ECE 2420 Programming Exercise #4a

(Processes and IPC)

# Overview

This programming exercise utilizes various multi-processing and inter-process communication mechanisms. The purpose of this code is to emulate a distributed system in which one process composes problems to be solved. These problems are passed to the second process which solves the problem. This second process then passes the result to a third process which consumes the result.

# Produce and Consumer Architecture

This PEX uses message queues to pass data between processes. This exercise is broken down into two phases. The first phase of consists of only two processes where the producing process and the consuming process are the same. The second phase separates the producing and consuming processes. The architecture of each are shown below:

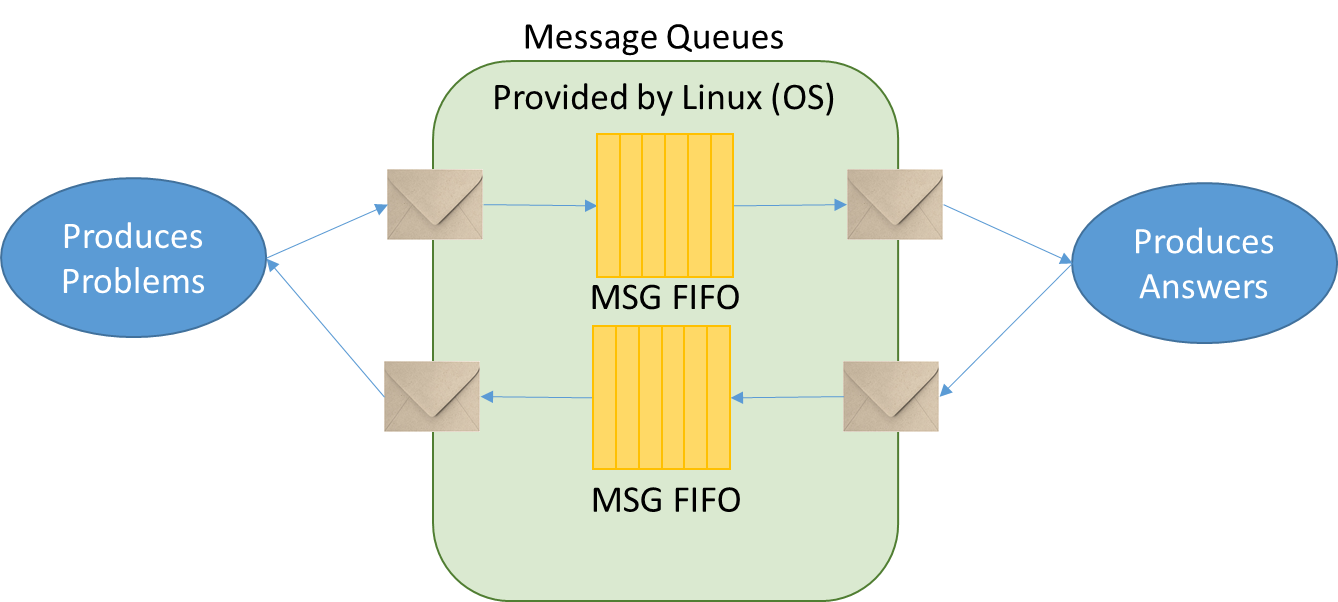


Figure 1. Message Queue Phase 1

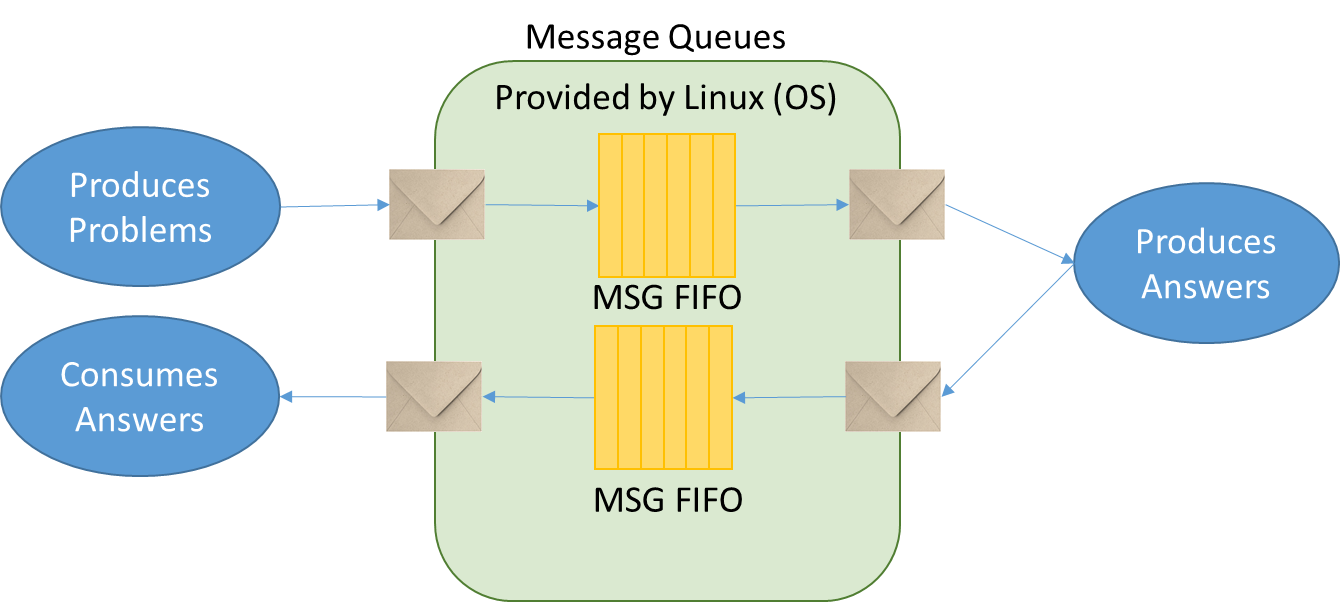


Figure 2. Message Queue Phase 2

Regardless of which phase is being developed, a producing process will randomly generate a trivial math problem. This problem is stored in the C structure called “Problem” in the provided header. The problem is then placed in a message queue where another process will execute the problem and produce an “Answer” object. The answer is then placed in another queue where either the originating process or a third process can consume the result.

# Programming Concepts

This exercise covers many programming concepts including processes, fork, wait, named message queues, random number generation, and producer consumer architectures.

# System Requirements

The design must use the provided header verbatim. All inter process communication primitives should be named as specified in the provided header. This will allow automated testing of the design you produce.

Here are addition specific requirements that shall be implemented/followed:

1. The service process shall create all needed queues
2. The service process shall initialize all IPC primitives to the proper initial state
3. The producing client process connects to IPC primitives created by the service
4. The producing client process creates Problem objects and passes them to the service process
5. The service process receives the Problem objects, executes the operation, and produces an Answer object
6. In the first phase, the Answer is passed back to the producing client process.
7. In the second phase, the Answer is passed back to a third client consumer process
8. In the first phase, the producing process will wait until it receives and processes an Answer before sending a new Problem
9. In the second phase, the producing process will create and queue Problems as quickly as the service can accept them
10. For each phase, you should provide the following analysis:
    1. Time for executing 1 iteration of ARRAY\_SIZE
    2. Time for executing 10 iterations of ARRAY\_SIZE
    3. Time for executing 100 iterations of ARRAY\_SIZE
    4. Time for executing 1000 iterations of ARRAY\_SIZE
    5. In prose, compare and contrast the performance of each and provide an explanation for your observed results

# Timing Results

Note that Linux has a command called ‘time’ that can be used to measure how long a process executes. For example, if you desired to time the ls command, you would issue the command ‘time ls’. Also included in this package is a framework for building a forked client which will spawn two processes. One of the processes will create the problems and the other will consume the answers. Using this framework will allow the use of the Linux time command to accurately measure the running time of both client processes. This framework is supplied in the file ForkedClient.cpp

# Turn-in Procedures

Turn in all source code and required analysis via git commit/push by 11:59p.m. on 8 October.

Grading Rubric

(ECE 2420 PEX4a)

|  |  |  |
| --- | --- | --- |
| Requirement / Criteria | Available Points | Student’s Score |
| Submitted via git push; contains makefile which builds correctly | 10 |  |
| Msg queue producer places Problems in queue | 10 |  |
| Msg queue service retrieves, executes and produces answers | 10 |  |
| In phase 1, Msg queue producer receives responses | 10 |  |
| In phase 2, Msg queue consumer receives messages and ARRAY\_SIZE problems are allowed “in flight.” Must use fork framework for accurate timing | 10 |  |
| Written analysis is complete and rational | 10 |  |
| Good coding practices are followed and corner cases handled | 10 |  |
| Total | **70** |  |