Lab 7.1 : IPC Mechanism using Named Pipe and Shared Memory

November 14, 2016

Objective : The purpose of this lab is to teach students IPC mechanisms and syncronization using semaphores.

You should refer to lecture_notes_IPC_mechanism.pdf and complete Lab 6.1 before proceeding with this lab

- Experiments with Semaphores

  1. Compile and run *read-write-3.c* with gcc. and run it. Review *read-write-3.c* to be sure you understand how it works.
  2. Use the idea of semaphores, as implemented in *read-write-3.c*, in place of the spinlocks in lab 6.1 to handle the synchronization of the reader and writer process from *read-write-3.c*.
  3. With this use of semaphores, explain whether your code in step 11 will work when there are multiple readers or when there are multiple writers. In each case, if the code would work, explain why. If the code would not work, give a timing sequence involving the several readers and/or writers showing what might go wrong.

- Experiments with Multiple Readers and Multiple Writers

  1. Compile and run *read-write-4.c* with gcc. and run it. Review *read-write-4.c* to be sure you understand how it works.
  2. This program contains a third semaphore, mutex. Explain the purpose of this semaphore. Specifically, if semaphore mutex were omitted, give a timing sequence involving the several readers and/or writers showing what might go wrong.
  3. Program *read-write-4.c* prevents any reader from working at the same time as any writer. Assuming that the buffer contains several locations, however, writing to one buffer location should not interfere with reading from another. That is, the critical section for readers need not be considered exactly the same as the critical section for writers. Remove semaphore mutex and add additional semaphores, as needed, so that some reader could work concurrently with some writer (assuming the buffer contained some data but was not full – so both reading and writing made sense).

- A Simple Pipeline Program:

  Consider the following problem: A program is to be written to print all numbers between 1 and 1000 (inclusive) that are not (evenly) divisible by either 2 or 3.

  This problem is to be solved using three processes (P0, P1, P2) and two one-integer buffers (B0 and B1) as follows:

  1. P0 is to generate the integers from 1 to 1000, and place them in B0 one at a time. After placing 1000 in the buffer, P0 places the sentinel 0 in the buffer, and terminates.
  2. P1 is to read successive integers from B0. If a value is not divisible by 2, the value is placed in B1. If the value is positive and divisible by 2, it is ignored. If the value is 0, 0 is placed in B1, and P1 terminates.
  3. P2 is to read successive integers from B1. If a value is not divisble by 3, it is printed. If the value is positive and divisible by 3, it is ignored. If the value is 0, P2 terminates.

  Write a program to implement P0, P1, and P2 as separate processes and B0 and B1 as separate pieces of shared memory – each the size of just one integer. Use semaphores to coordinate processing. Access to B0 should be independent of access to B1; for example, P0 could be writing into B0 while either P1 was writing into B1 or P2 was reading.