CIS*3210
Assignment 1

Description

In this assignment, you need to write a simple server (C on Linux) capable of receiving text data (messages) from clients over TCP sockets and a client (C on Linux) for sending them. Each complete message is large, and will be broken by the client into multiple pieces for transfer. The server should print these text messages on the standard output.

The server should be listening for messages on a specific port. It should handle the client connections sequentially and accept connections from multiple clients. It would respond to clients one at a time - in this case, responding to a client means receiving the entire message from this client. After receiving the entire message from one client, the server would move on to the next. If multiple clients try to simultaneously send text messages to the server, the server should handle them one at a time (in any order).

Each "message" that a client sends is a file. The client should read the file, transmit it, and exit. You can assume that the client is run as

$./client server-IP-address filename

where "server-IP-address" is the IP address of the server, and "filename" is the text file the client sends. The server is run as

$./server

Since we will end up with multiple servers running on the same host, each of you will need a unique port number. **Use the port number from your labs, except replace the left most 1 with a 2. 16789 becomes 26789.** This is for the assignment only. Continue using 16789 (with your own numbers) for the lab server.

If the server cannot bind on a port, print a message to standard error. Assume that the file contents can be arbitrarily large, but the buffers you use to read/write to the file or socket must be small and of fixed size (e.g., 4096 bytes). (*The client should have an optional command-line argument that specifies the length length of the buffer; if an optional buffer is not provided, use a default size of your choice. Optional.)*

Make sure you do basic error handling. It's harder to do with blocking calls, since we can't do time-outs, but make sure you handle invalid files, invalid command line args, failure to bind, etc..

Make sure you handle the following correctly:

- **Local and remote operation**: Your program should be able to operate when connection over both localhost (127.0.0.1) or between machines. You can also use getaddrinfo() to get the IP address of the target server.

- **Handling return values**: By default, sockets are blocking, and for this assignment we will use only blocking sockets. "Blocking" means that when we issue a socket call that cannot be

done immediately (including not being able to read or write), our process waits until it can perform the action. This includes the case when
- a socket's internal buffer is full and therefore, no data can be written, or
- a socket's buffer is empty, and no data is available to be read.

However, if there is some data available to be read or some can be written, the call will return the number of bytes read or written respectively.

NOTE: This returned value (e.g. number of bytes read or written) can be less than the length specified in the function call. It can also indicate an error. You must handle this.

Required files

- All the .c and .h files for your client and server
- A Makefile that compiles the server and the client
  - make all creates executables server and client
  - make clean deletes all executables and intermediate files
- A README file that describes how to run the script(s) and provides all other instructions necessary to run your assignment

Submission

Create a tar.gz archive with all your deliverables and submit it on Moodle. The filename must be FirstnameLastnameA1.tar.gz. Do not include any binary files in your submission.