CS371 Project 1: Remote Folder

Implementor's Notes

Andrew Donovan
Department of Electrical and Computer Engineering
University of Kentucky, Lexington, KY USA
amdo257@uky.edu

ABSTRACT

This Project was designed to implement a remote folder. This was done in two sections, Server, and Client. Each section is called via the command line.

1. INTRODUCTION AND MOTIVATION

Remote folders are common implementations to allow for sharing of resources. In this case the design builds two sides client and server. The client sends requests to the server and the server responds with the required information. This is done by the client sending pre-agreed messages to the sever.

The motivation for this project was to deepen the understanding of server and client-side operations over the network. And to grow in understanding of socket programming of which this entire project is based off and uses.

2. IMPLEMENTATION AND DESIGN

The project is broken down into two main sections, the server, and the client. Both are very similar in structure in both programming and organization. Staring with the server side. Each section has a main and a header file then each of the sub functions has its own .cpp file. This is to help with both clarity and debugging.

a. The server main includes its own header file so that it keeps the size down and increases clarity and allows for the needed global variables to be kept in one place. Those global variables being the size of the buffer being used, the maximum number of logs that the server can handle. And then the current number of logs that the server is has. Along with this it also describes the struct called 'log_server'. This struct keeps all of the necessary information that the client could request, such as the file name, size of the file, when it was uploaded and the number of times that it (the file) had been downloaded. Then the server main includes all the implemented functions, these will be discussed later. The main loop of the function loops waiting for a connection; whose parameters were passed via the command line. In this implementation only the port number is needed, the client should connect to the IP address that the server is at. The once the connection binds it then waits for the client to send the username and password. During all of this the server prints out what it is doing broken up by a bar of equal signs to demark where new items are being worked on. This is the authentication that the server has.

b. The authentication is implemented in 'login.cpp', in which the server main passes the username and password as strings. The function then compares them to an internal text file called 'passwords.txt', and if they do match then, the function sends back a one. If they are not in the document then the function sends back a negative one and the server closes the connection. Once the go ahead that the username and password are good to go, the server waits for commands to be passed in. Upload is one of those commands.

c. The upload function is completed in 'upload_S.cpp'. In this the server passes in the current socket id and the array of log's (described above). Then because the socket is open it reads in the file name and uses 'open' with a write option to create and start writing a file. If that errors for some reason it reports back the failure to the client. If not it sends to the client an all good message. Then the function takes in the size of the file. If the file size is zero it sends an error message to the client. If not it sends a success message to the client. After that it updates the log structure with all of the requisite information (as described above). The updating of the log is set here because the server should not shut down between clients. Then the function loops taking in the rest of the data. After this it closes the file and returns. In any part where an error messages is sent the functions will send back a negative one to the main to let it know that an error has occurred. The next command is Download.

d. Download is implemented in 'download_S.cpp', this is also passed the current socket and the array of logs. It also takes in the name of the file, but then checks it against the names in the log array, if its not in there it returns an error. If it is in the log array responds the function responds with a go-ahead message? Then it opens the given file and defines the size of it and sends that to the client. Once that is sent and acknowledged, the function loops sending the rest of the data. After this it increments the number of downloads of the file and closes it and then returns. The next command is directory.
e. The directory command is implemented in 'stats.cpp', it is also passed the current socket id, and the array of logs. The function checks for issues, and then loops sending all the information about the files in the folder, this is done via sending the data in parts and waiting for a go ahead from the client. The last command is Delete.
f. The Delete command is implemented in 'del_S.cpp'. For this it is passed the current socket id and the array of logs. It checks if the file exists it removes it and updates the log and number of files and lets the client know if it succeeds or fails.

On the client side it has the same commands sans login as that is just passed to the server. The implementations are very similar to the server side the only differences are that the client is receiving more of the data in most cases. Also around the upload and download commands there is code that starts and stops a timer so that there was a way to get speed across the network. And there are separate timers internal to the actual socket commands getting the 'data-rate'. This time is written to a file so that the below graphs could be made. The timer started right before the function was called and stopped directly after it.

a. The client main function takes in from the command line the IP address of the server and its port number. It also includes all its own header files wherein it is defined several variables (the same ones as on the server side sans the number of files) and a log struct that is a mirror to its counter part on the server side. It then reaches out and connects to the server. Then it asks the user for their username and password and sends that to the server. If they are legit it asks for the next commands, if not the connection is closed. It then loops taking in commands until the user types exit or the connection is closed. One of the commands that the user can input is upload.
b. The upload command is implemented in the 'upload.cpp'. It is only passed the socket id. The function then asks for the file name they would like to upload and send this to the server. It then opens the file and defines it as well. Then the function using the definitions it sends off the size of the file to the server. It then loops sending the data, until its finished then it returns after closing the file. The next command is download.
c. The download command is implemented in 'download.cpp', it is also only passed the socket id. It also asks the user to enter the file they would like to download and sends that off to the server. If the file exists on the server, the client creates it locally with the 'fopen' function. It then takes in the size of the file and it loops taking in the data and writing it to the open file. Once it done it closes the file and returns. The next command is directory.
d. The directory command is implemented in the 'dir.cpp'. This command is passed the current socket id and the array of

ACM ISBN N.A.
DOI: N.A.

logs. It then takes in the number of files from the server. Then loops taking in the requisite data and storing it in the array of logs. Once it's done writing to the array it prints out the contents to the screen so that the user can see it. The final command is Delete.

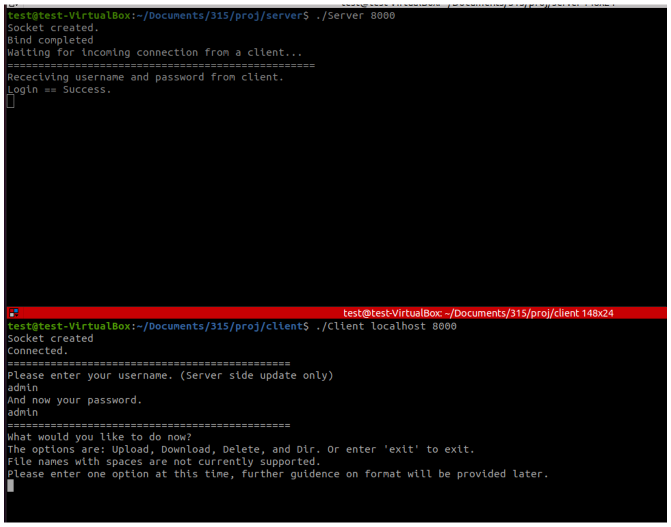
e. The Delete command is implemented in 'del.cpp'. This command is passed in the current socket id. Then it asks the user for the name of the file they wish to delete on the server. It then sends that request to the server and prints out what the server returns.

The other files in the .zip are the README.md, this document and some of the test files. The README file goes over the compilation of both the server and client programs. It also, gives a bit of detail on what each file is for and how to call each program after they have been compiled. It also goes over how to create large test files which were used in the experiments section. The other test files were there to give a smaller test with different file formats. Also included are the screenshots that are included in this report.

3. EXPERIMENTS

The experiments were conducted using only local host so the data may not line up with other devices and connections.

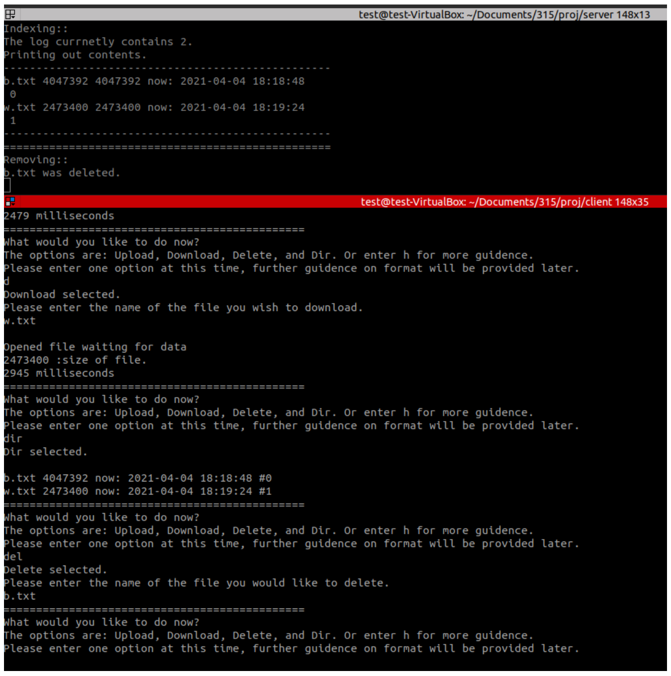
The screen shots included below are executions of each command and what appeared on both the server and client sides.



```
test@test-VirtualBox:~/Documents/315/proj/server$ ./Server 8000
Socket created.
Bind completed
Waiting for incoming connection from a client...
==================================================
Receciving username and password from client.
Login == Success.
```

```
test@test-VirtualBox: ~/Documents/315/proj/client 148x24
test@test-VirtualBox:~/Documents/315/proj/client$ ./Client localhost 8000
Socket created
Connected.
==============================================
Please enter your username. (Server side update only)
admin
And now your password.
admin
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter 'exit' to exit.
File names with spaces are not currently supported.
Please enter one option at this time, further guidence on format will be provided later.
```

Above is the successful execution of authentication.



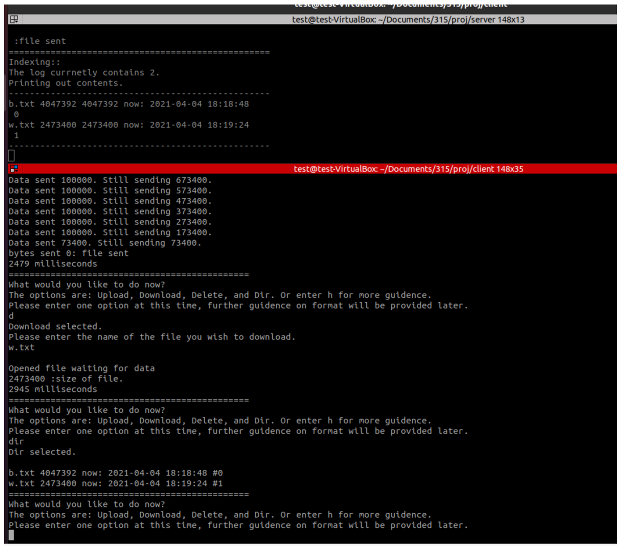
```
test@test-VirtualBox: ~/Documents/315/proj/server 148x13
Indexing::
The log currnetly contains 2.
Printing out contents.
--------------------------------------------------
b.txt 4047392 4047392 now: 2021-04-04 18:18:48
 0
w.txt 2473400 2473400 now: 2021-04-04 18:19:24
 1
--------------------------------------------------
==================================================
Removing::
b.txt was deleted.
```

```
test@test-VirtualBox: ~/Documents/315/proj/client 148x35
2479 milliseconds
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
d
Download selected.
Please enter the name of the file you wish to download.
w.txt

Opened file waiting for data
2473400 :size of file.
2945 milliseconds
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
dir
Dir selected.

b.txt 4047392 now: 2021-04-04 18:18:48 #0
w.txt 2473400 now: 2021-04-04 18:19:24 #1
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
del
Delete selected.
Please enter the name of the file you would like to delete.
b.txt
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
```

Above is the successful execution of the Delete command.



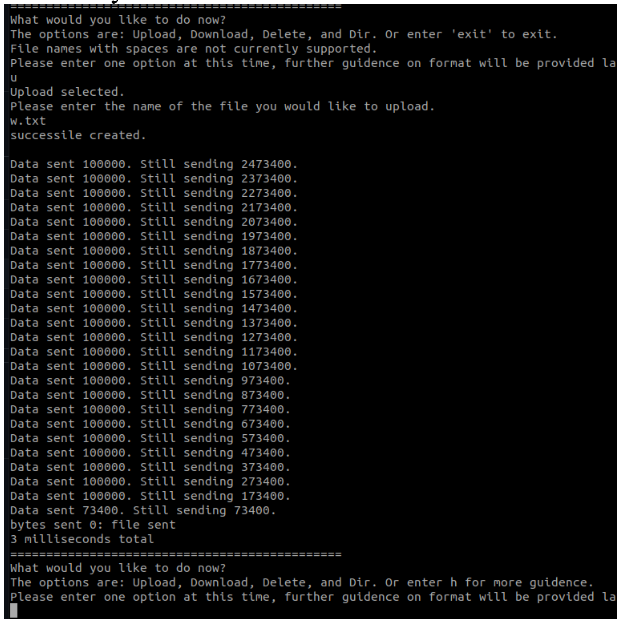
```
test@test-VirtualBox: ~/Documents/315/proj/server 148x13
 :file sent
==================================================
Indexing::
The log currnetly contains 2.
Printing out contents.
--------------------------------------------------
b.txt 4047392 4047392 now: 2021-04-04 18:18:48
 0
w.txt 2473400 2473400 now: 2021-04-04 18:19:24
 1
--------------------------------------------------

test@test-VirtualBox: ~/Documents/315/proj/client 148x35
Data sent 100000. Still sending 673400.
Data sent 100000. Still sending 573400.
Data sent 100000. Still sending 473400.
Data sent 100000. Still sending 373400.
Data sent 100000. Still sending 273400.
Data sent 100000. Still sending 173400.
Data sent 73400. Still sending 73400.
bytes sent 0: file sent
2479 milliseconds
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
d
Download selected.
Please enter the name of the file you wish to download.
w.txt

Opened file waiting for data
2473400 :size of file.
2945 milliseconds
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
dir
Dir selected.

b.txt 4047392 now: 2021-04-04 18:18:48 #0
w.txt 2473400 now: 2021-04-04 18:19:24 #1
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided later.
```

Above is the successful execution of the Directory and Download commands.

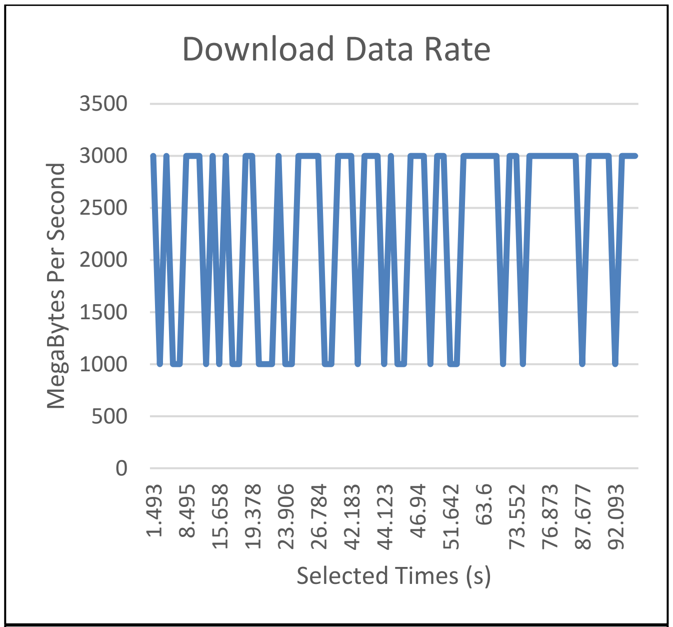


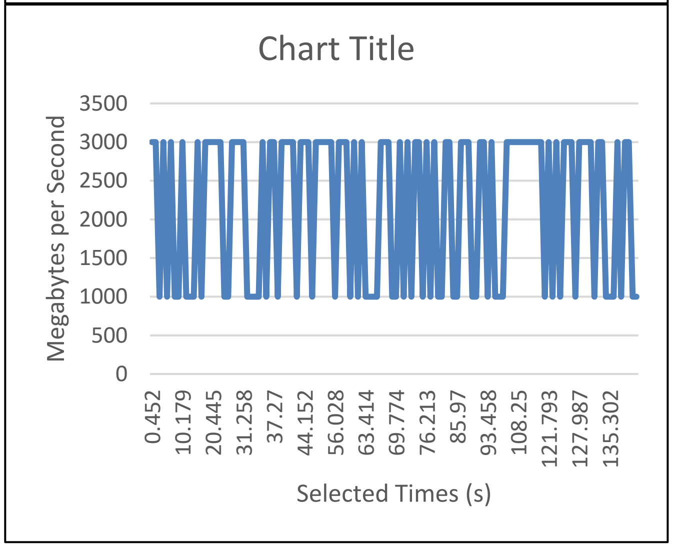
```
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter 'exit' to exit.
File names with spaces are not currently supported.
Please enter one option at this time, further guidence on format will be provided la
u
Upload selected.
Please enter the name of the file you would like to upload.
w.txt
successile created.

Data sent 100000. Still sending 2473400.
Data sent 100000. Still sending 2373400.
Data sent 100000. Still sending 2273400.
Data sent 100000. Still sending 2173400.
Data sent 100000. Still sending 2073400.
Data sent 100000. Still sending 1973400.
Data sent 100000. Still sending 1873400.
Data sent 100000. Still sending 1773400.
Data sent 100000. Still sending 1673400.
Data sent 100000. Still sending 1573400.
Data sent 100000. Still sending 1473400.
Data sent 100000. Still sending 1373400.
Data sent 100000. Still sending 1273400.
Data sent 100000. Still sending 1173400.
Data sent 100000. Still sending 1073400.
Data sent 100000. Still sending 973400.
Data sent 100000. Still sending 873400.
Data sent 100000. Still sending 773400.
Data sent 100000. Still sending 673400.
Data sent 100000. Still sending 573400.
Data sent 100000. Still sending 473400.
Data sent 100000. Still sending 373400.
Data sent 100000. Still sending 273400.
Data sent 100000. Still sending 173400.
Data sent 73400. Still sending 73400.
bytes sent 0: file sent
3 milliseconds total
==============================================
What would you like to do now?
The options are: Upload, Download, Delete, and Dir. Or enter h for more guidence.
Please enter one option at this time, further guidence on format will be provided la
```

Above is the successful execution of the Upload command.

As for the two following charts they follow a very straight forward testing methodology. A file was uploaded and its Megabytes per second were record. Not all points were record points were selected if the counting integer divided by 250 equaled zero. Owing to the speed of this machine the counters needed to be in milliseconds if not micro so if either timer equaled zero when passing the checks it was also discarded. When ever the parameters would be meet the Bits per second along with the time that it occurred relative to the start of each action; was written to a CSV file. From there the times were converted from milliseconds to seconds and the Bits per second to Megabytes per second. This conversion was done to meet the lab spec. Also when the data is written to the file it outputs on the client side the current data rate in bits per second.





4. CONCLUSION

This project works for all file types. The ease and usefulness of this project was interesting to see so many parts of the education come together to work so well. Each function and command works as intended and catches common errors and informs the user. Also, one extension was completed, the authentication was implemented on the server side. This entire project was done by myself.

One of the issues I ran into is using the 'sendfile' function, this function can send at most 2.1 GB of data, which limits the size of files that I can send, which is the reason for none of the test files being larger than 1 GB or for any of the times being over two seconds. I can increase the time via changing the size of the buffer that the client and server use. But seeing as this project was tested using one machine and local host the speed is only limited by the kernel on the machine handling both calls of the server and client. So for the results above, this is why the speeds are so high, for both upload and download. As for the networking side these speeds would more than likely not be possible to achieve. Owing to the fact that there is more overhead and other errors that can arise.

Overall, the project works well and executes on the given parameters.

ACM ISBN N.A.
DOI: N.A.