CSci 4061: Introduction to Operating Systems

Programming project 4 **due: Thursday April 18, 2019**

Ground Rules. You may choose to complete this project in a group of up to three students. Each group should turn in one copy with the names of all group members on it. The code must be originally written by your group. No code from outside the course texts and slides may be used—your code cannot be copied or derived from the Web, from past offerings, other students, programmer friends, etc. All submissions must compile and run on any CSE Labs machine located in KH 4-250. A zip file should be submitted through Canvas by 11:59pm on Thursday, April 18th. **Note**: Do not publicize this assignment or your answer to the Internet, e.g., public GitHub repo.

Objectives: The main focus of this project is to implement inter-process communication mechanisms in a server-client setting.

1 Project Description

Write a server program and two client programs so that the server can communicate privately to each client. There should also be a way to broadcast a message to both clients. You have to use message queues for this communication. To access a message queue, two processes need to use a common key. Use shared memory for passing the common key. For instance, the server stores the key in a shared memory segment. The client program reads the key from the shared memory segment. (Hint: use `shmget`, `shmat`, `shmdt` functions.)

Detailed description of the project is as follows:

1.1 From the server side

At the beginning, your server program should display a menu with options, like:

Enter 1 to choose client 1
2 to choose client 2
3 to broadcast a message
0 to exit

- If the user presses 1, then the server asks the user to enter a message. Then it sends the message to only client 1. When client 1 receives a private message from the server, it displays the message on the screen and asks the user to enter a response message. Then client 1 sends the response to the server. Next, the server again shows the menu to the user and asks for the choice.
- If the user enters 2, then communication proceeds in the same way with client 2 as described for client 1.
- If the user presses 3, then the server asks the user to enter a message for broadcasting. Then the server sends the message to both clients. When the clients receive a broadcast message,

they do not respond on that message unlike previous cases. The server again displays the menu to the user and asks to enter choice.

- If the user presses 0, the server closes the connection and terminates. The client programs should also terminate at this point.

1.2 From the client side

At the beginning, the client waits for a message from the server.

- If it receives a private message, it displays the message on screen and asks the user to enter a response message and sends it to the server. It then again waits for a message from the server.
- If it receives a broadcast message, it displays the message on screen and again waits for a message from the server.
- If the server closes connection, the client program displays the message, Server has closed connection. and terminates.

Note: You should use **message queues** for transferring messages but use **shared memory** for exchanging the key.

2 Deliverables

Students should upload to Canvas a zip file containing their C codes (server.c, client1.c and client2.c), a makefile, and a README that includes the group member names and student ids, what each member contributed, any known bugs, test cases used (we will also use our own), whether the extra credit has been attempted, and any special instructions for running the code.

3 Extra Credit

Pass the common key of the message queues using realtime signals instead of shared memory. Note that for the main project, you used shared memory for this purpose. If you do extra credit, you have to share the key using realtime signals, do not omit your code where you used shared memory. Instead, you should enable the realtime signal–based key exchange when the command-line argument, `-rt`, is specified.

4 Grading Rubric

- 5% Correct README contents
- 5% Code quality such as using descriptive variable names and comments
- 10% Using shared memory to pass the key
- 30% Implementation of private communication

- 20% Implementation of broadcast communication
- 10% Client being able to distinguish between private and broadcast message (for example, on receiving a private message, client asks user to type a response but on receiving broadcast message, it does not.)
- 10% Being able to terminate both server and client programs when server closes connection (server side user enters 0)
- 10% Error checking for all system calls
- 5% Extra credit