CECS 326-01 Assignment 2 (10 points)

Due: 10/5/2023 online on Canvas

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**. A process may obtain its own PID by calling **getpid**. 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.cpp*, *sender.cpp*, and *receiver.cpp*, which should be compiled into executables *master*, *sender*, and *receiver*, respectively. Be sure that *sender.cpp* and *receiver.cpp* are themselves C++ programs with main functions, NOT classes to be included in *master.cpp*. 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 a number of child processes to execute *sender* and the same number to execute *receiver*, and pass to them the message queue id. The number of senders and receivers would be as obtained in the commandline argument in the command that launches *master*’s execution. The *master* process should output its process ID, the message queue ID, the number of child processes it has created to serve as senders and the number of child processes it has created to serve as receivers, then waits for all child processes to terminate. Then *master* will remove the message queue and then exit.

*master* should run with the command below and produce the following sequence of output:

./*master x* [*x* is the number of senders as well as the number of receivers]

Master, PID xxxxx, begins execution

Master acquired a message queue, id zzzzz

Master created *x* child processes to serve as sender [*x* is a number from the aommandline argument]

Master created *x* child processes to serve as receiver [the same *x* from the aommandline argument]

Master waits for all child processes to terminate

Master received termination signals from all 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 the message queue created by *master*. It should then wait for an acknowledgement from *receiver* of the receipt of message, then terminate.

Each *sender* should produce the following sequence of output when it executes:

Sender, PID xxxxx, begins execution

Sender with PID xxxxx received message queue id zzzzz through commandline argument

Sender with PID xxxxx: Please input your message

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx [*this is* *user input message*]

Sender with PID xxxxx sent the following message into message queue:

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Sender with PID xxxxx receives the following acknowledgement message

yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy [*message sender retrieves from message queue*]

Sender with PID xxxxx 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 sent by a sender from the message queue and outputs the message, then sends an acknowledgement message into the message queue before terminating.

Each *receiver* should produce the following sequence of output when it executes:

Receiver, PID yyyyy, begins execution

Receiver with PID yyyyy received message queue id zzzzz through commandline argument

Receiver with PID yyyyy retrieved the following message from message queue

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx [*this is* *message retrieved from message queue*]

Receiver with PID yyyyy sent the following acknowledgement message into message queue

Receiver with PID yyyyy acknowledges receipt of message [*this line will be displayed by the sender that gets it*]

Receiver with PID yyyyy terminates

Notes:

1. The display of the above sets of output may interleave.
2. You may want to experiment with 1 sender and 1 receiver initially.
3. To enable the distinction between a message sent into the message queue by a sender and an acknowledgement message sent by a receiver, message tags should be used.
4. The program must run successfully on Linux.

**Submit on Convas** the following:

1. The source programs *master.cpp*, *sender.cpp*, and *receiver.cpp*;
2. A screenshot that shows successful compilation of *sender.cpp* to *sender*, *receiver.cpp* to *receiver*; and *master.cpp* to *master*;
3. A screenshot that shows successful run of *master*, which will create child processes to execute *sender* and *receiver*; producing outputs as described above;
4. 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 Canvas.
5. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.