ECEC 353: Systems Programming
Programming Project: Chat Server System

Instructor: Prof. Naga Kandasamy
ECE Department
Drexel University

January 28, 2020

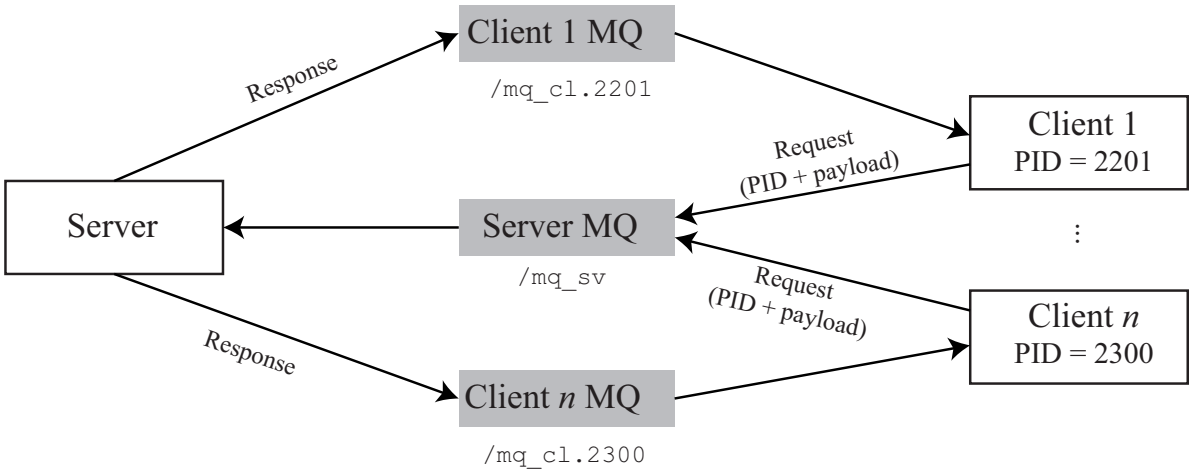
The project is due by 11:59 pm, February 16, 2020, via BBLearn. You may work on the project in a team of up to two people. If you are submitting as a group, please see the submission instructions later on in the document. You may discuss high-level concepts with other students but must write your own code.

Your code must be clearly written, properly formatted, and well commented for full credit. A good synopsis of the classic book, *Elements of Programming Style* by Kernighan and Plauger, is available at `en.wikipedia.org/wiki/The_Elements_of_Programming_Style`. Another good reference is Rob Pike's notes on *Programming in C*, available at `www.maultech.com/chrislott/resources/cstyle/pikestyle.html`. Alternatively you may follow the style that I use in the code examples provided to you.

Requirements

Develop a chat application consisting of multiple chat clients and a chat server. The chat application allows multiple chat clients to connect to a chat server. Once connected, clients can exchange messages with other clients that are connected to the server. The server has to accept and maintain connections to all active clients and relay chat messages between them.

The figure below shows a possible software architecture for the chat application, in which clients send their messages to a single message queue maintained by the server. The server posts broadcast or private message to individual message queues maintained by each client.



Following are the specifications for the server and client code:

- *Chat Client:* When the client program is executed, it attempts to connect to the server process. If the server is not running, the client application prints an error message to the user and exits. Once the client is connected to the server, it must: (1) transmit messages typed into the console to the server and (2) display messages that are sent to it by the server.
- *Chat Server:* The server must allow for a maximum of 100 users to enter and leave the chat system at any time. When a client message is received by the server, it must broadcast that message to the rest of the clients connected to it. Also, it should be clear to each client exactly who typed what. In other words, the broadcast message must also contain the identity of the originating client. Enhance the chat server to also allow private one-on-one communication between any two clients at any time. For example, consider two clients A and B. The public messages sent by these clients are broadcast to the rest of the group. However, if client A wants to exchange one-on-one private messages with client B, then your chat server must allow this.

Here are some specific implementation requirements:

- The client and server processes are unrelated. You *must* use *message queues* for inter-process communication as well as any appropriate synchronization mechanisms.
- You must find a solution to juggle multiple, potentially blocking descriptors within the client — since the client must accept keyboard input from the user while also displaying any messages sent to it by the server.

You have been provided with templates for the server and client programs. Please edit these files to complete the functionality of the chat system. You may add any number of additional functions and data structures that you deem necessary.

The server can be executed in the background as follows.

```
$ ./chat_server &
```

The client program accepts the user name as a command-line argument, connects to the server, and stays in a loop displaying the main menu as follows.

```
./chat_client nk78
User nk78 connecting to server

'B'roadcast message
'P'rivate message
'E'xit
```

The menu displays options for broadcasting messages to the group, sending private messages, and for exiting the chat rom.

Testing Scenarios and Grading Rubric

We will set up a test environment on *xunil* in which multiple clients, each running in a separate terminal, will connect to the server, running in its own terminal, and exchange messages between one another. Your chat application will be graded on the following features:

- **Basic Functionality**
  - (10 points) Ability for clients to dynamically enter and leave the chat room at various time intervals.
  - (10 points) Correct implementation of the broadcast feature between clients.
  - (10 points) Correct implementation of private messages between clients.
  - (5 points) Proper error handling in the case in which a client tries to connect to a server that does not exist. In this case, the client must shut down gracefully.
- **Extended Functionality for extra credit**
  - (10 points) Proper error handling in the case of a server that shuts down or crashes in the middle of a chat session. The client must detect this and shut down gracefully. (Hint: consider the use of "heartbeat" messages where the server periodically — for example, once every five seconds — posts these messages to the client message queue. If the client does not receive such a heartbeat message before the timeout, it may conclude that the server has crashed. Please read through the code listed within *simple_alarm.c* for ideas of how to generate periodic signals.)
  - (5 points) Proper error handling in case a server tries to write to a client that has crashed in the middle of a chat session.

Submitting Your Project

Once you have implemented all of the required features described in this document submit your code by doing the following:

- If you are using a make file, run `make clean` in your source directory. We must be able to built your server and client code from source and we don't want your precompiled executables of intermediate object files. If your code does not at the very least compile, you will receive a zero.
- Zip up your code, including complete instructions in a separate README file on how to compile and run your program on the *xunil* cluster.
- Name the zip file *abc123_chat.zip*, where *abc123* is your Drexel ID.
- Upload your zip file using the BBLearn submission link found on the course website.
- In addition, in a separate report, please include a brief description of the software architecture of the client and server processes as well as the key data structures used in these programs. The discussion should also include the design challenges faced and how you addressed them. Submit this report as a PDF document; limit your discussion to under ten pages.
- If you are working as a group of two, one submission will suffice. Please indicate the two authors in the cover page of your report.

Failure to follow these simple steps will result in your project not being graded.

Now, put on some music and have fun!