Producer-Consumer Problem

CIS 3207 Spring 2014 Lab Project 3

In this assignment, you are going to implement a solution to the producer-consumer problem. This assignment will help you understand issues in synchronization of concurrent threads and processes. The general form of a high level algorithm that can be used to implement a producer-consumer program is given later in this document.

Outline of the problem:

1. The problem consists of many (minimum of 4) producer threads and many (minimum of 3) consumer threads sharing a (memory) buffer with multiple (minimum of 20) slots (storage cells); each producer deposits one element (of whatever item each produces) into one slot of the buffer each time it "produces", and each consumer extracts one element from the buffer each time it "consumes". The buffer is treated as a circular queue, with insertions into one end of the queue, and extractions from the queue from the other end.

2. You should use **mutex (binary semaphore) objects** to implement mutual exclusion between producers and between consumers. You need to ensure that two tasks cannot access the same buffer slots at the same time.

3. In addition, a producer task may not write to the buffer if it is full, and a consumer task may not read from the buffer if it is empty. These activities will require that the producer or consumer blocks, until the operation it requires can be completed. This means that a producer blocked on a full buffer can be released from its block because a subsequent consumption has emptied a slot, or a consumer that is blocked on an empty buffer can be released from its block because a producer subsequently deposits a value in the buffer. In order to implement these "block and release" capabilities, you should use **general semaphores** for synchronization.

4. Your program should save critical event information to a log file. You should design your program to log the event occurrences that can demonstrate that the program (producer and consumer threads) are executing correctly, that no task has interfered with the other task as it is manipulating the buffer, and that the buffer is not corrupted by the work of the threads. Part of your design is to consider possible error conditions and to dynamically test to ensure that you have avoided these conditions.

5. You will need to vary the speeds of the producer (consumer) depositing (fetching) elements, and demonstrate proper buffer storage and access as the intervals increase and decrease. [timers or timing loops can be used to implement speed control.]

6. We suggest that you begin your development by solving the one consumer, one producer problem with a shared multiple storage cell buffer and then build the full solution.

7. You will be implementing this solution in two different operating systems – Linux and Windows 7. This will give you experience with the two different development/test environments and differences in system calls between these two systems. I suggest that you begin with the high level algorithm and develop the rest of the algorithm/solution. Then look at the Linux and Windows system calls to determine what you will need to create/destroy threads, and create and use mutex objects and semaphores.

    a. You have been working with Linux for lab project 2. I suggest that you begin the project

in Linux and then implement the Windows solution. In Linux, you have the Pthreads library available to you.

b. You should implement the solution to the problem on a Windows system, using the library functions or objects provided by Microsoft. The suggested environment is Visual Studio and C/C++. MSDN is a very good source of information regarding synchronization objects and details of how to use them.

We have discussed both the single producer/ single consumer solution and the multiple producer / multiple consumer solution in class and looked at the high level algorithms for solution in both cases.

You can use this following 'high level' algorithm for your implementation of the **single producer/ single consumer version** --you can also design your own solution – and then build your multiple producer/ multiple consumer version from there:

```
full: status of the buffer, 1 is full, 0 is not;
empty: status of the buffer, 1 is empty, 0 is not;
mutex: binary semaphore


 producer:

        produce item m;
        P(mutex);
        if(full)
            :                   //handle wait for not full
        else
            insert data into buffer;
            :                   // if buffer was empty, release blocked consumer
        V(mutex);

consumer:
        P(mutex);
        if(empty)
            :                   //handle wait for not empty
        else
            extract data from buffer;
             :                  //if buffer was full, release blocked producer
        V(mutex);
        Consume item;
```

This Project is due: Friday, April 4 at the end of the day.