CMPT 300 D200
Assignment 3
Our new chatbot – Lets-talk
Marks: 100 marks

Notes:

1. You can do this assignment individually or in a team of two. If you are doing it in a group, then only one submission per group is required.
2. You may submit multiple times until the deadline. Grade penalties will be imposed for late submissions (see the course outline for the details).
3. Always plan before coding.
4. All the codes in this lab must be done using C language only. No other languages should not be used.
5. Use function-level and inline comments throughout your code. We will not be specifically grading documentation. However, remember that you will not be able to comment on your code unless sufficiently documented. Take the time to document your code as you develop it properly.
6. We will be carefully analyzing the code submitted to look for plagiarism signs, so please do not do it! If you are unsure about what is allowed, please talk to an instructor or a TA.

Lets-talk

Background

For this assignment we are going to create a simple "chat"-like facility that enables a user at one terminal to communicate with a user at another terminal. We are not expecting a pretty interface but, it should be functional.

For this assignment, you will be using two main LINUX concepts.

1. Threads - creating a threads list and assigning tasks to them.
2. UDP - Check the following man pages for help with UDP:
    - socket
    - bind
    - sendto
    - recvfrom
    - getaddrinfo

There are a couple of good webpages that we will point you to:

- Socket programming: http://beej.us/guide/bgnet/
- Socket programming: https://habibiefaried.medium.com/cnet-01-basic-c-socket-programming-a818b0da5ae0
- Pthreads documentation: https://computing.llnl.gov/tutorials/pthreads/
- Pthreads : https://medium.com/@jithmisha/a-brief-intro-to-shared-memory-programming-with-posix-threads-a663b590e38c

This assignment will be done using Pthreads, a kernel-level thread implementation for LINUX. Pthreads allows you to create any number of threads inside one UNIX process. All threads running in the same UNIX process share memory (which means pointers valid for one thread are valid in another thread which is not possible between processes) and also have access to semaphores (mutexes) and the ability to use conditional signal/wait to synchronize their actions in relation to each other. UNIX itself also allows you to create multiple processes. Communication between UNIX processes can be done using something called "datagram sockets" which use a protocol called UDP (universal datagram protocol).

In this assignment, you will be dealing with processes/threads on two levels. You will have two LINUX processes (might not be on the same machine). Each one is started by one of the people who want to talk (by executing lets-talk).

Required threads (in each of the processes):

1. One of the threads awaits input from the keyboard.
2. The other thread awaits a UDP datagram from another process.
3. There will also be a thread which prints messages(sent and received) to the screen.
4. Finally, a thread which sends data to the remote UNIX process over the network using UDP(use localhost ip(127.0.0.1) while testing it on the same machine with two terminals).

All four threads will share access to a list ADT (check the required files for the code).

- The keyboard input thread, on receipt of input, adds the input to the list of messages that need to be sent to the remote lets-talk client.
- The UDP output thread will take each message off this list and send it over the network to the remote client.
- The UDP input thread, on receipt of input from the remote lets-talk client, will put the message onto the list of messages that need to be printed to the local screen.
- The screen output thread will take each message off this list and output it to the screen.

How the end product should look like?
You will be left with one executable file (lets-talk) after compiling the project. It should take 3 parameters to start.

1. IP address of remote client.

2. Port number on which remote process is running on.
3. Port number your process will listen on for incoming UDP packets.

Say that Alice and Bob want to talk. Alice is on machine "machine1" and will use port number 6060. Bob is on machine "machine2" and will use port number 6001.

To initiate lets-talk, Alice must type:
lets-talk 6060 machine2 6001

And Bob must type:
lets-talk 6001 machine1 6060.

The general format is:

lets-talk [my port number] [remote machine name] [remote port number]

Assignment requirements

Below are the requirements that must be implemented.

1. Require makefile to compile code and provide single executable.
    - Intermediate files should be cleaned.
    - "make" command should generate executable.
    - "make valgrind" should start valgrind for memory leak check.

2. Use list to communicate between threads.
    - For ex- receiver thread will put message in list and printer thread remove msg from list and prints it on screen
    - Implement the list with mutex such that the list should be accessible by only one thread at a time.

3. Encryption (use cipher encryption and description)
    - Use a fixed Encryption key
    - Increment each character by key before sending (take care about character size,take modulo by 256)
    - Decrement each character at receiver side by key.

Note - In the above encryption method, it can happen that a character in the string translates to null character in encrypted string. That will cause the termination of the message. This happens because in LINUX implementation of UDP, it terminates the message sending as soon as it finds null character or new line character.

4. Checking other 'online' status
    - !status should print Online/Offline if user on another end is online/offline

5. "!exit" this will quit the connection, handling this in string provided by user. The !exit should be case sensitive.
    - Program should be terminated on both the users if any of them types this command

6. Check for bind fail (socket bind function).

7. Require support for one lets-talk talking to itself (talk on localhost).
    - Should work on localhost(127.0.0.1)

8. Require support for copy-paste up to any number of lines of text, each line any length characters long.

9. Require support for very long lines (up to 4k characters)

10. Require no busy waits and not take up 100% CPU usage when nothing to do.
    - Avoid deadlock using mutex locks

11. Require no extra outputs when user typing in text (no prefix to sending, or receiving messages).

12. Require checking return values on all socket functions only

13. Require calling fflush(stdout) after each message received.

14. Require must pass Valgrind cleanly:

    * OK to have unfreed memory which comes from pthread_cancel_init
    * Everything else must be freed.
    * No uninitialized memory to syscall errors.
    * Use this if you don't know how to use valgrind.

To help you to get started, below are the steps to implement the chatbot.

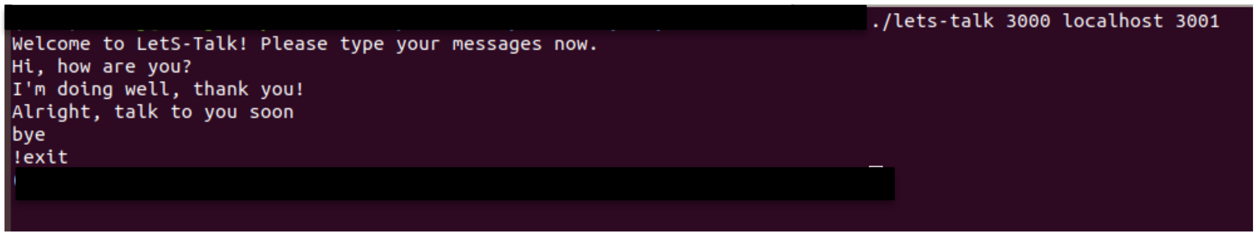
1. Create 4 threads as mentioned above in the main program.
2. Create two lists
    - First list will be used by keyboard thread and sender thread
    - Keyboard thread will get input from keyboard and put message in the list
    - Sender thread will get message from list and send it to another client
    - Second list will be used by receiver thread and printer thread
    - Receiver thread will put message got from other client in the list

- Printer thread will remove messages from list and print on monitor

3. Main thread will wait until any of them gives a termination signal, as soon as the main thread gets a termination signal it cancels all threads one by one.

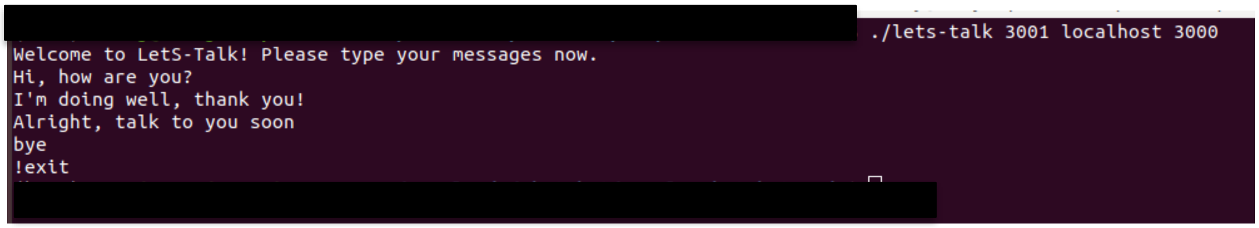
Sample output:

User at terminal 1:



```
[redacted] ./lets-talk 3000 localhost 3001
Welcome to LetS-Talk! Please type your messages now.
Hi, how are you?
I'm doing well, thank you!
Alright, talk to you soon
bye
!exit
[redacted]
```

User at terminal 2:



```
[redacted] ./lets-talk 3001 localhost 3000
Welcome to LetS-Talk! Please type your messages now.
Hi, how are you?
I'm doing well, thank you!
Alright, talk to you soon
bye
!exit
[redacted]
```

Question 2. [2 marks]
For each question, please specify how long you spend on it and what you learned. Any extra feature you would like to add to the chatbot? (This question is worth marks, so please do it!)

Test Cases

- Run the !exit command to validate that the connection is terminated for both users by running it on either user's side
Note: The !exit should be case sensitive
- Run the "!status" on either side to validate if the other user is online or offline
- Send messages from either terminal and validate if the message is received on the other terminal.
- The messages must be transmitted between both users only when both users are online.
- Since there is multithreading involved, you need to use mutex locks effectively. Validate to make sure that multiple threads aren't accessing the same list simultaneously.
Note: You can do this by checking mutex before accessing the list functions.

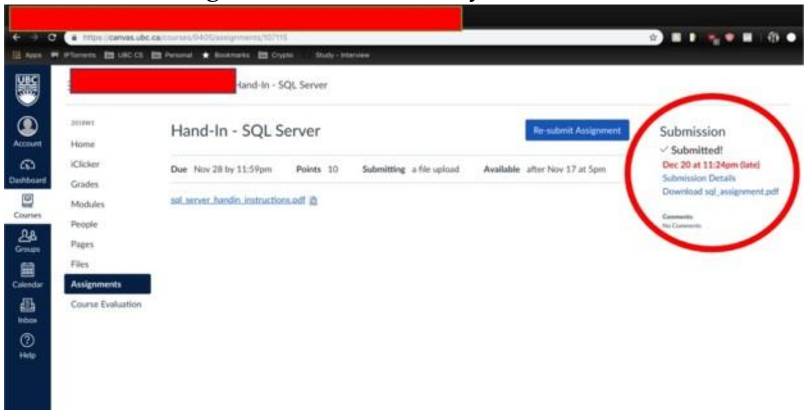
Submission Instructions:

Submit a ZIP file of your assignment folder. Your submission is composed of the following

- **Source code**
    - Include your source code file(s) (.h and .c)
-
- **Makefile**: the Makefile provides the following functionality:
    - **all**: compiles your program (this is the default behaviour), producing an executable file named same as the C file.
    - **clean**: deletes the executable file and any intermediate files (.o, specifically)

- **README.txt**: documentation
    - Include your name (and team member name) and assignment number at the top of the file
    - Write down any resources that you consulted (URLs).
    - Write down the name of the class members with whom you have discussed your solution.
    - Mention the claimed late days for the assignment.
    - The answer to Question 2.

After submitting the file on canvas, you should be able to see the below message:



[redacted]

[redacted] Hand-In - SQL Server

Hand-In - SQL Server

Due Nov 28 by 11:59pm Points 10 Submitting a file upload Available after Nov 17 at 5pm

sql_server_handin_instructions.pdf

Submission
✓ Submitted!
Dec 20 at 11:24pm (late)
Submission Details
Download sql_assignment.pdf

Comments
No Comments

Grading Criteria

- Correctness (96%): passes all of the (hidden) unit tests
    - There should only be 4 threads in the lets-talk process
    - You must use mutex locks to protect against simultaneous access of the lists of different threads
    - You need to a do a cleanup upon exiting
- Makefile (2%): satisfies the make file requirements
- README.txt (2%): contains the required information