of this section include error handling, checking the integrity of command issued by client, routing the command to appropriate command handler, decryption/encryption, deserializing command buffer, preparing command to be send to client and setting all its variables to free.

3. userCmds.h & userCmds.c

This file handles all the commands related to user on the server front. As seen from above the commands from client, this module bears the responsibility of handling user related commands like: register, login, logout, editprofile etc.

4. FileCmds.h & FileCmds.c:

This file handles all the commands related to files on the server front. As seen from above the commands from client, this module bears the responsibility of handling file related commands like: list, upload, download, share, delete, rename etc. It generally interacts with database to get the required info for file operation.

5. dbServer.h & dbServer.c

This file handles interaction of server with database server to fetch user and file related information as and when requested. following are the API's designed for this specific purpose:-

1. deleteUserAccountByLoginid: To delete user account by login ID;
2. deleteUserAccountByUid: To delete user account by Uid;
3. addUser: To add New User;
4. getAvailableSpace: To get available space for the user;
5. updateAvailableSpace: To update the availavble space if the user;
6. isFilePresent: To check whether the file is present;
7. addFileInfo: To add information about file;
8. removeFileInfo: to remove file information;
9. isUserPresent: To check if user is present;
10. isFileShared: To check whether the file is shared?;

11. addShareRequest: To add share request;
12. unshareFileFrom: To unshare file from designated peer;
13. unshareFileFromAll: To unshare file from all designated peer;
14. getUserNameFromUid: To get user name from UID;
15. getUidFromLoginId: to get UID from LoginID;
16. getLoginIdFromUid: to get Login ID from UID;
17. getAllFiles: To get all files of the requested user;
18. getPendingShareRequests: to fetch all pending share requets for an user;
19. getFilesListByFileName: to list the files by name;
20. renameFile: to rename the file;
21. handlePendingShareRequest: to handle pending share requests;
22. getAllFilesSharedByMe: to get all the files shared the requesting client;
23. getUserProfile: to get user profile;
24. void freeUserProfile: to free user profile;
25. void freeFileInfoArray: to free file info array;
26. void freePendingShareReqArray: to free pending share requets array;
27. doesShareReqExistForUid: to check for share request to a particular UID;
28. updateFileSize: to update file size;

6. Connectivity.c & Connectivity.h:

This function starts the server by creating a socket and binding the server on to the socket and starts listening on it. It even accepts a new incoming connection from a client and returns the socket fd for the new established connection.

Error Handling Module:

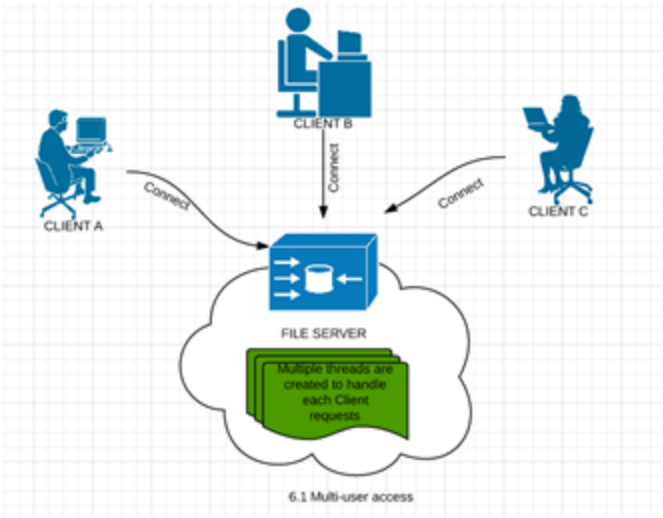
Error handling is used for showing error messages when an error occurs while executing commands. This helps users to understand the flow and can understand that where its going wrong and may be can try to rectify them.

1. E_SUCCESS: When function accepts false value then this message is displayed. It tells users that desired job has not been completed.
2. E_FILETOOBIG: When file is too big and it exceeds the size of buffer then this message is executed.
3. E_ALREADYLOGGEDIN: This error message generally comes when we trying to login even after we are logged in
4. E_FILEEXISTS: When user tries to create file with the same name of another that is already allocated in database then this message is displayed.
5. E_FILENOTFOUND: When user wants to access a file that does not exist in database then this error message is executed.

6. E_NOTLOGGEDIN: If user haven't logged in and still if he tries to execute command then this error message is displayed.
7. E_NOTAUSER: When user passes another user's ID that is not registered in database then this error message is displayed.
8. E_USEREXISTS: If user wants to registered with username that is already assigned to another user then this error message is displayed
9. E_INVALSHAREOPTION: If users enters some permission other than read only or write only then this error message is displayed
10. E_AUTHFAILED: When Authentication failures happens this message is displayed
11. E_ABORTCMD: When Operation is not performed properly i.e. if username entered is incorrect then abort cmd is executed to stop the operation.
12. E_BADFILESELECTED: When format of the selected file is not proper then this message is displayed.
13. E_INVALID_EMAILID: When the entered email id is not in proper format then this message is displayed.
14. E_INVALID_PHONE: When the entered phone no. is not in proper format then this message is displayed.
15. E_BADPERM: The permission given on file is not in proper format then error message occurs.
16. E_ACCESSDENIED: When user tried to access some document on which it has no permission then error message occured.
17. E_FILETOOBIGSHARED: The file selected to be uploaded is very large in size and is not getting saved in buffer then error message displays.
18. E_BADACCEPTANCE: When file selected is not in proper desired format then error message displayed.

Flow charts:

Client-server interactions can be broadly broken down into 7 major flows namely: User Registration, User Login, uploading a document, downloading a document, sharing a document, deleting the document, logout processes. To facilitate multi-user interaction to the server resources multithreading with lock mechanism is been incorporated. Upon request from client server starts a unique thread to handle all the requirements of the specific user as shown in the fig. 6.1



6.1 Multi-user access

Understanding this, let's understand the flow of messages between server and client for the above mentioned processes.

User Registration:-

1.User sends the connect message to Server.

2.Server creates a thread for the client thus connected and sends authentication/encryption parameters.

3.User sends the REGISTER command with all user details required for registration in encrypted format.

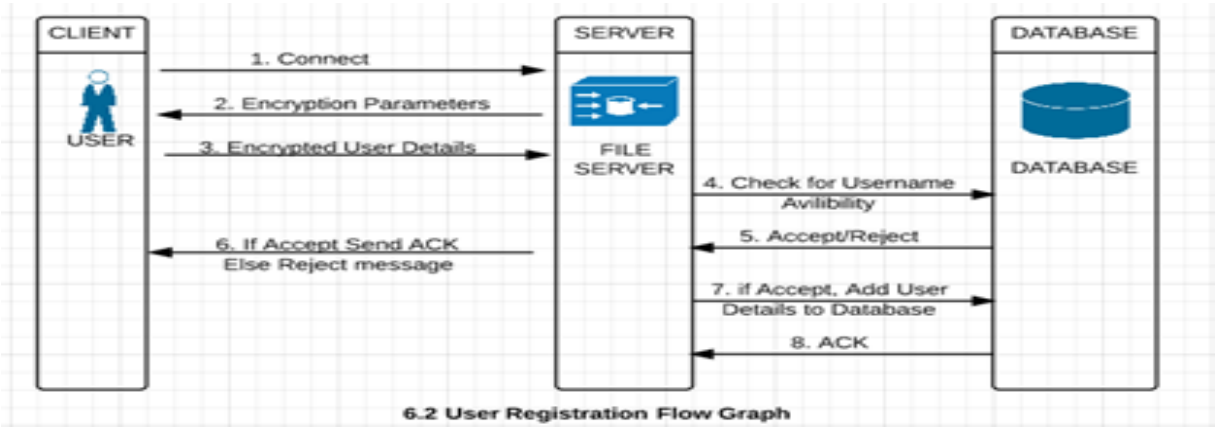
4. Server reads the username field and checks for its availability in the database.

5. If the username requested by the new user is available then, Database server would respond with positive accept message else a reject message would pop up.

6.Upon reception of accept message, server now sends registration successful acknowledgement message to client. If server have had received reject message then, it would have send the reject message with cause code username unavailable.

7. Meanwhile, server updates the given user details in database server.

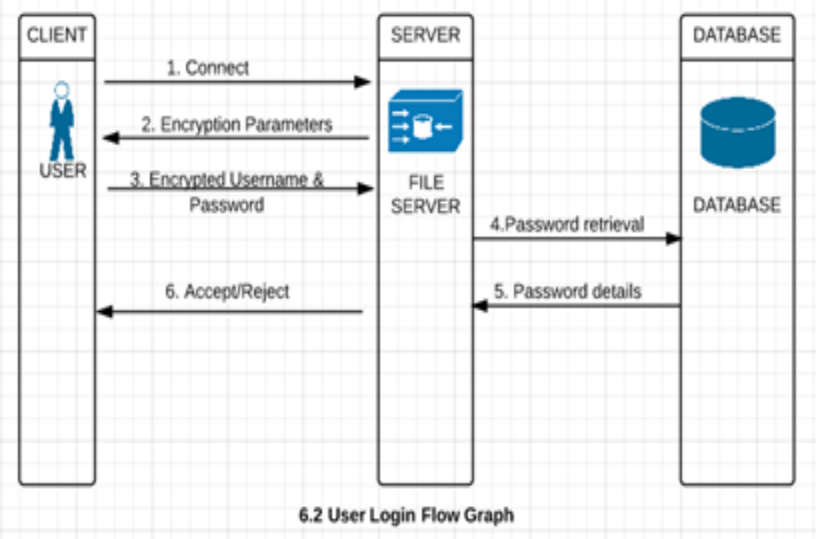
8.Upon successful updation of user details, database server sends ACK to the file server.



6.2 User Registration Flow Graph

User Login:-

1. User sends the connect message to Server.
2. Server creates a unique thread for the client and sends encryption parameters.
3. User uses these encryption parameters to encrypt username and password. It then sends LOGIN command with encrypted username and password to server for authentication.
4. Server upon reception of message from client, it decrypts and reads the username field. It now requests the database server for password details.
5. Database server replies with password details of the specific user.
6. Server now decrypts the password send by the client and matches it with the password it received from database server. If password matches, then server sends accept command to user with cause code 'login successful' else it sends reject command with cause code 'Authentication failed'.



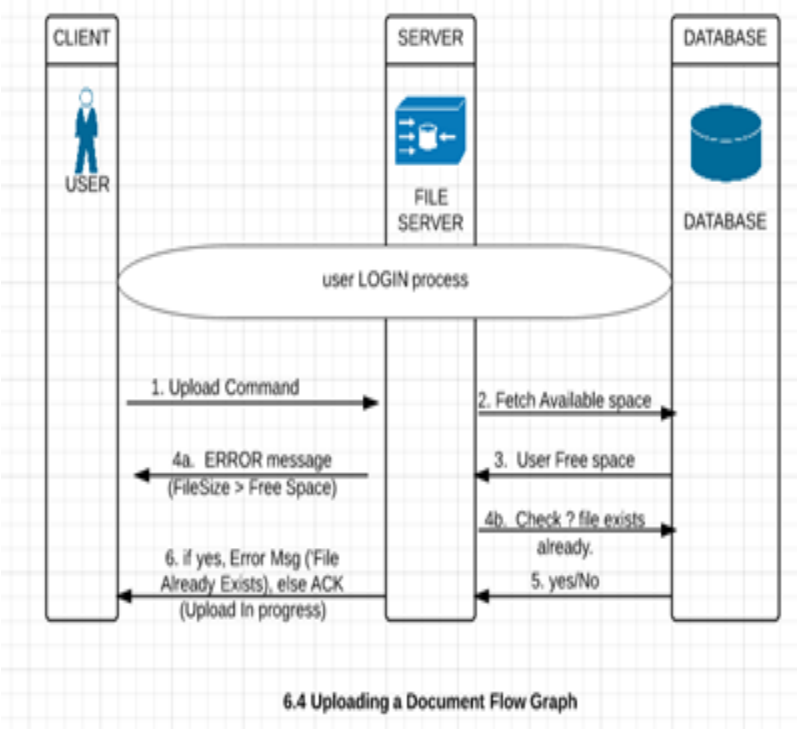
6.2 User Login Flow Graph

Uploading a document:-

In-order to upload any document the user has to first get logged into the server. So the login process is same as shown above and this context explains the uploading process.

1. After successful login, client issues UPLOAD command with file name and file path from where on local machine the file has to be uploaded as arguments in order to upload a document on server. Client encrypts UPLOAD command and sends to Server.
2. Server on receiving UPLOAD command from client, requests the Database server for available space to the specific user.
3. Database server checks the space that can be granted for the user and sends back to server.
4. Server now compares the size of file received and the free space available. If file_size> avail_space

the server sends error message to user with cause code ‘user limit exceeded’. Else server requests database server to check whether the file already exists or not?
5. Database server replies with either yes (file exists) or no (file doesn’t exist).
6. Upon receiving no from database server, the file server starts uploading the document else it sends error message to client with cause code ‘file already exists’.



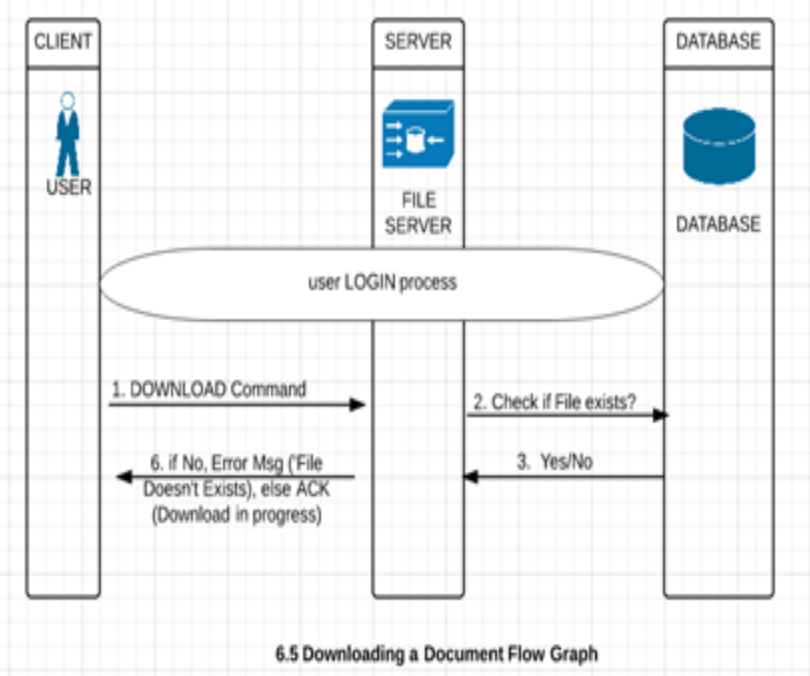
6.4 Uploading a Document Flow Graph

Downloading a document:-

In-order to download any document the user has to first get logged into the server. So the login process is same as shown above and this context explains the downloading process.

1. After successful login, client issues DOWNLOAD command with file name and file path specifying where to download the file as arguments in order to download a document from server. Client encrypts DOWNLOAD command and sends to Server.
2. Server on receiving DOWNLOAD command from client, requests the database server to check whether requested file name is present in the user file database or not.
3. Upon receiving ‘yes’ from Database server, file server sends the file to client else if it had received

'no' from database server then an error message with cause code 'file doesn't exist would be send to client'.



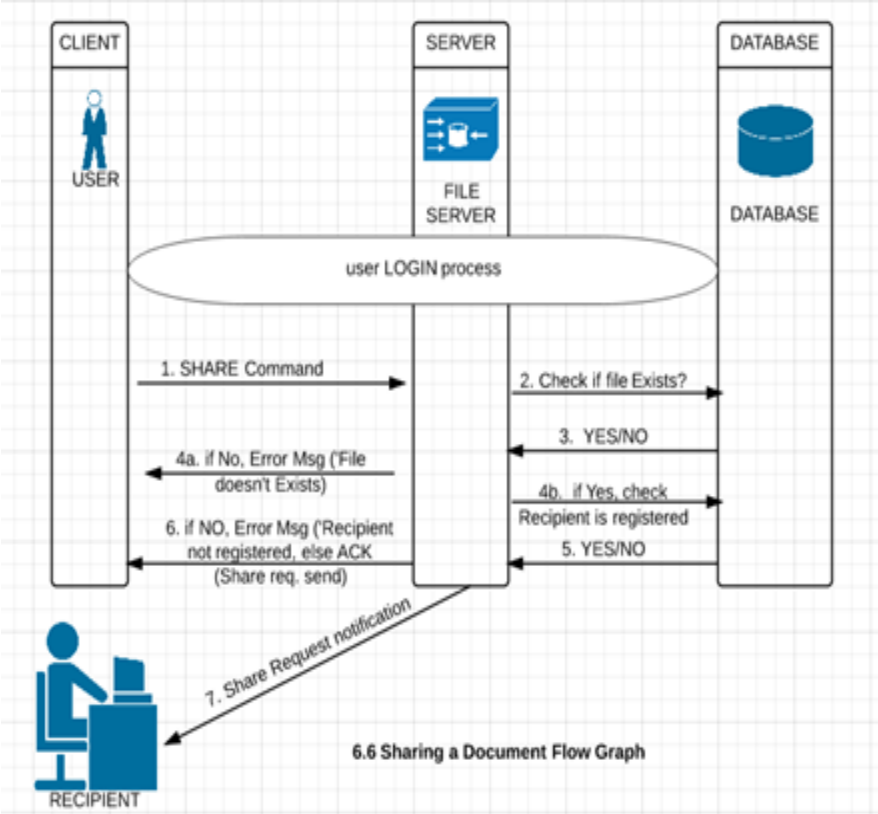
6.5 Downloading a Document Flow Graph

Sharing a document:-

In-order to share any document the user has to first get logged into the server. So the login process is same as shown above and this context explains the sharing process.

1. After successful login, client issues SHARE command with file name, file permissions & username of the other user with who file is been shared with. Client encrypts SHARE command and sends to Server.
2. Server on receiving SHARE command does multiple requests to the database server to authenticate the request send by user. Firstly, it checks with database server whether the file requested for sharing is in database or not?
3. Database server, receives the file name and checks for its presence in the user directory with Yes if present and No if absent.

4. Upon receiving 'yes' from database, it checks whether the recipient is registered with server or not? Upon receiving 'No' from database it sends Error message to client with cause code 'file doesn't exists'.
5. Database now checks its database for the recipient user details. If present it replies 'yes' else 'no'.
6. Server, if it receives 'yes' then it sends acknowledgement to sender and a sharing notification to recipient. Else if it receives no then it sends error message to sender with cause code 'Recipient doesn't exist'



6.6 Sharing a Document Flow Graph

VI. DATA ANALYSIS AND DISCUSSION

As the project is centered on developing a prototype model for file sharing system, we would be checking the integrity of model by forcing different simple and complex scenarios. As said above the functionality of server is implemented in C and is run on a unix/Linux machine, while client is made platform independent by deploying the solution in Java. The metric to measure integrity of the system performance is user accessibility, latency and ability to handle the multi-requests during sharing scenario. Following are the scenarios that are currently designed to test the prototype model-

- User registration for membership.
- Validating the registered user while accessing the server.
- Uploading a document of a size M which is less than the space available in the server
- Uploading a document of a size M that violates the space available to the user.
- Downloading the Document from the server.

- Sharing file with other peers who are registered in the server database and file sharing is granted by the peer.
- Accessing a file on the database without sharing permission granted by the user.
- Muti-request handling, scenario where more than 1 active user is present.
- Muti-request handling, scenario where more than 1 active user is accessing the same shared file.

some of the above mentioned scenarios are abnormal and conflicting cases that can arise while simulation. Lets look into each of these one by one.

- First lets see what happens when user is trying to access the space beyond the allowed limits. by default every registered user is given 1GB space. If he/she tries to access more than the allowed room on server then the server sends out an exception asking user to check his/her usage and reupload the file accordingly.
- Another abnormal user behavior is when he/she is trying to access the file which is not been permitted to him/her. Server in this ambiguous situation sends a warning requesting client to check the sharing permissions from peer and also notify the peer about anonymous file access. upon grant of permission from the peer the requested client will either get access to file or denial.
- Another bottleneck to be considered while designing the model is how it behaves when more than one active user is trying to access the same shared file. To avoid such conflicts we are implementing multithreading concept with synchronization based on locks. During such scenarios, one of the requested client will acquire lock and holds on to the shared resource. The client 2 who is trying to access the same file will get a read-only copy of the file and would get write access when client 1 releases the lock

OUTPUT GENERATION:

1. HELP

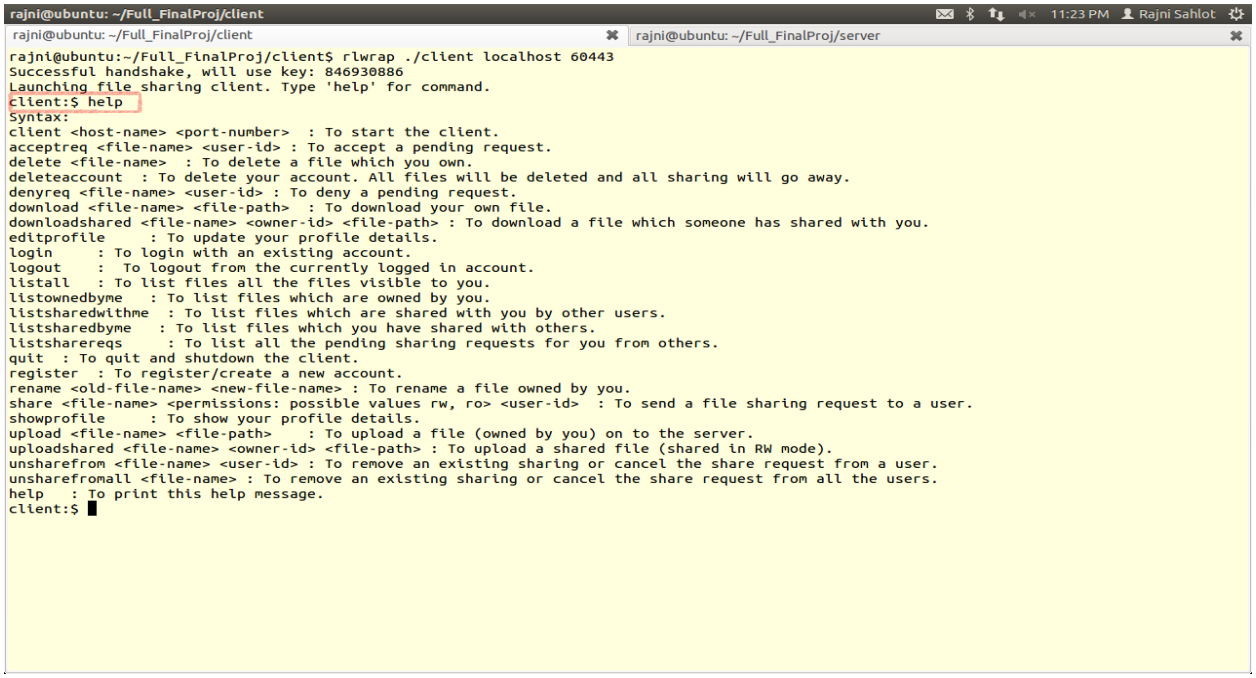


Fig 7: Help Command

2. REGISTER

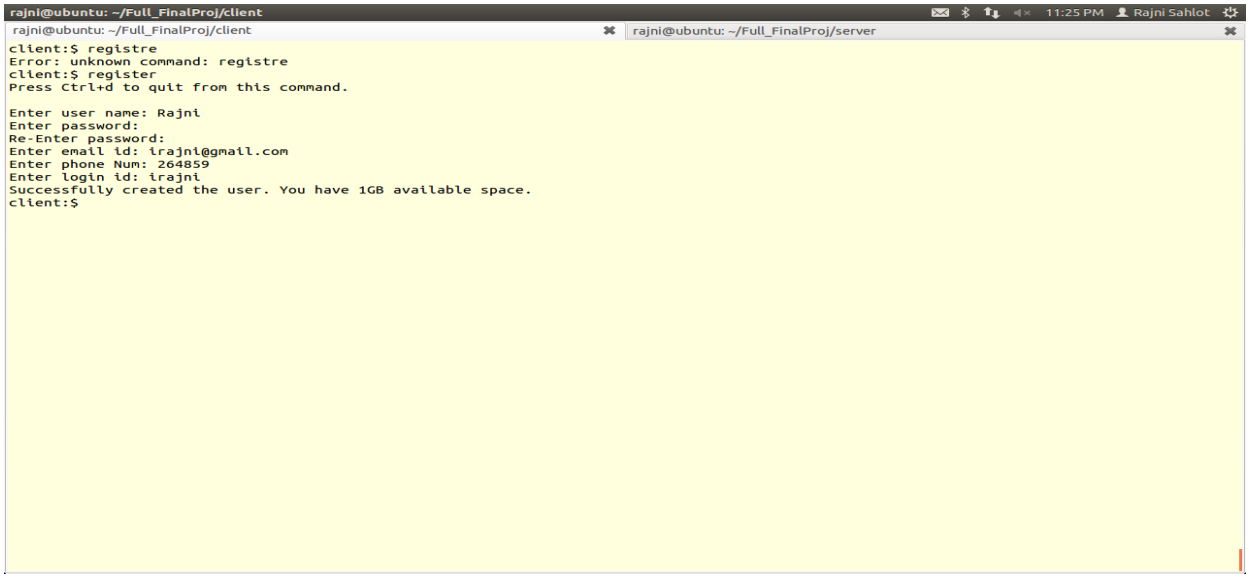


Fig 8: Register Command

3.LOGIN

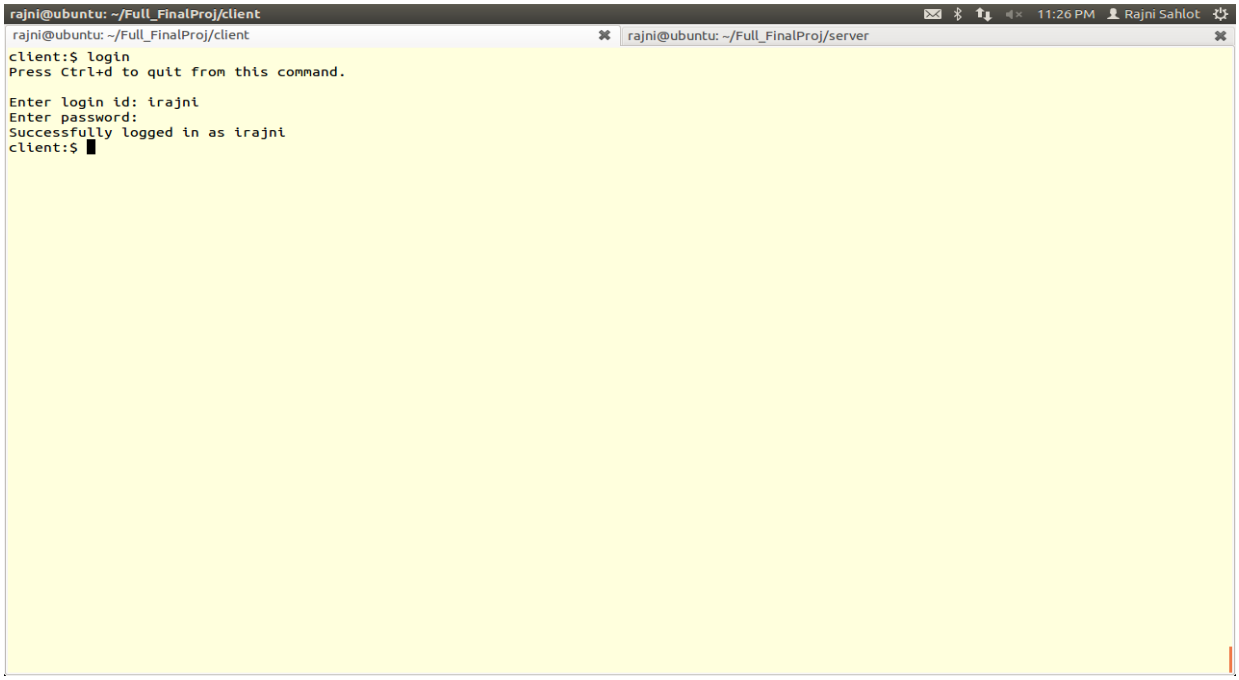


Fig 9: Login Command

4. UPLOAD AND LISTALL

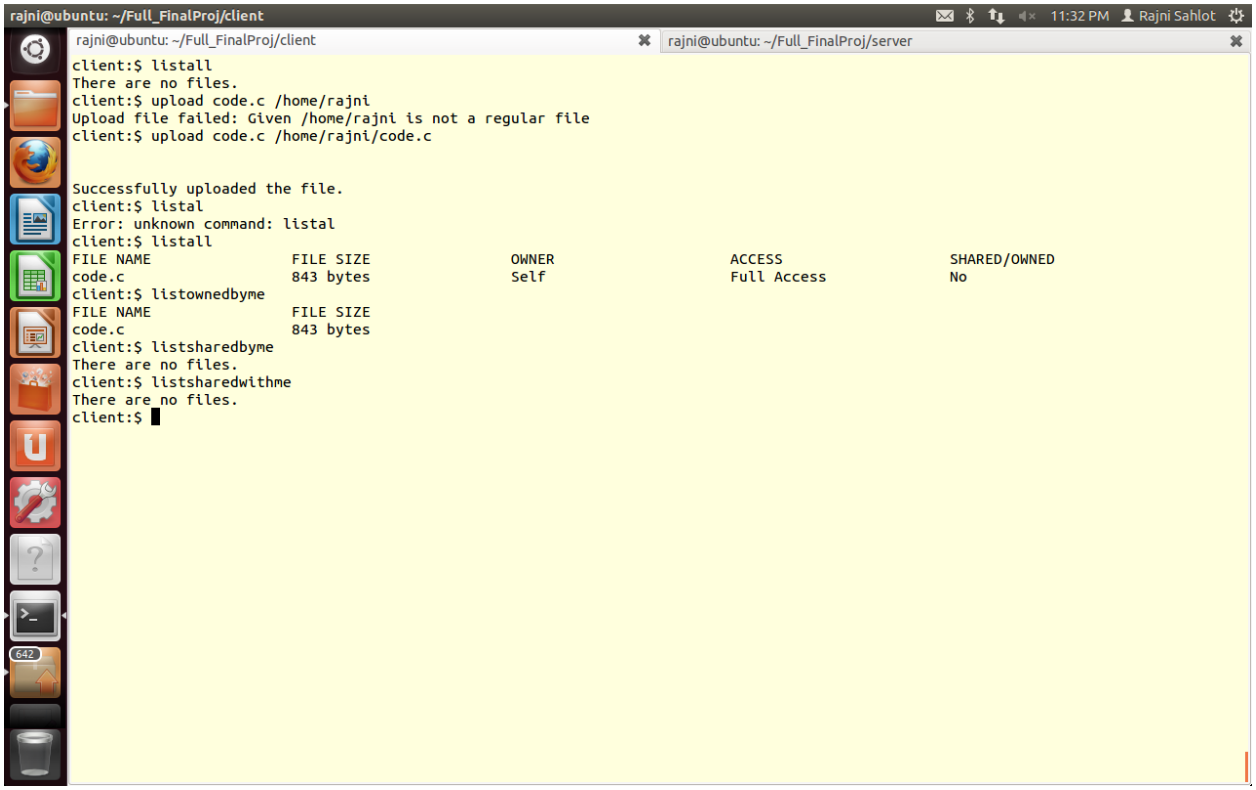


Fig 10: Upload and ListAll Command

5. DOWNLOAD

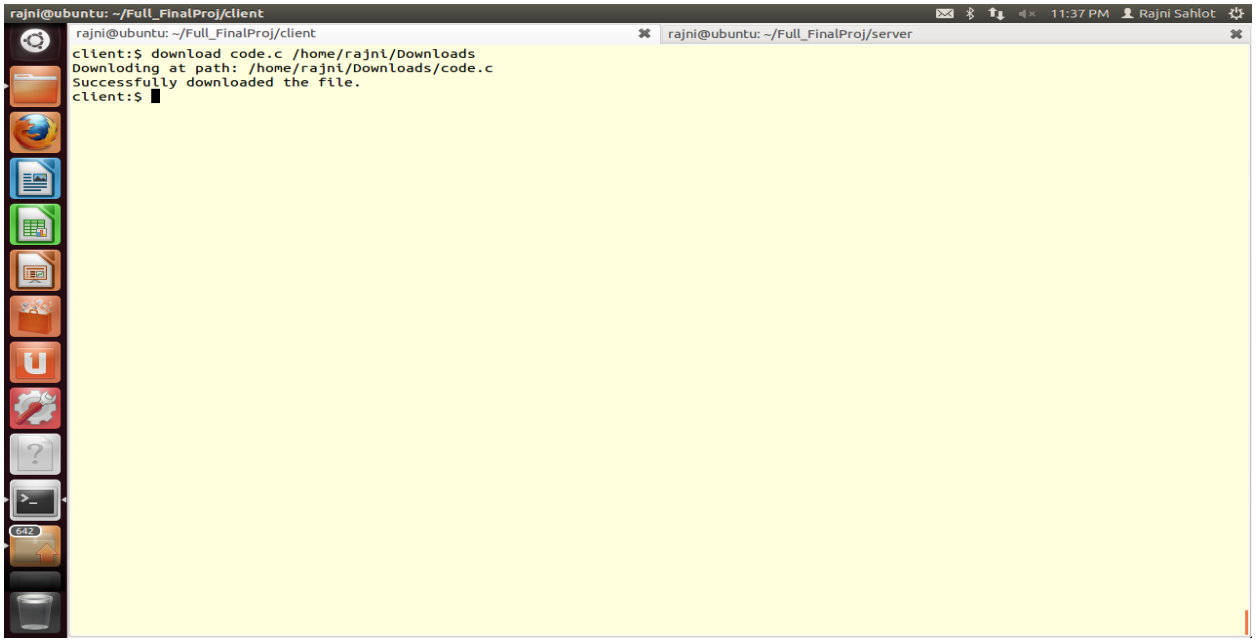


Fig 11: Download Command

6. SHARE

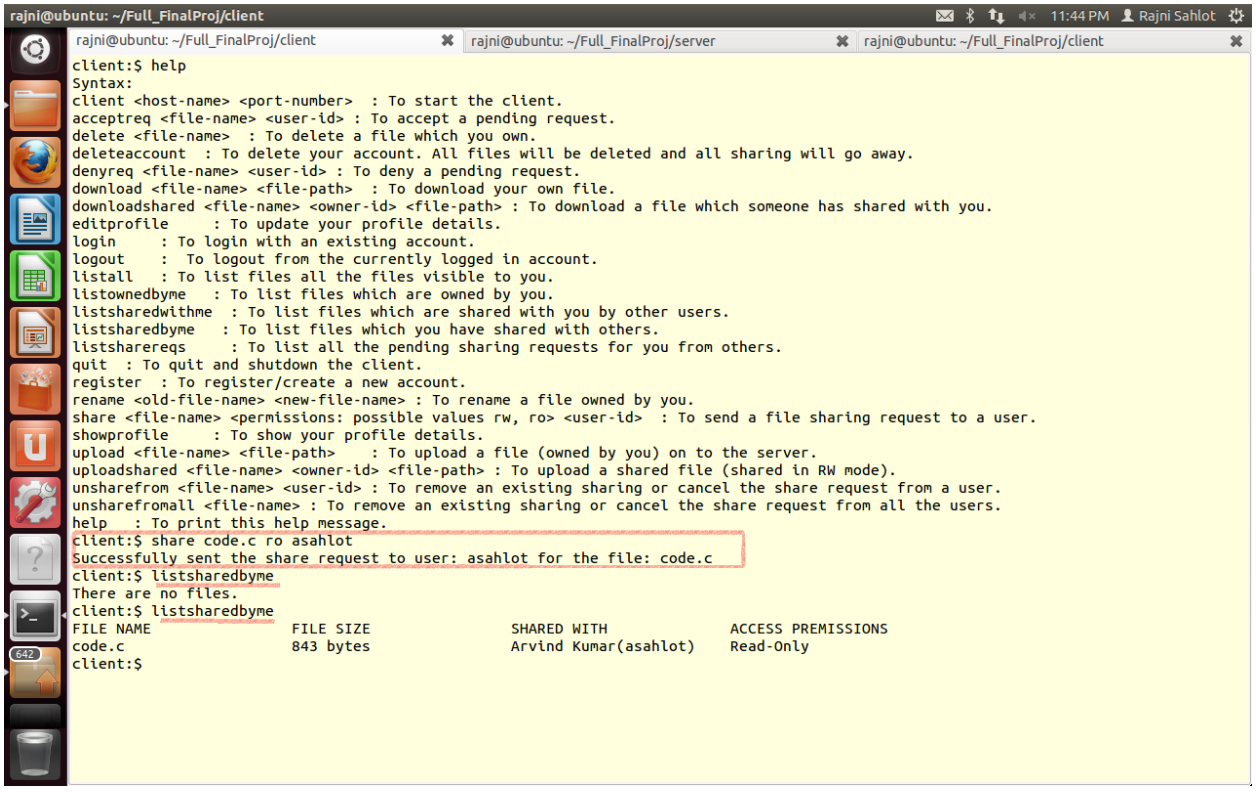


Fig 12: ShareCommand

7. RENAME

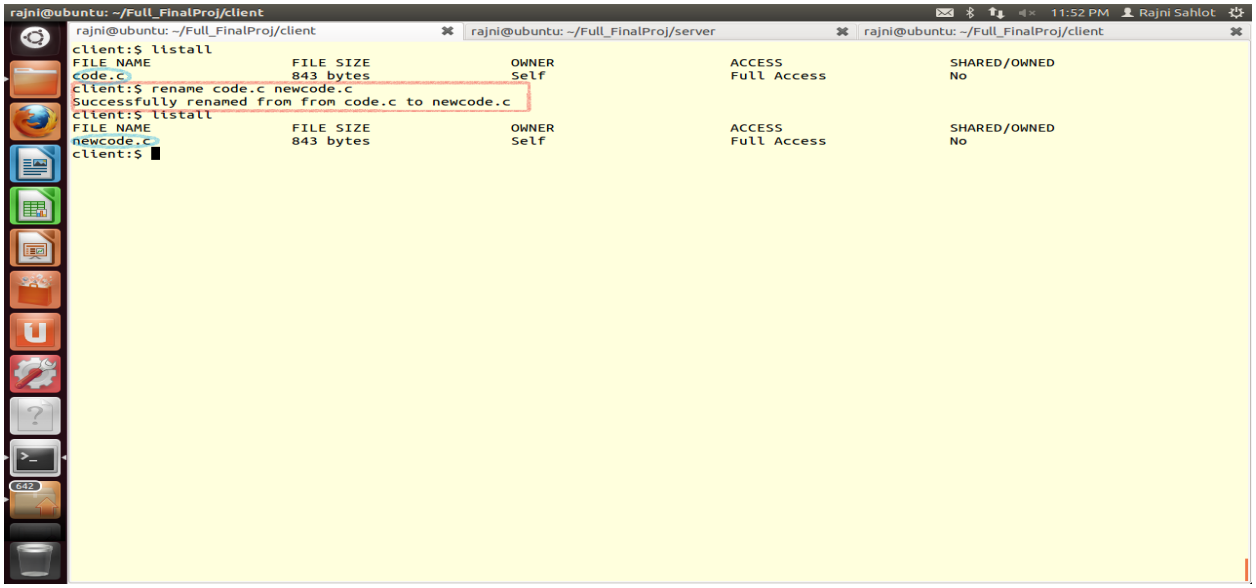


Fig 13: Rename Command

8. SHOW and EDIT PROFILE

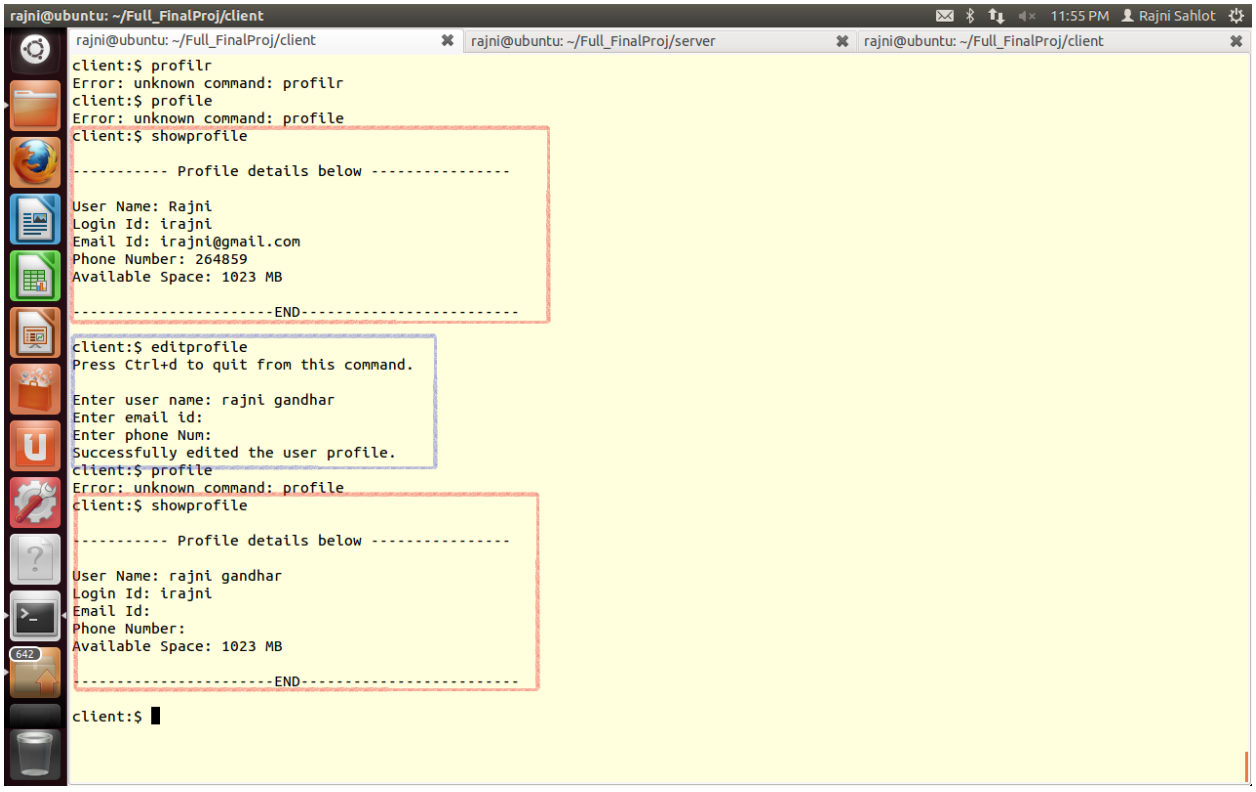


Fig 14: Show and Edit Profile Command

9. DELETE

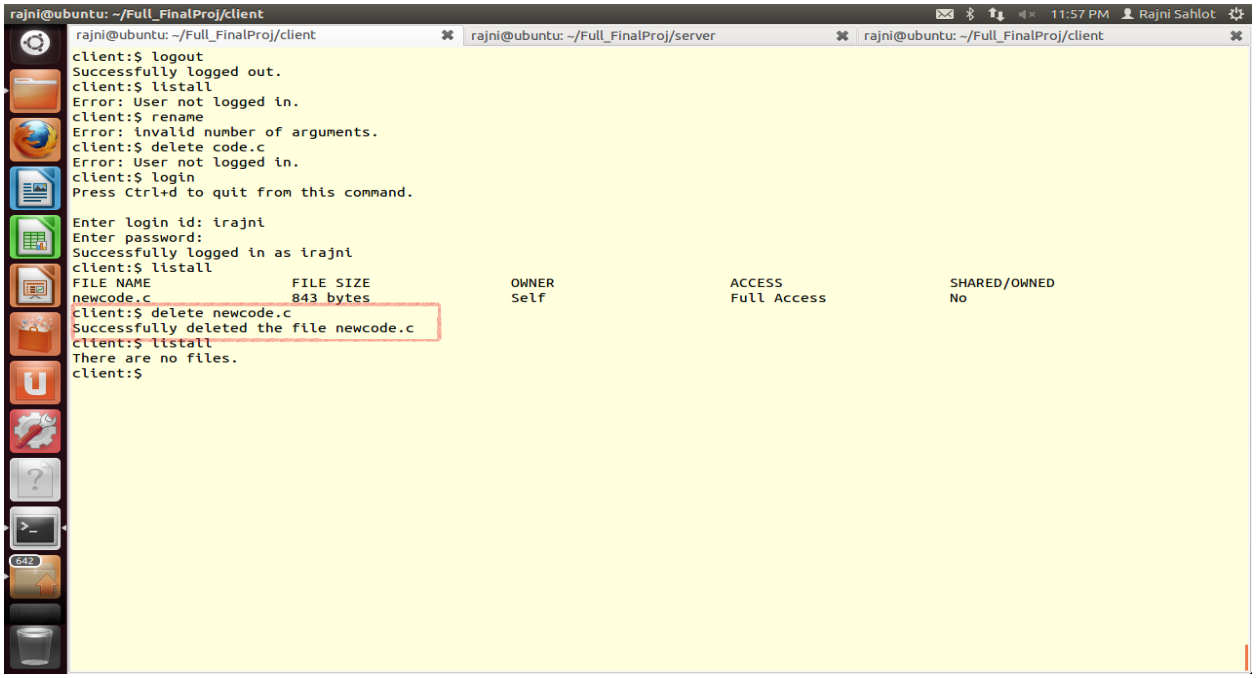


Fig 15: Delete Command

10. DELETE ACCOUNT

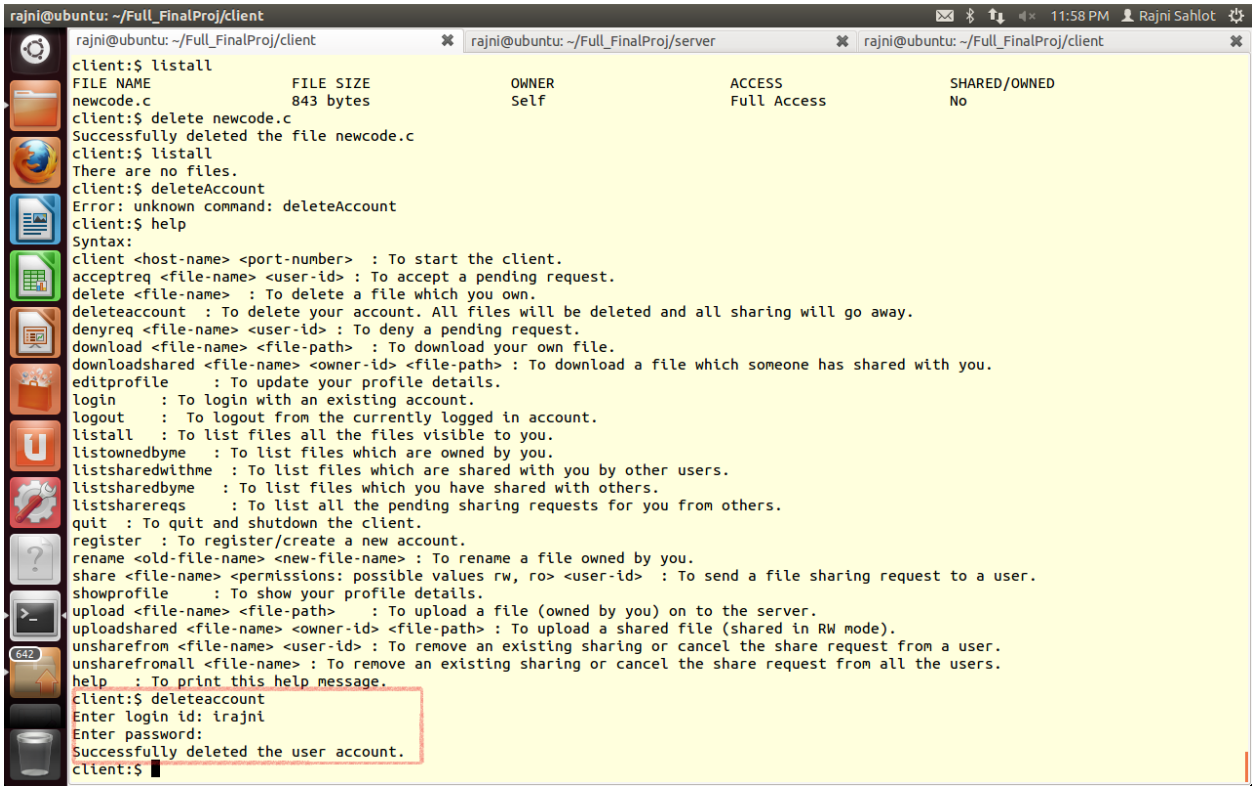


Fig 16: Delete Account Command

VII. Conclusions and recommendations:

Overall this system makes users close to their data i.e. users can see, modify, and share their data anytime from anywhere, they don't have to carry their computer always with them to work on their data.

So here we are using a client-server server architecture which makes this thing possible. A user who wants to use the services of this system must have a client of the system and that client sends request to an always running server then server serves the requesting service to the user.

There are main 6 commands in this file-sharing system as follows:

1. Users can register to get the benefits of file sharing services offered by file sharing system.
2. Server will always does user authentication before allowing any user to use the services of the system.
3. After successful authentication, users can upload files on the file sharing system (server).
4. User can download the file from the file sharing system(server).
5. Users can share their file with other registered users.
6. Users can delete the file from the file sharing system.

Future scope:

This project is a small model of the file-sharing area applications. But there are many future scopes for this project some of them are as follows:

- There can be a very sophisticated client with very intuitive graphical user interface. If time permits we will try to do this in this project itself.
- For removing the need of desktop based client we can think about a web-based client. That will provide more flexibility to users to use the file-sharing system.
- we can make our server more robust by doing some kind of replication of the server's data and by running more than one server. Since in this project we are using only one centralized server if something wrong happens users will lose their data and that will be a very big loss.