Program 2
File Transfer Utility

Prof. Kredo

Due: By 23:59 Friday, March 13

Introduction

In this program you will accomplish several goals:

- Further investigate the socket API
- Write the client and server portions of a file transfer program

All programming assignments must be done in pairs.

Requirements

You may use the updated client and server socket programs from your textbook as a base for this program. The updated versions, named `stream-talk-client.c` and `stream-talk-server.c`, are available on Learn. Even though I allow you to use the program, it is good practice to indicate you started with this code in your comments.

Your objective for this program is to write a file transfer utility. The server portion waits for a file request and responds with either an indication of error or the file requested and the client portion requests and saves a file from the server based on user input.

The client should follow these general steps:

1. Accept the filename from the command line
2. Send the filename to the server
3. Receive the server's response and process it
   (a) If the server indicates an error, display a message to the user and quit; no file is saved
   (b) If the server indicates success, receive and save the file to the local directory **using the filename from the command line** and quit

The client may overwrite the local file if it exists. You should differentiate the error messages caused by errors on the client side (e.g., connection refused, missing arguments) from the errors received from the server (file unavailable).

The server should follow these general steps:

1. Wait for a client connection
2. Receive the filename from the client
3. Send a response to the client
   (a) If the server can't read the file for any reason (doesn't exist, bad permissions, etc.), then send an indication of error to the client and quit
   (b) If the server can read the file, then send the file to the client and quit

The format and structure of the messages sent from server to client are up to you, though you should strive for low overhead in your messages. Your programs do not need to differentiate the possible error conditions related to the file on the server (doesn't exist, bad permissions, etc.). A simple error that encompasses all possibilities is acceptable.

Additional requirements for this program include:

- All submissions must pass compilation checks before they will be graded. Run the script `/user/faculty/kkredo/public_bin/program2_check` on `jaguar` to check your submission. Run the script without arguments or with the argument `help` for directions.
- The client should accept three command line arguments. The first is the server, which may be provided as an IP address or a hostname. The second argument is an integer indicating the server port number. The third is the file name to request from the server, which is also the filename used to save the file.
- The server should accept one command line argument: the port number on which it should listen, as an integer.
- You must include a Makefile with your submission.
- The client executable must be called `file_client` and the server executable must be called `file_server`.
- Your program must compile cleanly on the lab computers in OCNL 340 or `jaguar` and you must use the gcc/g++ argument `-Wall`.
- Check all function calls for errors. Points will be deducted for poor code.
- Put both group member names, the class, and the semester in the comments of all files you submit.

Input/Output

Below is an input and output example for the client. Your server should not have any output. Make your programs match the style and formatting of these examples, but you are free to write your own error messages.

```
$ ./file_client farnsworth.ecst.cschico.edu 5000 some_file
Client Error: Unable to find host ‘farnsworth.ecst.cschico.edu’
$ ./file_client farnsworth.ecst.csuchico.edu 5000 some_file
Server Error: Unable to access file ‘some_file’
$ ./file_client farnsworth.ecst.csuchico.edu 5000 that_file
```

Evaluation

Your program will be evaluated based on:

- 20 points: Filename sent to server
- 30 points: File contents sent to client
- 20 points: File contents saved to file by client
- 20 points: Server sends file unavailable error
- 10 points: Client displays file unavailable error

Hints

- Beej's Guide to Network Programming is a valuable resource.
- Make no assumptions about the file contents. Any bit pattern may appear at any point in the file. **Do not use string functions to handle file data!**
- Test and debug in steps. Start with a subset of the program requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Grading requirements are ordered in a way to ease this implementation process.
- Your programs (client and server) should close when the file has been transferred. The answer on how to do this lies in `recv(2)` (HINT:"orderly shutdown" means to close a connection).
- You may have an optional command line argument that enables debugging output (for an example, see the server program from Program 1). The debug argument must be optional (meaning that your program must run when it is not specified) and the debug output in your code must be cleanly organized.
- For the declarations `char buff[256]; char *bp = buff;`, ensure you know the difference between `sizeof(buff)` and `sizeof(bp)`. **They are not equal!**