PROGRAMMING ASSIGNMENT#3 (DUE BY 11:59PM JULY 28TH)

PURPOSE

This assignment intends to familiarize you using POSIX semaphores to solve the bounded-buffer problems (aka the classical producer-consumer problem.)

DESCRIPTION

The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a bounded buffer. Producers write data to the buffer and consumers read data from the buffer.

- Producers must block if the buffer is full.
- Consumers must block if the buffer is empty.

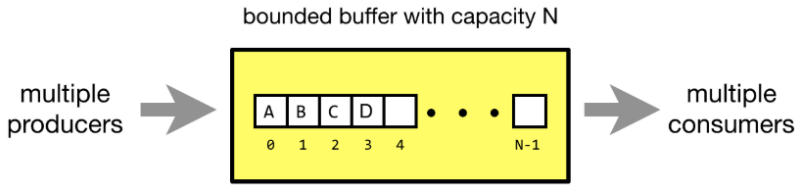

Figure 1: Bounded buffer

A bounded buffer with capacity N has can store N data items. The places used to store the data items inside the bounded buffer are called slots. Without proper synchronization the following errors may occur.

- The producers doesn't block when the buffer is full.
- A Consumer consumes an empty slot in the buffer.
- A consumer attempts to consume a slot that is only half-filled by a producer.
- Two producers writes into the same slot.
- Two consumers reads the same slot.
- And possibly more …

The buffer is represented by the following C struct.

```
Typedef struct
{
   int value[BUFFER_SIZE];
   int next_in, next_out;
} buffer_t;
```

In the struct there is an array `value` used to store integer values in the buffer and two integer indexes `next_in` and `next_out`. Index `next_in` is used to keep track of where to write the next data item to the buffer. Index `next_out` is used to keep track of from where to read the next data item from the buffer.

In the below example three data items B, C and D are currently in the buffer. On the next write data will be inserted to index `next_in = 4`. On the next read data will be removed from index `next_out = 1`. If `next_in/next_out` reaches `N-1`, it will be reset to `0,` i.e. rounded buffer.

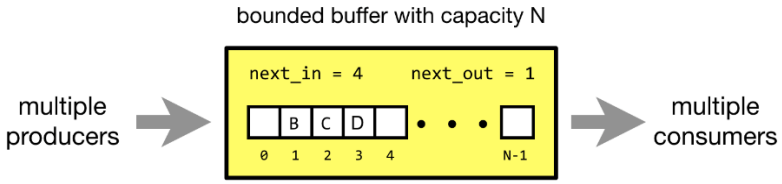

Figure 2. Bounded Buffer In-Out

There will be multiple producers and consumers share the bounded buffer. A producer will insert to the buffer and update `next_in` and a consumer will remove from the buffer slot `next out` and update it.

ASSIGNMENT

You will find an almost working implementation of the bounded buffer program in three.zip, which includes the following files:

```
bbuffer.h       // header file defines Marcos and etc.
bbuffer.c       // implementation of rounded buffer program
testbbuffer.c   // test program
makefile
readme      //   Design documentation, you need to answer the questions
                 included in this file to explain your design.
```

Your assignment is to write/modify code to complete the implementation. To be more specific, you need to use several semaphores to enforce proper synchronization between producers and consumers (and possibly some other critical regions also) to avoid race conditions.

The `testbbuffer.c` is used to test your program, so don't change this file.

It's worthy pointing out that:

1) The buffer is bounded by size BUFFER_SIZE defined in the header file,
2) Only one bounded buffer is shared by all the producers and consumers,
3) `next_in` always points to the slot a producer will insert the next new item into,

4) `next_out` always points to the slot a consumer will remove the item from.

TESTING

Your program must execute correctly on Edoras machine, the instructor will type the following commands to test your code:

```
make// generate testbbuffer executable file
./testbbuffer  // run the testbbuffer program (Note: the instructor may modify your source code to change the number of consumers/producers)
```

Your program should behave as follows (7 requirements):

1) When a producer thread inserts a new item into the buffer, a message should be print to screen showing which producer (with ID) insert which item (with value) to which buffer slot (with `next_in`),
2) When a consumer removes a new item from the buffer, a message should be print to screen showing which consumer (with ID) remove which item (with value) from which buffer slot (with `next_out`),
3) Producers must block if the buffer is full,
4) Consumers must block if the buffer is empty,
5) No two/more producers insert items to the same buffer slot,
6) No two/more consumers remove the items from the same buffer slot,
7) The messages must be printed in the order showing the real execution scenarios, here is the example:

```
consumer 0 started!
producer 0 started!
consumer 2 started!
consumer 1 started!
producer 1 started!
producer 2 started!
consumer 4 started!
producer 0: inserted item 3215 into buffer index 0
consumer 3 started!
producer 0: inserted item 7530 into buffer index 1
producer 0: inserted item 7392 into buffer index 2
consumer 1: removed item 3215 from buffer index 0
producer 1: inserted item 2897 into buffer index 3
producer 0: inserted item 6592 into buffer index 4
producer 2: inserted item 5249 into buffer index 0
```

```
consumer 2: removed item 7530 from buffer index 1
consumer 2: removed item 7392 from buffer index 2
consumer 2: removed item 2897 from buffer index 3
producer 1: inserted item 1706 into buffer index 1
producer 2: inserted item 4868 into buffer index 2
producer 0: inserted item 3293 into buffer index 3
consumer 3: removed item 6592 from buffer index 4
and etc …
```

From this example, you can see the buffer slots are inserted/removed in order by the producers/consumers.

DIRECTIONS TO COMPLETE YOUR ASSIGNMENT

1. Download the source code three.zip from piazza
2. Upload three.zip to Edoras using sftp to programming folder
3. Unzip the three.zip on edoras using the commands:
   `unzip three.zip`
   so you will have one more folders: three (source files) under programming directory on edoras machine
4. Modify source files under folder three to complete this assignment (Note: you should add appropriate level of comments in your code, otherwise, up to 20% penalty may apply.)
5. Test your program to make sure it works correctly,
6. Answer the questions included in readme file.

HOW TO SUBMIT YOUR ASSIGNMENT

- The source files under the *three* folder on edoras machine will be considered for grading.
- Please finish your coding by **11:59pm July 28th** and make sure your program must execute correctly on edoras. Your grading may be started immediately after the deadline unless noticed otherwise, so please make your files ready by the deadline.
- Excuses of "but it worked on my machine" will not be accepted, so if you develop elsewhere, plan to leave time for any migration problems that might arise.

GRADING

1. Implementation – 60%
2. Appropriate level of Comments - 10%
3. README file – 30%