SIENA*college*
Computer Science

Computer Science 330
Operating Systems
Siena College
Fall 2020

Lab 5: Concurrency

Due: 11:00 AM, Monday, September 28, 2020

In this lab, you will consider another software solution to the critical section problem and write a program to simulate a bounded buffer problem with multiple producers and multiple consumers.

You may work alone or with a partner or two on this lab.

Learning goals:

1. To learn about Dekker's Algorithm, and see how it provides a solution to the critical section problem.
2. To work with POSIX threads and POSIX semaphores in a solution to a generalized bounded buffer problem.

Getting Set Up

You will receive an email with the link to follow to set up your GitHub repository, which will be named `concurrency-lab-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file before the end of class.

Dekker's Algorithm

We looked at Peterson's Algorithm in class as a software solution to the critical section problem for two processes. Another solution is due to Dekker:

- Shared data

```
int turn=0;
boolean flag[2];
flag[0]=false;
flag[1]=false;
```

- Process P_i

```
while (1) {
  flag[i]=true;
  while (flag[j]) {
    if (turn != i) {
      flag[i]=false;
      while (turn != i);
      flag[i]=true;
    }
  }

    /* critical section */

  turn=j;
  flag[i]=false;

    /* non-critical section */

}
```

? Question 1:
State the three conditions that must be guaranteed for an approach such as this to be considered a solution to the critical section problem (2 points)

? Question 2:
Explain, as precisely as you can, how each of the these conditions is satisfied by Dekker's Algorithm. (3 points)

Bounded Buffer with Semaphores

Write a C program that implements the bounded buffer problem for an arbitrary number of producers and consumers and items to be processed (each specified by command-line parameters). If i items are to be processed by p producers and c consumers, each producer should produce $\frac{i}{p}$ items and each consumer should consume $\frac{i}{c}$ items. You may either check that these divide evenly and report an error if not, or account for the uneven division by having some producers or consumers process one extra or one fewer item.

Use POSIX threads to create your producers and consumers and use POSIX semaphores for synchronization. Your solution should avoid the busy wait from the class examples (other than any busy waiting that might takes place inside a semaphore `wait` or `signal` operation, which is not your fault).

Write your program in a file called `prodcons.c`. You may (and should!) use the source code from any of the class examples as your starting point and/or to guide your solution. Be sure to indicate clearly which code you borrow and which code you add or modify.

See the examples and man pages for more information on POSIX threads and POSIX semaphores. Please don't hesitate to ask questions!

Submission

Commit and push!

Grading

This assignment will be graded out of 20 points.

Feature	Value	Score
Question 1	2	
Question 2	3	
Arbitrary numbers of producers/consumers/items	5	
Correct and efficient usage of POSIX semaphores	8	
Documentation	2	
Total	20	