**CSC 3304 Prog 6 (200 points) Fall 2024**

**Due on Tuesday Dec 3rd by the start of class**

**Late deadline is two days later (Thursday Dec 5th): -10%**

**Turn it in any time before the final exam: -20% (make sure to email me if you turn it in after the 3rd or after the final)**

**If you turn nothing in: -50%**

**Cheating: 0%**

**Assignment rules (read carefully):** Like the previous program, this program will not be graded on *correctness,* however thereare still requirements as described below. To get a high grade, do your own work on the topic below, turn it in on time and make sure to correct warnings and errors (which I can help you with if you start early). I suggest each task not marked as advanced, at minimum, to understand the material well. At that point with errors and warnings corrected, you can turn it in for full points, then attempt the advanced suggestions for additional understanding and practice (without worrying about errors or warnings affecting your grade).

**Requirements:**

1. For this program, your files must compile via a working makefile using the make command. The network examples come with a make file already setup so you may use that as is and just modifying the client and server programs. If you switch to a single program, make sure to modify the makefile accordingly. You must test your programs using multiple Putty instances, one for the server and one per client.
2. Set your client and server to use a fixed port equal to 30000 + ID. If your classes login ends with numbers, just use that as your ID. Example: cs330499 will ID=99 so port is 30000+99 = 30099. If your ID ends with letters such as aa, convert the left letter a=100, b=200, c=300 and so on. Then convert the right letter to a value from 1 to 26 (a=1, b=2 and so on). So cs3304aa uses port 30000+100+1=30101. And cs3304bf must use port 30000 + 200 + 6 = 30206. You must use your assigned port or you may have conflicts with other students.
3. You must do your own work and it must be on the current topic described below.
   * I will use a program that compares your program to those of other students and to my examples to check for similarity. If your program is grouped with others due to similarity, I’ll manually review them to check for cheating. This checker is not fooled by changing variable names, order of statements, adding comments or other basic program changes.
4. For full points, you must still correct errors (especially) and warnings generated by using `flags` when compiling.
   * Note: code you find on the Internet is likely to trigger a lot of compiler warnings as it’s usually not written with our expanded compiler warnings in mind. Also, if two or more students copy a large part of the same code from the Internet, this will look like cheating (because it is) so don’t take that risk. It’s ok to be inspired or informed by outside code, just avoid copying it, especially large blocks of it.
5. I’ll also be checking for excessively simple or empty programs. It’s easy to avoid warnings and errors if your program does very little. This may be considered cheating if very excessive or otherwise may result in a 50% to 70% grade (similar or worse than a program with errors).

**Topic and Suggested Work**: Networking (example 18a-networking or 19a/19b)

1. Modify the first network example so that the client and server engage in back-and-forth communication by putting the read and write code in a loop (recall both server and client can read from or write to the connection socket once established).
2. Notice that when either client or server is waiting for the user to input a message, it will block and thus cannot receive and print messages on the socket, and similarly while waiting to read from the socket, the user cannot input messages to send. Reprogram your client and server to handle reading and writing in separate threads to avoid this problem. This will allow each program to receive user or socket input asynchronously. Note: printf output sent to screen by the socket reading thread will interrupt partially typed user input (this is not correctable with the material we’ve learned this semester).
3. **Advanced**: modify the server to accept multiple clients and spawn read/write threads for each. Allow the clients to interact through the server such as implementing a game or simple chat server where the server passes messages from one client to other clients. Use mutex protected shared structures to share data among threads as needed.