Programming Assignment 5
Producer-Consumer Problem

Objective

The objective of this assignment is to provide a semaphore-based solution to the producer consumer problem using a bounded buffer.

Assignment: Using Threads and Mutex/Counting Semaphores

In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes. The solution presented in Figs 1 & 2 uses three semaphores: empty and full, which count the number of empty and full slots in the buffer, and mutex, which is a binary semaphore that protects the actual insertion or removal of items in the buffer. For this project, you will use standard counting semaphores for empty and full and a mutex lock, rather than a binary semaphore, to represent mutex. The producer and consumer—running as separate threads —will move items to and from a buffer that is synchronized with the empty, full, and mutex structures. You can solve this problem using the *Pthreads* library.

```
while (true) {
     . . .
   /* produce an item in next_produced */
     . . .
   wait(empty);
   wait(mutex);
     . . .
   /* add next_produced to the buffer */
     . . .
   signal(mutex);
   signal(full);
}
```

Figure 1: Structure of the producer process

```
while (true) {
   wait(full);
   wait(mutex);
     . . .
   /* remove an item from buffer to next_consumed */
     . . .
   signal(mutex);
   signal(empty);
     . . .
   /* consume the item in next_consumed */
     . . .
}
```

Figure 2: Structure of the consumer process

The Buffer

Internally, the buffer will consist of a fixed-size array of type buffer item (which will be defined using a typedef). The array of buffer item objects will be manipulated as a circular queue. The definition of buffer item, along with the size of the buffer, can be stored in a header file such as the following:

```
/* buffer.h */
#define BUFFER_SIZE 30
typedef struct {
   uint16_t   cksum;
   uint8_t    buffer[BUFFER_SIZE];
} buffer_item;
```

```
struct buffer_item item[NUM_ITEMS];
```

The buffer will be manipulated with two functions, `insert_item()` and `remove_item()`, which are called by the producer and consumer threads, respectively. The `insert_item()` and `remove_item()` functions will synchronize the producer and consumer using the algorithms outlined in Fig 3. The buffer will also require an initialization function that initializes the mutual-exclusion object mutex along with the 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 (in seconds)
2. The number of producer threads
3. The number of consumer threads

A template for this function appears in Fig 4.

```
#include "buffer.h"

/* the buffer */
buffer_item buffer[BUFFER_SIZE];

int insert_item(buffer_item item) {
   /* insert item into buffer
    return 0 if successful, otherwise
    return -1 indicating an error condition */
}

int remove_item(buffer_item *item) {
   /* remove an object from buffer
    placing it in item
    return 0 if successful, otherwise
    return -1 indicating an error condition */
}
```

Figure 3: Outline of buffer operations

```
#include "buffer.h"

int main(int argc, char *argv[]) {
 /* 1. Get command line arguments argv[1],argv[2],argv[3] */
 /* 2. Initialize buffer */
 /* 3. Create producer thread(s) */
 /* 4. Create consumer thread(s) */
 /* 5. Sleep */
 /* 6. Exit */
}
```

Figure 4: Template for main program

Producer/Consumer threads

The producer thread(s) will create the data *item(s)* which includes the checksum and random data (produced using the `rand()` function, which produces random integers between 0 and RAND MAX). The consumer thread(s) read the shared memory buffer of item(s), calculate the checksum and compare that with the value stored in the shared memory buffer to ensure that the data did not get corrupted.

The producer/consumer program (*prodcon.c*) that takes three arguments from the command line (no prompting the user from within the program).

1. To start the *prodcon* program

   *./**prodcon** <delay> <#producer threads> <#consumer threads>*where the argument *<delay >* indicates how long to sleep (in seconds) before terminating and *<#producer threads>* indicates number of threads and *<#consumer threads>* indicates the number of consumer threads.

Error Handling

Perform the necessary error checking to ensure the correct number of command-line parameters. If the consumer detects a mismatched checksum it is to report the error along with the expected checksum and the calculated checksum and exit the program.

Grading

The program will be graded on the basic functionality, error handling and how well the implementation description was followed. Be sure to name your program **prodcon.c** (no extra characters, capitals) Note that documentation and style are worth 10% of the assignment's grade!

Submission

The source code for program should be available on Canvas along with a README/Output file that provides any documentation and sample output.