CIS*3210
Assignment 1
Deadline: Friday, October 5, 9:00am
Weight: 10%

Description

This assignment can be done **in teams of two**. You are welcome to use any of the code examples that I have provided for this course.

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

You will do some tests on your code and report your results.

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 service clients one at a time - in this case, servicing 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:port-number
```

where "server-IP-address" is the IP address of the server, and "port-number" is the TCP port the server listens on. The server is run as

```
$./server port-number
```

Since we will end up with multiple servers running on the same host, each of you will need a unique port number. I will post a list on Moodle with your names and port numbers. Use whatever port number you wish for initial testing. I assume that your initial development/ testing will use the localhost, anyway.

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.

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 to 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 is 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.

Testing

Make sure that your server can handle more than one client. Create a script (bash or python) that spawns several clients that try to write to the server simultaneously.

You will need to test your client and server in four different environments:

- On the same machine
- Client at home, server on `linux.socs.uoguelph.ca` (use a VPN when connecting from home)
- Client at school on a wireless connection, server on `linux.socs.uoguelph.ca`
- Client at school on a wired connection, server on `linux.socs.uoguelph.ca`
- Remember that `linux.socs.uoguelph.ca` is an alias for several different hosts (`ginny.socs.uoguelph.ca`, `percy.socs.uioguelph.ca`, etc.). Make sure you know which host you're connecting to.

For each environment, record the following:

- Ping time - let the ping utility do at least 50 pings, record the stats (round-trip min/avg/max/stddev, packet loss rate).
- The route from output from client to sever (use traceroute to figure it out).
- The average, minimum, and maximum time that your code took to do the transfer. Do at least 20 file transfers, use the same file each time.
- The min/max/average transfer rates.

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 script for spawning and running multiple clients simultaneously

- A `README` file that describes how to run the script(s) and provides all other instructions necessary to run your assignment
- A report that shows the test results from your four test environments.

Submission

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