REPORT

Name: Chris Banci
Date: March 31, 2017
Course: CS433 - Operating Systems
Assignment: 4 - Multi-threaded Programing for the Producer-Consumer Problem

---
Description:

This program is an implementation of a solution to the producer-consumer problem when using multi-threaded programming. Also know as the bounded-buffer problem, it is a classic example of a multi-process synchronization problem where two processes try to access a shared resource and fail doing so because of the other process already using it. In this case, the producer and consumer are the two processes that are sharing a bounded circular buffer as its resource and needs to be synchronized in order to effectively insert and remove items from the buffer. The problem is to make sure that the producer does not insert items into the buffer when it is already full and also make sure that the consumer does not remove items from the buffer when it is already empty. To achieve this, semaphores can be used.

Lastly, in order to study the behavior of concurrent threads, this program keep track and prints out the content of the shared buffer when an item is produced or consumed.

Implementation:

This program takes in arguments from the command line to specify the number of producer and consumer threads to use in the producer consumer problem.

Circular Buffer

- Internally, the buffer is a fixed array that acts as a circular queue by using position markers that point to the buffer's first and last item. Inserting and removing items in the buffer update the buffers first and last item position with the help of the modulus operator.

Producer-Consumer

- The producer and consumer process uses two counting semaphores (full/empty) and a mutex lock to synchronize access to the bounded buffer.
- To insert an item into the buffer, the producer process must first wait for an available slots in the buffer by acquiring the &empty semaphore. Next, to prevent race conditions, access to the critical section must be restricted from other processes by locking the mutex. The producer can now insert an item in the buffer. Lastly, the mutex is unlocked and the &full semaphore is released.

- To remove an item from the buffer, the consumer process must first wait for the buffer to not be empty by acquiring the &full semaphore. Next, to prevent race conditions, access to the critical section must be restricted from other processes by locking the mutex. The consumer can now safely removed an item from the buffer. Lastly, the mutex is unlocked and the &empty semaphore is released.

Included Files:

Source:

- main.cpp // driver file & includes producer/consumer
- buffer.cpp // implementation file for buffer.

Header:

- buffer.h // header file for buffer.

Other:

- assign4 // the executable.
- makefile // used to compile the program into an executable.

How to run:

To compile this program, use the makefile which will compile the source files and create an executable called assign4.

To run the executable, enter ***./assign4 <X> <Y> <Z>*** in the console.

- ***<X>*** being the how long the main thread should sleep before terminating (in seconds).
- ***<Y>*** being the number of producer threads.
- ***<Z>*** being the number of consumer threads.

Example:

./assign4 10 5 5

This will run 5 producer threads, 5 consumer threads, and sleep for 10 seconds.