UNX511 Lab 8: Client/Server with Messaging

Due: Sunday, July 27, 2025 (11:59pm)

In this lab you will create a server (**server.cpp**) that communicates with three clients (**client1.cpp**, **client2.cpp** and **client3.cpp**) using one message queue. The client code for all three clients has been given to you along with the Makefile, a start-up script, and a stop script in case you have to manually stop the server and all the clients. You can retrieve all of these from the following links:
client1.cpp,
client2.cpp,
client3.cpp,
client.h,
Makefile,
startClient.sh,
stop.sh.

Your server and clients will be passing messages as defined in the structure **Message** as seen in **client.h**:

```
const int BUF_LEN=64;

// structure for message queue
typedef struct mesg_buffer {
   long source;
   long dest;
   char buf[BUF_LEN];
} MesgBuffer;

typedef struct mymsg {
   long mtype; /* Message type */
   MesgBuffer msgBuf;
} Message;
```

Your server will receive on message type (**mtype**) 4 but will transmit on one of three message types depending on the destination of the message. If the message is for client 1, then the server sets the message type to 1. If the message is for client 2, then the message type is set to 2. If the message is for client 3, then the message type is set to 3. This means client 1 filters all messages of message type 1, client 2 of message type 2, and client 3 of message type 3.

Your message buffer (**msgBuf**) will consist of a client source (**source**), client destination (**dest**), and a message (**buf[]**). The client source is the client that is sending the message. The client destination is the client which is receiving the message.

The server will act as a dispatcher of the messages. That is, all clients will send their messages to the server for distribution. When a server receives a message, it must eventually extract the destination of the message and send it to the destination client.

The key for communication between server and clients is as follows:
Server/Client1, Client2, Client3: key=ftok("serverclient", 65);

Study the client code to see how message reads and writes are performed. Reads are performed in a read thread (**recv_func**) and writes are performed from **main()**. Both reads and writes are performed within an infinite **while-loop** with **is_running** as the condition flag in both cases. Note that the messages are queued and mutex protected. Note also that your clients perform a controlled shutdown on ctrl-C.

For your server, the requirements are as follows:

- Your server must perform a controlled shutdown on ctrl-C.
- Your server must store the key for communication with all clients as specified above and as follows:
    - **Server/Client1, Client2, Client3: key=ftok("serverclient", 65);**
- The server should have one message queue for all clients:
    - **queue<Message> message;**
- The server should protect the message queue with a mutex:
    - **pthread_mutex_t lock_x;**
- The server should have one receive thread where it receives messages from all clients:
    - **void *recv_func(void *arg);**
- The server can perform writes in **main()** or in a separate write thread if you wish.
- The server's receive thread will simply push the message into a message queue. Be sure to protect the message queue with mutexes since it will be used in both the read and the write.
- The server's message write will pop a message from the message queue, extract the destination client from the message and forward the message to that destination client. Be sure to mutex the message queue when popping.
- On ctrl-C, the server's condition flag **is_running** will be set to false, and both the reading and writing will quit. When the write loop exits, be sure to send a "Quit" message to each client and then destroy the queue. Be sure also to join the read thread with **main()** before exiting **main()**.

In testing your work, it might help to isolate each client. For instance, if you want to see messages received by client3, use the following from the command line to start the clients:
$ ./startClient.sh | grep 'client 3:'

Questions:
Compare and contrast the mechanisms of inter-process communication that you have studied so far.

1. Between sockets, pipes, fifos, and messages, which is your favorite and why?
2. Which is your least favorite and why?
3. For the scenario presented in this lab, is there a need for a server? 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-Lab8
* 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.