# Assignment 4. Thread Synchronization: Producer-Customer Problem
**due at midnight March 12, 2018**

*Please provide a* ***Make file and README file*** *to explain how to compile and run your program.*

You have learned a semaphore-based solution to the producer-customer problem using a bounded buffer. In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in Figures 1 and 2 below. The solution presented in class (and in your textbook) uses three semaphores : empty and full, which count the number of empty and full slots in the buffer, and mutex, which is a binary (or mutual-exclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, standard counting semaphores will be used for empty and full, and a mutex lock, rather than a binary semaphore, will be used to represent mutex. The producer and consumer----- running as separate threads---- will move items to and from a buffer that is synchronized with these empty, full and mutex structures.

```
do{
 …
 //produce an item in nextp
 wait(empty);
 wait(mutex);
 …
  //add nextp to buffer
  signal(mutex)
  signal(full)
}while (TRUE);
```

**Figure 1**

```
do{
  wait(full);
  wait(mutex);
  …
  //remove nextp from buffer to nextp
  …
  signal(mutex);
  signal(empty);
  ….
  //consume the item in nextc
  …..
}while(TRUE);
```

**Figure 2**

## 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 following:

```
/*buffer.h*/
typedef int buffer_item;
#define BUFFER_SIZE 5
```

The buffer will be manipulated with two functions, insert_item() and remove_item(), which are called by producer and consumer threads, repectively. A skeleton outlining these functions appears in Figure 3 below.

```
#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**

The insert_item() and remove_item() functions will synchronize the producer and consumer using the algorithms outlined in Figures 1 and 2. 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 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.

a. How long to sleep before terminating
b. The number of producer threads
c. The number of consumer threads

A skeleton for this function appears in Figure 4 below.

```
int main(int argc, char *argv[]){
   /*1. Get command line arguments argv[1], argv[2], argv[3]*/
   /*2. Initialize buffer*/
```

```
    /*3. Create the producer thread(s)*/
    /*4. Create the consumer thread(s)*/
    /*5. Sleep*/
    /*6. Exit*/
}
```

**Figure 4**

**Producer and Consumer Threads**

The producer thread will alternate between sleeping for a random period of time and inserting a random integer into buffer. Random numbers will be produced using rand() function, which produces random numbers between 0 and RAND_MAX. The consumer will also sleep for a random period of time and, upon awakening, will attempt to remove an item from the buffer. An outline of the producer and consumer threads appears in Figure 5 below.

```
#include <stdlib.h>
#include “buffer.h”

void *producer(void *param){
    buffer_item item;
    while(TRUE){
      /* sleep for a random period of time*/
      sleep(...);
      /*generate a random number*/
      item = rand();
      if(insert_item(item))
        fprint(“report error condition”);
      else
         printf(“producer produced %d\n”,item);
 }


void *consumer(void *param){
   buffer_item item;

   while(TRUE){
      /* sleep for a random period of time*/
      sleep(...);
      if(reove_item(item))
        fprint(“report error condition”);
      else
         printf(“consumer consumed %d\n”,item);
}
```

**Figure 5**