CECS 326-01 Assignment 2 (10 points)
Due: 10/4/2022 on BeachBoard

When a user launches a program for execution, the OS creates a user process to execute it. Many OS, including Linux, provides the mechanism for a process to create child processes to run concurrently with the parent process which creates them. Linux provides such support with system calls **fork**, **exec and wait**. These concurrent processes may need to communicate among them. One way Linux supports interprocess communication is message queue. Using the System V implementation, a message queue must first be acquired from the operating system by calling **msgget**. Control operations, e.g., remove, can be performed on an existing message queue by calling **msgctl**. Processes with appropriate permissions may send and/or receive messages via the message queue by calling **msgsnd** and **msgrcv**. Please consult the man pages of these system calls for details.

In this assignment you will make use of what you learned about the Linux message queue mechanism in class, and make use of the fork(), exec() and wait() system calls to create child processes and control the child process' execution. For this assignment you need to write three C++ programs named *master.cc* (source program file names may have .cpp suffix), *sender.cc*, and *receiver.cc*, which should be compiled into executables *master*, *sender*, and *receiver*, respectively. Be sure that *sender.cc* and *receiver.cc* are themselves C++ programs with main functions, NOT classes to be included in *master.cc*. Together they should do the following:

When *master* executes, it should first output a message to identify itself. It should then acquire a message queue from the kernel, followed by creating two child processes, with one to execute *sender* and the other to execute *receiver*, and pass to them the message queue id. The *master* process should output its process ID, the message queue ID, the process IDs of the *sender* and *receiver* processes it has created, then waits for both child processes to terminate. Then *master* will remove the message queue and then exit.

master should produce the following sequence of output:

Master, PID xxxxx, begins execution
Master acquired a message queue, id xxxxx
Master created a child process with PID xxxxx to execute sender
Master created a child process with PID xxxxx to execute receiver
Master waits for both child processes to terminate
Master received termination signals from both child processes, removed message queue, and terminates

When *sender* executes, it should first output a message to identify itself, and show the message queue id it obtained from the exec() system call that invokes its execution. It should then prompt user for a line of input, send the input line to *receiver* via the message queue created by *master*. It should then wait for an acknowledgement from *receiver* of the receipt of message, then terminate.

sender should produce the following sequence of output:

Sender, PID xxxxx, begins execution
Sender received message queue id xxxxx through commandline parameter
Sender: Please input your message
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx [*this is user input message*]
Sender sent message to message queue
Sender receives acknowledgement of receipt of message
Sender terminates

When *receiver* executes, it should first output a message to identify itself, and show the message queue id it obtained from the exec() system call that invokes its execution. It then retrieves a message from the message queue, sends an acknowledgement to *sender*, and outputs the message on the screen.

receiver should produce the following sequence of output:

Receiver, PID xxxxx, begins execution
Receiver received message queue id xxxxx through commandline parameter
Receiver acknowledged receipt of message
Receiver: Retrieved the following message from message queue
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx [*this is message retrieved from message queue*]
Receiver terminates

It should be noted that the display of the above three sets of output may interleave.

The program must run successfully on Linux.

Do the following for this assignment:

1. Develop three C++ programs (*master.cc*, *sender.cc*, and *receiver.cc*) as described above. Make sure your programs are properly formatted as well as adequately and clearly commented. Detailed explanations on all system calls are required and must be in your own words. Simply copying those from the man pages are not acceptable.
2. Submit on BeachBoard the three C++ programs, a screenshot that shows successful compile of all three programs as well as a successful run, and a cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear program description detailing what the programs are about. Format of the cover page should follow the cover page template posted on BeachBoard. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on <u>all</u> system calls are especially needed.