UNX511 Lab 9: Server with Multiple Clients and a Receive Thread with Mutexing

Due: Sunday, August 3, 2025 (11:59pm)

NB: Due to the length and complexity of this lab, this lab will be considered as two labs. Meaning, this lab is worth **2%** of your final mark.

In this lab you will create a server (**server.cpp**) that communicates with three clients using a receive thread for each. The client code has been given to you along with the Makefile, a start-up script, and a stop script in case you have to manually stop all the clients. You can retrieve these from the following links:
client.cpp,
Makefile,
startClient.sh,
stopClient.sh.

Your server will use non-blocking stream sockets in the internet domain. Your socket will bind to the **localhost** (**127.0.0.1**) and the port number will be specified from the command line. For instance, if you want to use port 1153, you would start your server and clients with:
$ ./server 1153
$ ./startClient.sh 1153
Be sure to start your server first. You will probably want to start your clients in another window. **startClient.sh** will start three clients with a 1 second delay between each.

Once your socket is bound, you will listen for connections. Be sure to distinguish between the master file descriptor and the connection file descriptors for each of the clients. You will have one master file descriptor for listening and accepting, but three connection file descriptors for each of the three clients.

Once a client connects, you will start off a **receive thread** to service all socket reads from this client. In this way you avoid using the **select()**function. Be sure to pass through the connection file descriptor for that client to the thread as a parameter. This means your socket reads and writes are no longer synchronized as before. They are asynchronous. You will perform socket writes from your **main()** function and socket reads from your **receive threads**.

The **receive thread** will change the options of its connection socket to set a read timeout of 5 seconds. Documentation on how to do this can be found at:
setsockopt - Linux man page,
Chapter 61 page 624 of the text Advanced Programming in the UNIX Environment, and
the article Linux: is there a read or recv from socket with timeout?.

Since communications are now asynchronous, the server does not have to instruct the client to send data. As soon as the client connects to the server, the client will start sending data (see client.cpp).

When a client sends text to the server, the server will read the data in its **receive thread** and push it into a message queue. The message queue could be of type:
queue<string> message;

The **main()** function, in its infinite while-loop, will test for new connections with a non-blocking call to **accept()**, and write the contents of the message queue to the screen if there is anything in the message queue. If the queue is empty, **main()** will sleep for 1 second and check the queue size again. Be sure to pop the message off the message queue once you have written it to the screen.

The infinite while loop (conditioned on **is_running**) in **main()** should look something like this:

- Have we accepted all clients?
    - If not, call **accept()**. Since the master socket file is non-blocking, **accept()** will not halt the code if no client wishes to connect.
    - If a client wishes to connect, create a thread for that client, passing through the connection file descriptor and increment the number of clients.
- Print out to the screen anything in the message queue. Be sure to mutex this message queue since it is used in the receive thread as well.
- Sleep for one second.

The receive thread should look like this (the same code can be used for all three clients):

- Extract the connection file descriptor from the argument passed to the receive thread.
- Use **setsockopt()** to set the read timeout to 5 seconds.
- Enter infinite while loop conditioned on the variable **is_running**.
    - If there is something on the read, add it to the message queue. Be sure to mutex this message queue since it is used in **main()** as well.
    - Note that read should block for a maximum of five seconds.

The communications stop when a ctrl-C is issued to the server. The signal handler for the server will set **is_running** to **false** causing the infinite while-loops in the **receive thread** and in **main()** to end. The server in its **main()** function will then send **"Quit"** to each client, close all connections, and exit. The clients will interpret **"Quit"** to mean it is time to finish and they will shutdown as well. Be sure to join the **receive threads** to **main()** before **main()** exits and be sure to close all connection file descriptors, as well as the master file descriptor.

Questions

1. What is the difference between synchronous and asynchronous communication?
2. Do you prefer socket reads in a receive thread or do you prefer both socket read and write to be in the main() function? Why?

Assignment Submission:

- Complete all steps, Add all output-screenshot and explanations (if required) to a MS-Word file.
- Add the following declaration at the top of MSWORD file and source code

```
/*********************************************************************************
* UNX511-Lab9
* I declare that this lab is my own work in accordance with Seneca Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
*
*********************************************************************************/
```

- Please answer the following two declarations:
    - D1) On a scale from 1 to 5, **How much did you use generative AI to complete this assignment?**
        - where:
        - **1** means you did not use generative AI at all
        - **2** means you used it very minimally
        - **3** means you used it moderately
        - **4** means you used it significantly
        - **5** means you relied on it almost entirely
        - **Your answer :**
    - D2) On a scale from 1 to 5, **How confident are you in your understanding of the generative AI support you utilized in this assignment, and in your ability to explain it if questioned?**
        - where:
        - **1** means "Not confident at all – I do not understand the generative AI support I used and cannot explain it."
        - **2** means "Slightly confident – I understand a little, but I have many uncertainties."
        - **3** means "Moderately confident – I understand the majority of the support, though some parts are unclear."
        - **4** means "Very confident – I understand most of the AI support well and can explain it with minor gaps."
        - **5** means "Extremely confident – I fully understand the generative AI support I used and can clearly explain or justify it if asked."
        - **Your answer :**

- Please submit the Source code (zip all .c, .h, and makeFiles)

Important Note:

- **LATE SUBMISSIONS for labs**. There is a deduction of 10% for Late assignment submissions, and after three days it will grade of zero (0).
- This labs should be submitted along with a video-recording which contains a detailed walkthrough of solution. Without recording, the assignment can get a maximum of 1/3 of the total.
    - Note: In case you are running out of time to record the video, you can submit the assignment (source code + screenshots) by the deadline and submit the video within 24 hours after the deadline.