Daniel Walton
ECE 4220
12/16/10

File Transfer Program

Abstract:

This project is a file transfer program that models the File Transfer Protocol. This program uses sockets to transfer any type of file to a server from multiple clients. It allows clients to use four different commands to manipulate the files. Also, the file transfer program allows for simultaneous operations on the server file system. The concepts from class that are used are sockets, file input/output, and forks.

Introduction:

The purpose of this project is to create a program that uses a protocol similar to FTP (File Transfer Protocol). In this particular protocol, I will provide four different commands that the clients can use. One command is UPLOAD which allows the client to choose a file from any directory on their computer and send it to the designated directory on the server. Another command is DOWNLOAD. This command allows the client to select a file to download from the server computer, transfers the file to the client computer and saves the file under any filename in any directory. The LIST command is used to display all the files that are available to the clients to download. The DELETE command is used to remove any particular file from the server side computer. One objective of this program is to allow multiple client access to the server which means that more than one person can login to the server and access files. In

conjunction with multiple client access, I provided simultaneous access to the operations pending they weren't operations on the same file.

Problem Statement and Technical Background:

The problem I am solving is providing a centralized way of distributing files to multiple users in a network setting. To begin with, the program must use socket communication in order to communicate with other computers on the network. Two of the possible protocols to choose from for socket communication are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). For the purposes of this program, UDP is not a logical choice because it is not connection based meaning that packets could arrive in a different order than they were transmitted which would contaminate the integrity of the files that are being transferred. Therefore, TCP is used to provide a direct link between the server and each of the clients. To allow multiple client access to the functions of the server, forks are used. The purpose of a fork is to create child processes from a parent process so that the functions can be duplicated and run simultaneously in separate threads. As new clients are accepted by the server, the parent thread forks and creates a unique identifier for the client. Another important component in the execution of my program is the use of system calls in Linux. To be able to list files in a particular directory, the system call "ls –la" is used. The output of the command is then piped into a file so that it may be read later and transferred to client computers. To allow the simultaneous operation of clients on the server, lock files are used to prevent clients to manipulate the same exact file at the same time. It will allow the simultaneous operations on two separate files however.

Proposed Approach:

The block diagram for the file transfer program is shown in **Figure 1**. The program consists of two different types of files: server and client. There is one server but there can be any number of clients. The two types of commands that are sent between the server and clients are commands and the files themselves.

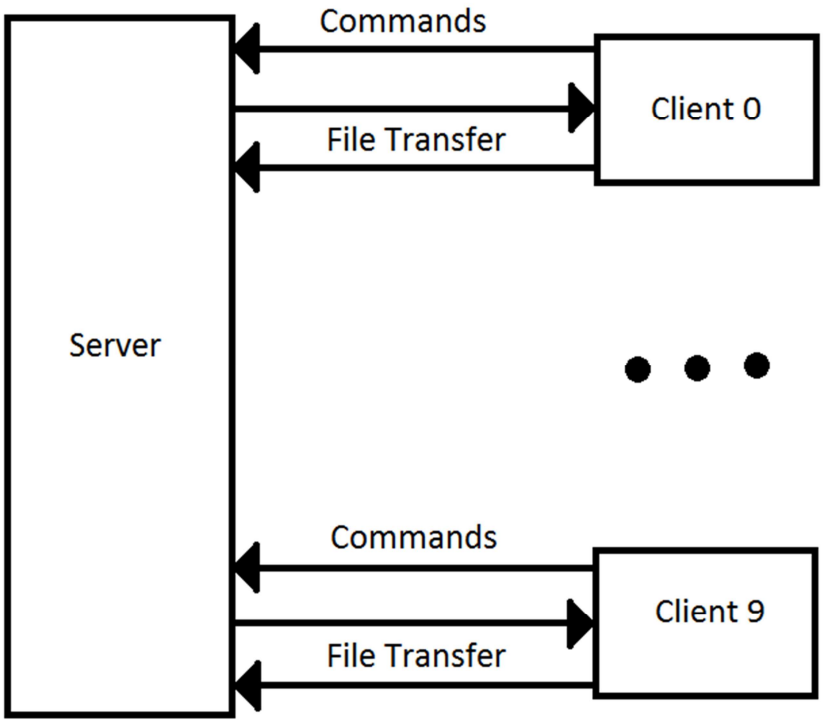

Figure 1 - Block Diagram

The client begins by setting up a TCP socket .It is set up to the ip address of the server and the port number 1948. Once the socket is set up, an integer is received which determines if the client connection with the server was successful. There was an arbitrary limit of ten set on the number of clients that could be connected. Then the client enters into a while one loop that asks the user to enter the command that is used for operations. If the command given is equal to the string “UPLOAD”, the command name is first sent through the socket connection.

Following that, the client is prompted to enter the location of the file that they wish to send. Then the number of slashes is counted and is used to tokenize the string and pull of just the file name without the directory. This is the name it will be saved as on the server. Then the size of the file is determined by opening the file, using the fseek function from zero until the end of the file, and then returning the value from the ftell function. The size of the file is then sent through the socket. Then the name of the file that is going to be transmitted is sent. Following that, the file is opened using the fopen function for read only functionality. It then reads in each byte to a byte array within a while loop. Then the byte array is sent through the socket and the file is closed. If the command given is equal to the string “DOWNLOAD”, the command name is first sent through the socket connection. Following that, the client is prompted to enter the file name that is to be received as well as the location to save it. Then the file name to be downloaded is sent through the socket. Then the size of the file to be downloaded is received. Then a file is opened or created for writing using the fopen function. Then a while loop is entered that reads in 4096 bytes of data at a time and writes them to the file using the fwrite command. Following that the file is closed using the fclose function. If the command given is equal to the string “LIST”, the command name is first sent through the socket connection. First the number of lines to be listed is received. Then a loop receives a line at a time and prints it. If the command given is equal to the string “DELETE”, the command name is first sent through the socket connection. Following that, the client is prompted for the file name to delete off the server computer. Once the file name is read in from the client it is sent through the socket. If the command given is equal to the string “EXIT”, the while one loop is broken and the socket is closed for the client.

The server begins by setting the maximum number of clients as well as setting all their statuses to an integer representing an offline state. Then a TCP socket is set up to be a server on port 1948. Then it waits for connections using the listen function. When a client is accepted, it is assigned an id number and is checked to see if it is greater than the maximum number of clients. Then a child process is forked off of the parent process to allow for simultaneous client access and a message indicating success is sent to the client. Then, like the client, the server sits in a while one loop to accept commands. If the command given is equal to the string "UPLOAD", the server reads in the file name and size. Then the server checks to see if any other clients are using a particular file by attempting to create a lock file. If the lock file creation is a success, then no other client is using the file so the client can continue the upload process. Then the file is received 4096 bytes at a time and is written to the file using the fwrite function. Then the file is closed and the lock file is deleted. If the command given is equal to the string "DOWNLOAD", then the server checks to see if any other clients are using the file that is to be downloaded by attempting to create a lock file. Once the lock file is created, the file size is sent through the socket. Then the file is opened and read in a byte at a time and saved to a byte array and is sent through the socket. Following that, the lock file is deleted. If the command given is equal to the string "LIST", a system call of "ls –la /home/ddwxtd/ServerFiles > client_NUM" is executed. This call uses the list shell command and pipes the results into a file so that it may be sent through the socket for use on the client. The file is then opened and sent a line at a time to the client that requested it. If the command given is equal to the string "DELETE", the server receives the file name to be deleted. It then checks to see if any other clients are using the file that is to be deleted by attempting to create

a lock file. Then the system call “rm –rf” is used for the particular file to be deleted. Then the lock file is deleted. If the command given is equal to the string “EXIT”, the socket is closed for that client id and the forked process is exited.

Results:

The input as well as output for both the server and client is shown in **Figure 2** and **Figure 3**. To test my program, I used a JPG picture titled “Mizzou1”.

```
ts7250-02:/home/ddwxtd/ProjectServer/Debug# ./ProjectServer
Client 0 connected
Client 0 Listing...
Client 0 Uploading...
Client 0 Uploaded /home/ddwxtd/ServerFiles/Mizzou1.jpg
Client 0 Listing...
Client 0 Downloading...
Client 0 Downloaded /home/ddwxtd/ServerFiles/Mizzou1.jpg
Client 0 Deleting...
Client 0 Deleted /home/ddwxtd/ServerFiles/Mizzou1.jpg
```

Figure 2 – Server

```
ts7250-07:/home/ddwxtd/ProjectClient/Debug# ./ProjectClient
Enter command: LIST
total 8
drwxr-sr-x    2 root     510         4096 Dec  7 13:35 .
drwx--S---   19 510      510         4096 Dec  7 13:35 ..
drwx--S---   19 510      510         4096 Dec  7 13:35 ..
Enter command: UPLOAD
Enter filename to send: /home/ddwxtd/Mizzou1.jpg
Enter command: LIST
total 36
otal 36
drwxr-sr-x    2 root     510         4096 Dec  7 13:35 .
drwx--S---   19 510      510         4096 Dec  7 13:35 ..
-rw-r--r--    1 root     510        28189 Dec  7 13:35 Mizzou1.jpg
Enter command: DOWNLOAD
Enter filename to receive: Mizzou1.jpg
Enter saved location: /home/ddwxtd/Test/M1.jpg
Enter command: DELETE
Enter file to delete: Mizzou1.jpg
```

Figure 3 - Client

I started out by listing the files on the server. It showed that there were no files currently located on the server. Then I uploaded the Mizzou1 file to the server. The original file

size was 28, 189 bytes. Following the upload, I listed the files again to ensure that the file was transmitted. It showed that the size of the file was 28,189 bytes which was the same as the original file so it was a successful transfer. I repeated this test ten more times with different files. The results are found in **Table 1**. Out of the ten tests I ran all ten of them showed the same number of bytes on both the server and client. Therefore the program was a success.

Table 1 - Test Results

File Name	Size on Client (Bytes)	Size on Server (Bytes)
Testing.txt	7	7
Mizzou1.jpg	28,189	28,189
Mizzou.jpg	184,156	184,156
Cubs.bmp	921,654	921,654
TestAgain.txt	9	9
Tiger.bmp	13,398	13,398
A10.jpg	87,330	87,330
F16Cockpit.jpg	87,825	87,825
Johnny5.jpg	77,201	77,201
SaleenS7.jpg	62,757	62,757

Conclusions and Future Work:

This program dealt with the transferring of files from multiple clients to a central server. The server allows for simultaneous access of the files pending the clients aren't accessing the same file. The program works as indicated by the test results in **Table 1** so this project was considered a success. Given more time, some improvements could be made. One improvement would be to provide more error checking algorithms. The way mine was set up, you had to ensure that the directories preexisted and that the file names were spelled correctly. Also, I would like to make the system more secure by creating a login for the server and perhaps encrypting the files during the transfer. In addition it would be nice to have a GUI setup where the clients could drag and drop files to or from the server.