Programming Project 4
Threads & Synchronization
CS 441/541 – Fall 2018

Project Available		Oct. 18
Component	**Points**	**Due Date (at 11:59 pm)**
Proper use of Pthreads	5	
Proper synchronization	5	
Style & Organization	2	
Documentation	3	
Total	15	Oct. 25
Submission	Individual	Canvas

Objectives

In this project, you'll implement the **bounded buffer** problem as we talked about in class as a way to learn how to use Pthreads and semaphores.

Packaging & handing in your project

You will turn in a single compressed archive of your completed project directory to **Canvas**. A template project is available on BitBucket. You ***MUST*** use the provided semaphore library as it works more reliably on various operating systems.

Deliverables

- ◯ Code written in a **consistent style** according to the **Style Requirements** handout.
- ◯ Properly parses the command line arguments.
- ◯ Properly displays the state of the buffer.
- ◯ Properly synchronizes access to the buffer to protect against reading from an empty buffer and writing to a full buffer.
- ◯ Your program **must never busy wait** for an event to occur. If you need to cause a thread to wait then it should do so using semaphores. **Do not assume** that the semaphore wait provides a proper queue for the waiting threads (if you need such a concept).
- ◯ Make sure to seed the random number generator before using it. Seed the random number generator before starting threads with the following command: `srandom(time(NULL))`
- ◯ You will need to put bounds on the random number generator by using the modulus operator on the output: `i = random()%LIMIT;`
- ◯ To sleep for less than 1 full second you must use the usleep() command (instead of sleep()) to sleep for some number of microseconds.
- ◯ When printing output to stdout you may notice that the output from multiple threads interleave themselves. To *fix* this problem you might consider creating a binary semaphore to protect calls to the `printf` function so only one thread is printing to the console at a time.
- ◯ You must print out the parsed command line parameters before starting execution. If an optional parameter is not supplied, then you must display the default value for that parameter.
- ◯ If the user provides an incorrect set of command line arguments to your program you must immediately display an error message and a message describing the correct use of your program. After displaying those messages your program will immediately exit.

The Bounded-Buffer Problem – (15 Points)

We will be following the bounded buffer problem and solutions presented in the Little Book of Semaphores (LBS) `http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf`.

Buffer

The buffer will be manipulated with two functions, insert_item() and remove_item(), which are called by the producer and consumer threads, respectively. A skeleton outlining these functions is found below. The insert_item() and remove_item() functions will synchronize the producer and consumer using the algorithms outlined in the LBS.

```
#include "buffer.h"
/* the buffer */
int * buffer;
int insert_item(int item) {
/* insert item into buffer
return 0 if successful, otherwise
return -1 indicating an error condition */
}
int remove_item(int *item) {
/* remove an object from buffer
placing it in item
return 0 if successful, otherwise
return -1 indicating an error condition */
}
```

Initialization and `main()`

The buffer will also require an initialization function that initializes the mutex, empty and full semaphores. The main() function will initialize the buffer and create the separate producer and consumer threads. Once it has created the producer and consumer threads, the main() function will sleep for a period of time and, upon awakening, will terminate the application. The main() function will be passed three parameters on the command line: 1. How long to sleep before terminating
2. The number of producer threads
3. The number of consumer threads

The user may supply a command line option specifying the buffer size which can be any integer value greater than zero (0). This should be an *optional* fourth argument. If it is not provided then the buffer size should default to ten (10) items.

```
#include "buffer.h"
int main(int argc, char *argv[]) {
/* 1. Get command line arguments argv[1],argv[2],argv[3] */
/* 2. Dynamically allocate space for buffer and initialize it */
/* 3. Create producer thread(s) */
/* 4. Create consumer thread(s) */
/* 5. Sleep */
/* 6. Exit */
}
```

The Producer and Consumer Threads

The producer thread will alternate between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers will be produced using the rand() function, which produces random

integers between 0 and RAND MAX, you will need to use modulus division to produce only a number between 0 and 9. The consumer will also sleep for a random period of time and, upon awakening, will attempt to remove an item from the buffer. Producer and consumer threads should sleep for a random amount of **microseconds** between 0 and 1 second.

Output and Bookkeeping

You will need to print the buffer upon initialization, and after a element has been inserted or removed. You will need to track the number of elements produced and consumed. These totals will be displayed just before the program terminates (after the time to live expires). When a thread updates the buffer it will print its thread identifier, the total corresponding to their task, and the value either produced or consumed.

Examples

Example specifying buffer size of 4 elements:

```
shell$ ./bounded-buffer 2 2 1 4
Buffer Size               :   4
Time To Live (seconds)    :   2
Number of Producer threads:   2
Number of Consumer threads:   1
------------------------------
Initial Buffer:                          [  -1^v  -1  -1  -1]
Producer  1: Total    1, Value   6       [   6v  -1^  -1  -1]
Consumer  0: Total    1, Value   6       [  -1  -1^v  -1  -1]
Producer  0: Total    2, Value   4       [  -1   4v  -1^  -1]
Consumer  0: Total    2, Value   4       [  -1  -1  -1^v  -1]
Producer  1: Total    3, Value   9       [  -1  -1   9v  -1^]
Consumer  0: Total    3, Value   9       [  -1  -1  -1  -1^v]
Producer  0: Total    4, Value   5       [  -1^  -1  -1   5v]
Consumer  0: Total    4, Value   5       [  -1^v  -1  -1  -1]
Producer  0: Total    5, Value   1       [   1v  -1^  -1  -1]
Producer  1: Total    6, Value   6       [   1v   6  -1^  -1]
Consumer  0: Total    5, Value   1       [  -1   6v  -1^  -1]
Producer  0: Total    7, Value   4       [  -1   6v   4  -1^]
-----------+---------
Produced   |    7
Consumed   |    5
-----------+---------
```

Example using the default buffer size of 10 elements.

```
shell$ ./bounded-buffer 3 3 2
Buffer Size               :  10
Time To Live (seconds)    :   3
Number of Producer threads:   3
Number of Consumer threads:   2
------------------------------
Initial Buffer:                          [  -1^v  -1  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  1: Total    1, Value   9       [   9v  -1^  -1  -1  -1  -1  -1  -1  -1  -1]
Consumer  1: Total    1, Value   9       [  -1  -1^v  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  2: Total    2, Value   6       [  -1   6v  -1^  -1  -1  -1  -1  -1  -1  -1]
Consumer  1: Total    2, Value   6       [  -1  -1  -1^v  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total    3, Value   6       [  -1  -1   6v  -1^  -1  -1  -1  -1  -1  -1]
Consumer  0: Total    3, Value   6       [  -1  -1  -1  -1^v  -1  -1  -1  -1  -1  -1]
Producer  2: Total    4, Value   2       [  -1  -1  -1   2v  -1^  -1  -1  -1  -1  -1]
Consumer  0: Total    4, Value   2       [  -1  -1  -1  -1  -1^v  -1  -1  -1  -1  -1]
Producer  1: Total    5, Value   3       [  -1  -1  -1  -1   3v  -1^  -1  -1  -1  -1]
Consumer  0: Total    5, Value   3       [  -1  -1  -1  -1  -1  -1^v  -1  -1  -1  -1]
Producer  0: Total    6, Value   7       [  -1  -1  -1  -1  -1   7v  -1^  -1  -1  -1]
Producer  1: Total    7, Value   9       [  -1  -1  -1  -1  -1   7v   9  -1^  -1  -1]
Consumer  1: Total    6, Value   7       [  -1  -1  -1  -1  -1  -1   9v  -1^  -1  -1]
Producer  2: Total    8, Value   6       [  -1  -1  -1  -1  -1  -1   9v   6  -1^  -1]
Producer  0: Total    9, Value   1       [  -1  -1  -1  -1  -1  -1   9v   6   1  -1^]
Consumer  1: Total    7, Value   9       [  -1  -1  -1  -1  -1  -1  -1   6v   1  -1^]
Consumer  1: Total    8, Value   6       [  -1  -1  -1  -1  -1  -1  -1  -1   1v  -1^]
Producer  1: Total   10, Value   1       [  -1^  -1  -1  -1  -1  -1  -1  -1   1v   1]
Consumer  0: Total    9, Value   1       [  -1^  -1  -1  -1  -1  -1  -1  -1  -1   1v]
Consumer  0: Total   10, Value   1       [  -1^v  -1  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  1: Total   11, Value   8       [   8v  -1^  -1  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total   12, Value   4       [   8v   4  -1^  -1  -1  -1  -1  -1  -1  -1]
Producer  0: Total   13, Value   3       [   8v   4   3  -1^  -1  -1  -1  -1  -1  -1]
Consumer  0: Total   11, Value   8       [  -1   4v   3  -1^  -1  -1  -1  -1  -1  -1]
Consumer  1: Total   12, Value   4       [  -1  -1   3v  -1^  -1  -1  -1  -1  -1  -1]
Producer  2: Total   14, Value   5       [  -1  -1   3v   5  -1^  -1  -1  -1  -1  -1]
Consumer  1: Total   13, Value   3       [  -1  -1  -1   5v  -1^  -1  -1  -1  -1  -1]
Producer  1: Total   15, Value   5       [  -1  -1  -1   5v   5  -1^  -1  -1  -1  -1]
Consumer  0: Total   14, Value   5       [  -1  -1  -1  -1   5v  -1^  -1  -1  -1  -1]
Consumer  1: Total   15, Value   5       [  -1  -1  -1  -1  -1  -1^v  -1  -1  -1  -1]
Producer  0: Total   16, Value   3       [  -1  -1  -1  -1  -1   3v  -1^  -1  -1  -1]
Consumer  0: Total   16, Value   3       [  -1  -1  -1  -1  -1  -1  -1^v  -1  -1  -1]
Producer  2: Total   17, Value   4       [  -1  -1  -1  -1  -1  -1   4v  -1^  -1  -1]
Producer  0: Total   18, Value   4       [  -1  -1  -1  -1  -1  -1   4v   4  -1^  -1]
-----------+---------
Produced   |   18
Consumed   |   16
-----------+---------
```

Testing

Testing is an important part of the software development life cycle. You must describe in your documentation how you tested your project to ensure it met the requirements of the problem.

Note, that for synchronization problem just running the program one, twice, ..., 100, ..., 10,000 times **will not suffice** to highlight all of the possible ways that the threads can be interwoven during execution. You will want to think about how to force more contention in the system to become more confident in your solution. Testing and debugging tend to be one of the greatest challenges to writing concurrent code because of this reality.

Your documentation must discuss the testing methodology that you used used to validate that your solution was valid in design and in the implementation of that design.