TCP Server-Client Based Distributive File System

A PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE
B.Tech Degree in Computer Science and Technology

Submitted By:
Shubhamoy Nandi (Roll No. 510518015),
Rajeev Ranjan Sinha (Roll No. 510517035),
Narapureddy Srikanth (Roll No. 510518066),
Ashutosh Bharati (Roll No. 510518087)

Under the supervision of

Prof. Manas Hira

November 29, 2019

TCP Server-Client Based Distributive File System

Shubhamoy Nandi (Roll No. 510518015),
Rajeev Ranjan Sinha (Roll No. 510517035),
Narapureddy Srikanth (Roll No. 510518066),
Ashutosh Bharati (Roll No. 510518087)

Under the supervision of

Prof. Manas Hira

Department of Computer Science and Technology
IIEST, Shibpur
India-711103

Acknowledgement

We would like to express our deep gratitude to Professor Manas Hira, Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur, our project supervisor, for his patient guidance, enthusiastic encouragement and useful critiques for this project.

We would like to thank Professor Sekhar Mandal, the Head of Department, for his support.

We would also like to thank all the faculty members and staff of the Department, as well as our fellow classmates for their help and support.

(Shubhamoy Nandi - 510518015)

(Rajeev Ranjan Sinha - 510517035)

(Narapureddy Srikanth - 510518066)

(Ashutosh Bharati - 510518087)

Certificate

This is to certify that the project titled "**TCP Server-Client Based Distributive File System**" has been carried out by **Shubhamoy Nandi (Roll No. 510518015), Rajeev Ranjan Sinha (Roll No. 510517035), Narapureddy Srikanth (Roll No. 510518066) and Ashutosh Bharati (Roll No. 510518087)** under my supervision and guidance in their third and fourth semesters in partial fulfillment of the requirements for the B.Tech programmme in the Department of Computer Science and Technology, IIEST Shibpur. This work has not been reported anywhere for any other purpose.

(Manas Hira)

Abstract

In this report we will demonstrate distributive file system using TCP Server-Client Programming i.e. Socket Programming and Database Management.

Contents

1 Introduction

Over the last few years, there has been a drastic change in information technology. This includes the various ways in which files can be shared and stored. Android OS is a relatively new mobile OS which has been steadily taking over more and more market share. Easy to use, easy to develop for, and open-source, it has picked up a following of developers who want to create content for the masses. Cloud computing is publicized as the next major step for all forms of typical information technology use. From businesses, to non-profit organisations, to single users, there seems to be various applications which can use cloud computing to offer better, faster, and smarter computing. This paper aims to combine the two, building a cloud based application for Android, offering users the power of cloud computing in the palm of their hand.

File sharing is the practice of distributing or providing access to digitally stored information, such as computer programs, multimedia (audio, images and video), documents, or electronic books. It may be implemented through a variety of ways. Android is a relatively new mobile operating system developed by Google and the Open Handset Alliance.

Client-server program nowadays is no longer an unfamiliar concept in distributed computing and in computer networks. In fact, the internet up to this date has many client server applications readily available anytime. In client-server model, each computer terminal or process on the network could be a client or a server.

A “client” is a program or a computer terminal that allows users to access and view its interfaces. Every client communicates and connects with each through server. In an overall concept, client serves as the one who initiates the communication wherein the server is the one who waits passively to respond to the client’s request. On the other hand, a socket is an abstraction which allows program to send and receive data. The most essential characteristic of socket programming that drive programmers to design a program is due to its transparency. It simply means that, whether a socket program is designed and written in Java language, it still has the ability to communicate with other socket program which is designed and written in other programming languages such as C or C++.

1.1 Motivation

More available space for users to access programs
Easier to back up files
Reduce number of programs on any given network
Simply administration
Helps maintain consistency of shared data files

1.2 Proposed Project

The era of mobile technology opens the windows to the android app. Mobile has become the integral part of our daily routine, but we don't have much space in our mobiles so there are some problems in keeping some important but big files in our mobile. These files can be useful for many people and they all have to keep these files in their mobiles. To overcome this, we are making an android app for accessing any data what others shared in a closed room without using our mobile space.

A virtual space will be created using this app where we can share any file in a closed room. Any member of that room can access those shared files without using their mobile space. This virtual device will use mobile space of all users in that closed room to create a big space.

Due to lack of time we have implemented this on Desktop using TCP Server-Client Programming in C.

1.3 Outline of Project

Section 2: provides the Tools and Technology used in this Project.

Section 3: discusses all the function used in Socket Programming and SQLite.

Section 4: covers the interfaces of Project.

Section 5: outlines the work completed till now and limitations of the Project.

Section 6: discusses about implementation of Android Based File Distributive System.

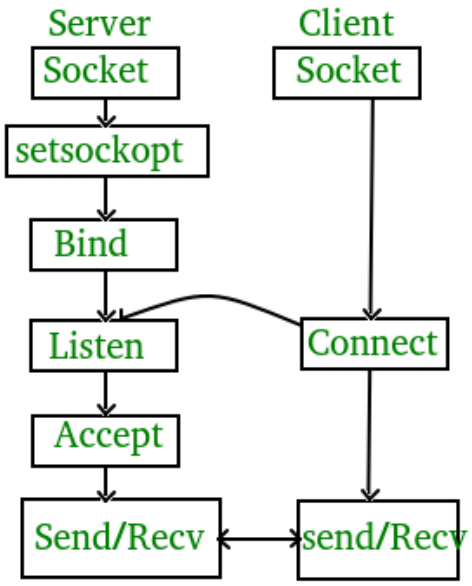
2 Tools and Technology

In this section we will be talking about implementation of this project with the help of following tools and technology:-

2.1 Server-Client Model (Socket Programming)

Socket programs are used to communicate between various processes usually running on different systems. It is mostly used to create a client-server environment. This post provides the various functions used to create the server and client program and an example program. In the example, the client program sends a file name to the server and the server sends the contents of the file back to the client.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



State diagram for server and client model

2.2 Database Management (Sqlite3)

SQLite is an in-process library that implements a self contained, server less, zero configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 600KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function in lining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of code changes.

3 Implementations

In this section we will discuss about different functions used in C for implementation of Socket Programming and Database Management.

3.1 Functions used in Socket Programming

3.1.1 Socket Creation:

```
int sockfd = socket(domain, type, protocol);
```

sockfd: socket descriptor, an integer (like a file-handle)
domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
type: communication type
SOCK_STREAM: TCP(reliable, connection oriented)
SOCK_DGRAM: UDP(unreliable, connectionless)
protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

3.1.2 Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_taddrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the local host, hence we use INADDR_ANY to specify the IP address.

3.1.3 Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

3.1.4 Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

3.1.5 Connect:

```
int connect(int sockfd, const struct sockaddr *addr,
socklen_taddrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

3.2 Sqlite3 Queries and Functions

3.2.1 CREATE Database:

```
$sqlite3 DatabaseName.db
```

The above command will create a file **DatabaseName.db** in the current directory. This file will be used as database by SQLite engine. If you have noticed while creating database, sqlite3 command will provide a **sqlite>** prompt after creating a database file successfully.

3.2.2 CREATE Table:

```
CREATE TABLE database_name.table_name(
column1 datatype PRIMARY KEY,
column2 datatype,
column3 datatype,
.....
columnN datatype
);
```

CREATE Table is the keyword telling the database system to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement. Optionally, you can specify *database_name* along with *table_name*.

3.2.3 INSERT Query:

```
INSERT INTO TABLE_NAME [(column1, column2,
column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, ... columnN are the names of the columns in the table into which you want to insert data.

3.2.4 SELECT Query:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2 ... are the fields of a table, whose values you want to fetch.

```
SELECT * FROM table_name;
```

We use * for fetching all the field from a table.

3.2.5 Open Function:

```
sqlite3_open(const char *filename, sqlite3 **ppDb)
```

This routine opens a connection to an SQLite database file and returns a database connection object to be used by other SQLite routines.

If the *filename* argument is NULL or ':memory:', sqlite3_open() will create an in-memory database in RAM that lasts only for the duration of the session.

If the filename is not NULL, sqlite3_open() attempts to open the database file by using its value. If no file by that name exists, sqlite3_open() will open a new database file by that name.

3.2.6 Execute Function:

```
sqlite3_exec(sqlite3*, const char *sql, sqlite_callback, void *data, char **errmsg)
```

This routine provides a quick, easy way to execute SQL commands provided by sql argument which can consist of more than one SQL command.

Here, the first argument *sqlite3* is an open database object, *sqlite_callback* is a call back for which *data* is the 1st argument and errmsg will be returned to capture any error raised by the routine.

SQLite3_exec() routine parses and executes every command given in the **sql** argument until it reaches the end of the string or encounters an error.

3.2.7 Close Function:

```
sqlite3_close(sqlite3*)
```

This routine closes a database connection previously opened by a call to sqlite3_open(). All prepared statements associated with the connection should be finalized prior to closing the connection.

If any queries remain that have not been finalized, sqlite3_close() will return SQLITE_BUSY with the error message Unable to close due to unfinalized statements.

3.3 File Handling

In this section we will discuss about different functions used in File Handling.

3.3.1 Open File:

```
FILE *fopen( const char * filename, const char * mode );
```

The fopen() function accepts two parameters:

- The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored.
- The mode in which the file is to be opened. It is a string.

3.3.2 Close File:

```
int fclose( FILE *fp );
```

This function closes the file pointed by file pointer fp.

3.3.3 Read Function:

```
read (int fd, void* buf, size_t cnt);
```

This function reads cnt bytes of input into the memory area indicated by buf.

3.3.4 Write Function:

```
write (int fd, void* buf, size_t cnt);
```

this function writes cnt bytes of input into the memory area indicated by bu

4 Interfaces

```
srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ gcc finalclient.c
srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ gcc finalclient.c -l sqlite3 -o client
srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ ./client 127.0.0.1 8080
Socket successfully created..
connected to the server..
Press 1 for Login
New User?, Press 2 for Sign Up
Press 3 for Exit
```

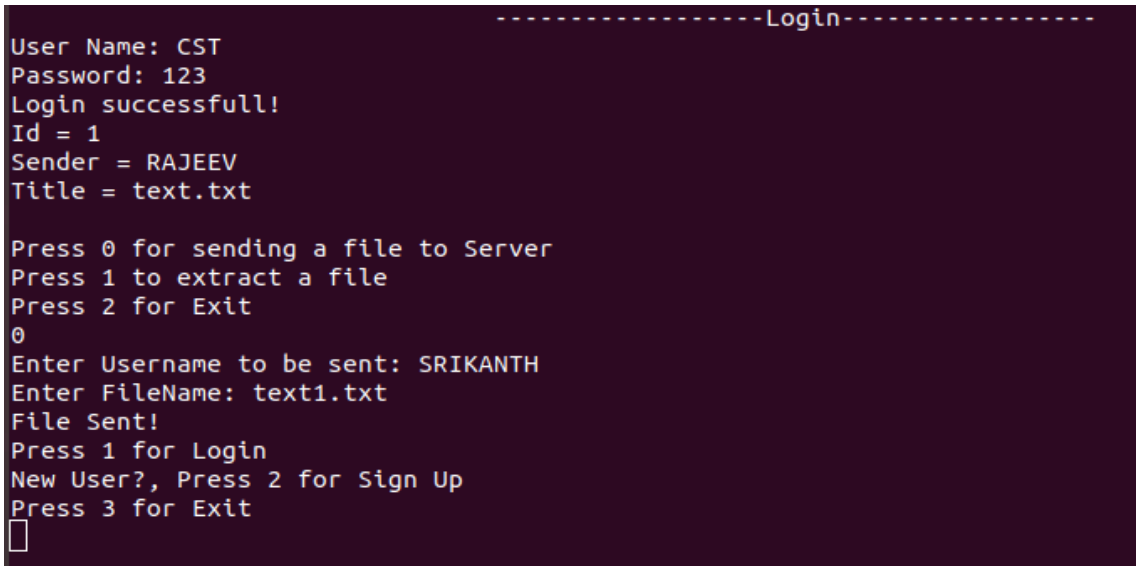
```
srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ gcc finalclient.c -srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ gcc finalclient.c -l sqlite3 -o client
srikanth@srikanth-HP-Pavilion-Laptop-15-cc1xx:~/miniproject$ ./client 127.0.0.1 8080
Socket successfully created..
connected to the server..
Press 1 for Login
New User?, Press 2 for Sign Up
Press 3 for Exit
2
                        ----------------Sign Up---------------
User Name: CST
Password: 123
Confirm Password: 123
Successfully Signed Up
Press 1 for Login
New User?, Press 2 for Sign Up
Press 3 for Exit
```

```
                                  ------------------Login-----------------
User Name: CST
Password: 123
Login successfull!
Id = 1
Sender = RAJEEV
Title = text.txt

Press 0 for sending a file to Server
Press 1 to extract a file
Press 2 for Exit
```

```
                                  ------------------Login-----------------
User Name: CST
Password: 123
Login successfull!
Id = 1
Sender = RAJEEV
Title = text.txt

Press 0 for sending a file to Server
Press 1 to extract a file
Press 2 for Exit
1
Enter FileName  you want to Download: text.txt
File Downloaded!
Press 1 for Login
New User?, Press 2 for Sign Up
Press 3 for Exit
```


-------------------Login-----------------
User Name: CST
Password: 123
Login successfull!
Id = 1
Sender = RAJEEV
Title = text.txt
Press 0 for sending a file to Server
Press 1 to extract a file
Press 2 for Exit
0
Enter Username to be sent: SRIKANTH
Enter FileName: text1.txt
File Sent!
Press 1 for Login
New User?, Press 2 for Sign Up
Press 3 for Exit

5 Work Done

In this project, an approach for developing TCP Server-Client Based Distributive File System was implemented.

Implementation so far we could make in sharing text file through Server-Client Program i.e. Socket Program in C.

6 Future Plan

Our proposed project was "**Android Based Distributive File System**" that could not be implemented in time. So far we have implemented TCP Server-Client Based Distributive File system in C.

In future we will switch the TCP Server-Client Based Distributive File System which is for Desktops to Android Based Distributive File System.

References

1 Wikipedia
https://www.wikipedia.org/.

2 Socket Programming - GeeksforGeeks
https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

3 Sqlite3 – Tutorials Point
https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm

Appendix

Source Code:
https://drive.google.com/open?id=15lX3tc7GWaQG6qqwpnUdUnE1c7WXXCej