Sending and Receiving Files (125 points)

This lab may be done with one other student in any section of this class.

So... now you know how to make sockets and use them in C... or Java, or some other language. Using the language of your choice (it must be a base language, nothing like Python Flask or Ruby on Rails, etc. If you're not sure, just ask! But ***this assignment may be done in Python, Java, C, etc.***), you are to make a server that handles two kinds of requests:

1. iWant
2. uTake

That is, the client now merely takes commands on the command-line, and then the client uses its socket connection to the server to send the command as-is to the server.

NOTE: Below, you are given things to output in the case of an error. You can use whatever error messages you want, feel free to get creative.

The server is to parse the command. The only commands accepted currently are iWant and uTake. If neither of these is the command given to the server, the server should respond to the client with a message "That just ain't right!" and move on to the next command. This response can be used for any number of generic errors found in the command.

iWant

On the client side, whenever a user types "iWant xyz" where xyz is the name of a file, the string is sent to the server without modification. The server must then parse the command and if the command is well-formed AND the file exists, sends the requested file. If the file does not exist, the server should send the string "Failure: What you talkin' bout Willis? I ain't seen that file anywhere!" For any other type of malformed request, you can send a generic error message like the one above.

On the client side, you should ask what directory the user wishes to place the file before the file transfer begins. If the user leaves that empty (by just hitting enter), then you should save the incoming file in a default folder as described below.

uTake

On the client side, whenever a user types "uTake xyz" where xyz is the name of a file, the client must validate if the file exists on the client-side (printing an error message if not), the string is sent to the server without modification if the file exists on the client-side informing the server that it should accept a file named xyz. The client can either send the file WITH the message (if the server is ready for it), or better, you can choose some protocol where the server informs the client that it is ready to receive the file after it has parsed the command successfully. In either case, the client is to send the server the file named xyz.

Directory Structure

Please keep the client and server programs in a separate directory! If you fail to do this, sending and receiving files (especially when using localhost) will get messy.

Please keep all client-side code in a directory named something like 'Lab2/client' where the files received from the server should default to 'Lab2/client/received_files'

Please keep all server-side code in a directory named something like 'Lab2/server' where the files kept to send to the client are kept in 'Lab2/server/store'. Files received from the client may be stored here by default or in another folder named something like 'Lab2/server/received_files'.

Extremely Important Considerations

Your program should work with any type of file (.txt, .pdf, .gif, .jpeg, mp3, mp4, etc.). It will be easy to start making this work for text-only files because that will be the most direct extension of your first lab. However, since this should work with ANY file type, you need to work with files in **binary mode**. It is very important that you read and send the files in binary mode (in whatever language you choose) because with text, the '\0' (null character) indicates the end. However, in any file type other than a text file, this character (or rather the binary encoding of this character) may appear in the file several times. For example, in C, when you are opening the file you should make sure to use fopen with the binary option (simple google search will work) and use fread and fwrite. When using these, a null character is not the end of the input/output, so you must be specific with exactly how many bytes are being read or written.

Note, this is pretty straightforward in other languages like python. This lab in python will be MUCH simpler if you are struggling (several students have started this lab in C, struggled a good amount and switched to python and finished in a matter of an hour or a few).

This lab must work with arbitrary file types and sizes. But if you are successfully able to send an mp3, you're probably where you need to be.

Application Protocol

You will DEFINITELY need to develop your own application protocol for this lab! When the user types "iWant file.pdf" there should be several points of communication between the client and server. You will probably want to communicate between the client and server things like the file size, where it is located or where it shoudl go, etc. In other words, you probably won't be able to do everything in just a simple send-the-command-and-get-a-file interaction.

When the file is finished being sent/received, you should maintain the connection, or start a new one (transparently to the user!). This means in the same run of one client/server program, you should be able to send and receive multiple files.

Note, the commands are meant to be typed only on the client so that iWant asks for a file from the server and uTake requests to send a file to the server.

You can specify the directory structure in the commands or in a separate follow-up command. For example, you can do one of the following two:

```
> iWant directory1/directory2/directory3/filexyz.txt
```

Or:

```
> iWant filexyz.txt
  From Server: not found in default directory, please provide the directory
> directory1/directory2/directory3
```

Note that the directory structure will work with absolute paths (/root/home/...) or relative (../someOtherDirectory/...). The first above will reduce your overall back-and-forth message sending between server and client but the second one may be easier depending on the protocol you develop.

Example Output

For this example, you can assume the server has a directory with files with the program that looks like the following.

Server side:

```
> pwd
/labs/Lab2/server

> ls
dir1  dir2  file1.txt  file2.pdf  server.c  server

> ls dir1
file3.jpg file4.txt

> ls dir2
file5.gif

> ./server 6789
```

Below is an example interaction on the client given the files that are on the server (you do NOT need to match the output below EXACTLY but fairly close would be preferable):

```
$> ./client localhost 6789
> iWant dir1/file3.jpg
  What directory would you like to save this file?
> .
    file transfer started...
    file transfer of 3172645 bytes complete and placed in current directory
> iWant /labs/lab2/server/dir2/file5.gif
  What directory would you like to save this file?
> localDir/someOtherDir
    file transfer started...
    file transfer of 537985 bytes complete and placed in localDir/someOtherDir
> uTake clientFile.pdf
  What directory on the server would you like to save this file?
> dir1
    file transfer started...
    file transfer of 14253 bytes to server complete and placed in dir1
> iTypoWant filexyz.pdf
  That just aint right!
> uTypoTake filexyz.pdf
  That just aint right!
> iWant dir1/nonExistentFile.jpg
  What you talkin bout Willis?  I aint seen that file anywhere!
> uTake fileNoExistOnClient.txt
  What you talkin bout Willis?  I aint seen that file anywhere!
> exit
  See ya!
```

The server can display whatever you want, but the interaction is all done from the client.

Incentive Opportunities

(20 points) Do this lab in C

(20 points) Get this lab working where the client can enter the commands above, but ALSO the server can enter the same commands and it work in either direction (the client must still be able to send and receive files with uTake and iWant, but also the server). You will need either a multithreading or multiprocess server to do this

(20 points) Get this working so that the server can serve multiple clients on separate connections at the same time (where each client can be sending and receiving files simultaneously)

(10 - ??) After completing this lab, completing it in other languages will earn incentive. The total depends on which languages, how many versions and what is negotiated during the lab demo

(10 points) Handle directory creation when client gives a directory to save files that does not exist yet

... You can come up with others and the incentive points can be negotiated when you demo this lab.

Last modified: Tuesday, 2 April 2024, 12:37 PM

Contact site support

You are logged in as Rebecca Breland (Log out)
Data retention summary
Get the mobile app

Powered by Moodle