ICS 53, Winter 2020

Assignment 6: Producer-Consumer Problem

Introduction

The producer-consumer problem is a well-known problem in concurrent programming. In this assignment, you will implement an extension of this problem using the POSIX Threads library and you would need to synchronize executions of the threads to have the program working properly.

Project Specifications

We have two kinds of threads: producers and consumers. In this assignment there can be many producers and consumers. Each producer and each consumer must execute in a different thread. A producers' job is to produce items and put them into a bounded buffer. A consumer's job is to remove items from the buffer. The buffer can hold 8 items. There are the following constraints on how producers and consumers can access the buffer.

1. Only one thread can access the buffer at a time.
2. A producer cannot write into the buffer if it is full.
3. A consumer cannot read from the buffer if it is empty.

- Producer Behavior

- Each producer will have an associated producer number, **`n`**, which is an integer from `0` to `p-1`, where **`p`** is the number of producers.
- Each producer will produce a series of items and place each item in the buffer.
- Producers will not produce items in an infinite loop. Instead, each producer will produce only **`i`** items. After a producer thread has produced i items and placed them in the buffer, the producer thread will terminate.
- Each item produced is an integer between `(n * i)` and `((n+1) * i - 1)`. This ensures that the items produced by all producers are unique; no two producers ever produce the same item.
- When a producer produces an item, it prints "`producer_`***`n`*** `produced item` ***`item_value`***" . The term `n` in the printed output should be replaced with the

producer number, and the term `item_value` should be replaced with the item value.

- Consumer Behavior

- Each consumer will have an associated consumer number, **`m`**, which is an integer from `0` to `c-1`, where **`c`** is the number of consumers.
- Each consumer will remove items from the buffer.
- Consumers will not produce items in an infinite loop. Instead, each consumer will consume only `(p*i)/`**`c`** items. After a consumer thread has consumed `(p*i)/`**`c`** items, the consumer thread will terminate.
- When a consumer consumes an item, it prints "`consumer_`***`m`*** `consumed item` ***`item_value`***" . The term `m` in the printed output should be replaced with the consumer number, and the term `item_value` should be replaced with the item value.

- Delays

In order to demonstrate the behavior of the system under different conditions, you will add delays to the producer and consumer threads using the `usleep()` function. There will be two options for the insertion of delays. In the first option, a 0.5 second delay will be added to each producer thread immediately after it adds an item to the buffer. This will greatly slow down the rate of production. In the second option, a 0.5 second delay will be added to each consumer thread immediately after it removes an item from the buffer. The choice between these two delay options will be made by a command-line argument passed to your program.

- Command-Line Arguments

Your program will accept 4 command-line arguments which specify parameters of the producer-consumer system.

- **`p`** - The number of producers. This must be between 1 and 16.
- **`c`** - The number of consumers. This must be between 1 and 16. The number of consumers should always be smaller than the number of total items being produced (i.e. `c < p*i`).
- **`i`** - The number of items produced by each producer.

- **`d`** - The selection of the delay option. If d = 0 then the delay will be added to the producer threads,and if d = 1 then the delay will be added to the consumer threads.

Grading Criteria and Sample Output

Using pthread functions without specifying priorities, the order of lines in output may vary every time you run the code even with the same values.
Here are some criteria for grading:

- Your code must produce the correct number of threads and items.
- All threads should have an equal opportunity among threads of their kind to produce and consume items.
- Each item should be produced exactly once and consumed once as well.
- An item must not be consumed before its production.

Running code as `./a.out` *`p c i d`* your output should consist of producer_`0` to `p-1` each producing items for `i` times and consumer_`0` to `c-1` each consuming for `p*i/c`. Running your code with `d = 0` and `d = 1` you should see different behavior in your results (why?).

Here are some sample correct outputs:

```
$ ./a.out 2 4 10 0

producer_0 produced item 0
producer_0 produced item 1
producer_0 produced item 2
producer_0 produced item 3
producer_0 produced item 4
producer_0 produced item 5
producer_0 produced item 6
consumer_3 consumed item 0
producer_0 produced item 7
consumer_0 consumed item 1
consumer_2 consumed item 2
producer_0 produced item 8
producer_0 produced item 9
consumer_1 consumed item 3
```

```
producer_1 produced item 10
consumer_3 consumed item 4
consumer_2 consumed item 5
producer_1 produced item 11
producer_1 produced item 12
consumer_0 consumed item 6
consumer_1 consumed item 7
producer_1 produced item 13
producer_1 produced item 14
consumer_3 consumed item 8
consumer_2 consumed item 9
producer_1 produced item 15
producer_1 produced item 16
consumer_1 consumed item 10
consumer_0 consumed item 11
producer_1 produced item 17
producer_1 produced item 18
consumer_3 consumed item 12
consumer_2 consumed item 13
consumer_1 consumed item 14
producer_1 produced item 19
consumer_0 consumed item 15
consumer_3 consumed item 16
consumer_1 consumed item 17
consumer_2 consumed item 18
consumer_0 consumed item 19
```

```
$ ./a.out 2 4 10 1

producer_0 produced item 0
consumer_3 consumed item 0
producer_1 produced item 10
consumer_3 consumed item 10
producer_0 produced item 1
consumer_3 consumed item 1
producer_1 produced item 11
```

```
consumer_1 consumed item 11
producer_0 produced item 2
consumer_1 consumed item 2
producer_1 produced item 12
consumer_1 consumed item 12
producer_0 produced item 3
consumer_1 consumed item 3
producer_1 produced item 13
consumer_1 consumed item 13
producer_0 produced item 4
consumer_2 consumed item 4
producer_1 produced item 14
consumer_2 consumed item 14
producer_0 produced item 5
consumer_2 consumed item 5
producer_1 produced item 15
consumer_2 consumed item 15
producer_0 produced item 6
consumer_2 consumed item 6
producer_1 produced item 16
consumer_3 consumed item 16
producer_0 produced item 7
consumer_3 consumed item 7
producer_1 produced item 17
consumer_0 consumed item 17
producer_0 produced item 8
consumer_0 consumed item 8
producer_1 produced item 18
consumer_0 consumed item 18
producer_0 produced item 9
consumer_0 consumed item 9
producer_1 produced item 19
consumer_0 consumed item 19
```

Submission

Your source code must be a single c file. Be sure that your program must compile on openlab.ics.uci.edu using gcc version 4.8.5 with the command "`gcc`

`${yourfilename.c} -lpthread`” and do not require any flags or modifications. (Note: check your gcc version with the “gcc -v” command)

Submissions will be done through Gradescope. The first line of your submitted file should be a comment which includes the name and ID number of you and your partner (if you are working with a partner).