COSC 40203 - Operating Systems

Producer/Conusmer Problem with Faulty Threads

Spring 2022
Due: 15:30, April 12^{th}, 2022

Objective

The purpose of this programming project is to explore process synchronization. This will be accomplished by writing a simulation to the Producer / Consumer problem described below. Your simulation will be implemented using pthreads with semaphores and mutex locks. Tutorials on the pthread functions and their usage can be found in our text, in our notes, or online. This project is very similar to the project described in our textbook but has additional requirements.

The Producer/Consumer Problem

The producer consumer problem is a process synchronization problem that demonstrates cooperating threads. Producer threads will place an item into a common shared buffer. Consumer threads will remove an item from the buffer. In this project the buffer is "bounded"; i.e. it has a finite length to it. This could be implemented as a circular array or you could use one of the STL linear data structures (list, vector, or queue).

A producer thread can place an item into the buffer only if the buffer has a free memory location to store the item. A producer thread cannot add an item to a full buffer. A consumer thread can remove an item from the buffer only if the buffer is not empty. A consumer thread must wait to consume items if the buffer is empty.

The "items" stored in this buffer will be positive integers. Your producer process will "randomly find" a prime number (described below). The consumer process will have to "verify" that the number consumed from the buffer is a prime number.

Project Specifications

Write a program that simulates the producer/consumer problem using C/C++ and pthreads. The producer thread will alternate between finding a prime number and inserting it into the buffer. Prime numbers will be "randomly guessed". That is, the producer will use a random number generator to generate a random number [2..999999]. Then the producer will test if the random number is prime. If the number is a prime, the producer will (try to) insert it into the buffer.

Producer threads always insert prime numbers into the buffer. Faulty producers insert any random even number [4..999999] into the buffer.

Consumer threads will remove numbers from the buffer and test if the number is prime. If the number is not prime, a warning message will be displayed.

The main function will initialize the buffer and create the separate producer, faulty producer, and consumer threads. Once it has created the threads, the `main()` function will then wait for the producers to finish producing and the consumers to finish consuming. After joining all threads, the main thread will then display the simulation statistics. The `main()` function will be passed five required parameters on the command line and a sixth optional parameter:

1. The number of items to produce per producer thread (required) (-n)
2. The length of the buffer (required) (-l)
3. The number of producer threads (required) (-p)
4. The number of faulty producer threads (required) (-f)
5. The number of consumer threads (required) (-c)
6. Optional debug flag (optional) (-d)

Perform any and all necessary validation on the command line arguments. There are no default values for the command line arguments. There are C/C++ functions to help parse command line arguments received in any order. See the next section for an example of how the program will be run with command line arguments.

Program Output

Output for this simulation is critical to verify that your simulation program is working correctly. Use this sample output to determine what your simulation should output when various conditions occur (buffer empty/full, etc.) To run your program with a 10 items per producer, buffer length of 5, 2 producer threads, 1 faulty producer thread, 2 consumer threads, and showing the debug statements, run your program as follows. Your program output format should be (nearly) identical to the following:

```
linux$ osproj3 -n 10 -l 5 -p 2 -f 1 -c 2 -d
Starting Threads...

(PRODUCER  1 writes  1/10   37): (1): [  37   ]
(PR*D*C*R  1 writes  1/10   18): (2): [  37  18  ]
(CONSUMER  2 reads   1      37): (1): [  18  ]
(CONSUMER  1 reads   1      18): (0): [   ] *NOT PRIME* *BUFFER NOW EMPTY*

...SOME TIME GOES BY...

(CONSUMER  1 reads   12     97): (0): [   ] *BUFFER NOW EMPTY*

...SOME TIME GOES BY...

(PRODUCER  2 writes  9/10   17): (5): [  37   17   61   67   5  ] *BUFFER NOW FULL*
```

```
...SOME TIME GOES BY...

PRODUCER / CONSUMER SIMULATION COMPLETE
=======================================
Number of Items Per Producer Thread: 10
Size of Buffer: 5
Number of Producer Threads: 2
Number of Faulty Producer Threads: 1
Number of Consumer Threads: 2

Number of Times Buffer Became Full 6
Number of Times Buffer Became Empty 4

Number of Non-primes Detected 8
Total Number of Items Consumed: 30
  Thread 1: 22
  Thread 2: 8
etc...

Total Simulation Time: 0.500 seconds
linux$
```

Additional Notes

Creating pthreads using the pthreads API is discussed in our textbook, notes, and online. Please refer to those references for specific instructions regarding creation of the producer, consumer, and faulty producer pthreads. A suggested algorithm for the `main()` function is the following:

```
int main( int argc, char *argv[] )
{
   Get command line arguments
   Initialize buffer and simulation stats
   Create producer threads
   Create faulty producer threads
   Join Threads
   Display Statistics
   Exit
}
```

Assessment and Grading

This assignment may be completed by teams of two. Your program must be written using C or C++ and you are required to use the pthread and semaphore libraries. Comment and document all code submitted! Your program will be tested on lovelace.cs.tcu.edu. You may do development work on your personal machine but final submissions must compile without errors or warnings and execute without core dumping. Use good programming practices by implementing procedures and

functions where necessary. You may use the STL in your solution. This project is worth 100 points.

Project Deliverables

1. Follow the project submission guidelines.
2. Follow the project documentation standards.
3. Your project must have a Makefile and a README file.
4. Update your project using GitHub Classroom.