Homework 3 Report
CSC 4320 Operating Systems
Spring 2018
Name: Jimmy Tran
Email: jtran25@student.gsu.edu

As a side note regarding my program, I intentionally delayed when the producer and consumer threads started in order to avoid attempts to consume before producing and to avoid getting too much output at once.

1) Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define TRUE 1
typedef int buffer_item;
#define BUFFER_SIZE 8

buffer_item START_NUMBER;

int insert_item(buffer_item item);

int remove_item(buffer_item *item);



buffer_item buffer[BUFFER_SIZE];

pthread_mutex_t mutex;
sem_t empty;
sem_t full;

int insertPointer = 0, removePointer = 0;
```

```
void *producer(void *param);
void *consumer(void *param);

int insert_item(buffer_item item)
{
   /* Implementation of the insert_item() function */
        if(insertPointer>=BUFFER_SIZE) //return to beginning of buffer
                insertPointer=0;
        if(insertPointer<BUFFER_SIZE){
                buffer[insertPointer]=item;
                insertPointer++;
        }
        else
                return -1;
        return 0;
}


int remove_item(buffer_item *item)
{
        /* Implementation of the remove_item function */
        if(removePointer>=BUFFER_SIZE) //return to beginning of buffer
                removePointer=0;
        if(removePointer>=0&&removePointer<BUFFER_SIZE){
                *item = buffer[removePointer];
                removePointer++;
        }
        else
                return -1;
        return 0;
```

```
}


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

        int sleepTime, producerThreads, consumerThreads;
        int i, j;

        if(argc != 5)
        {
                fprintf(stderr, "Usage: <sleep time> <producer threads> <consumer threads>
<start number>\n");
                return -1;
        }

        sleepTime = atoi(argv[1]);
        producerThreads = atoi(argv[2]);
        consumerThreads = atoi(argv[3]);
        START_NUMBER = atoi(argv[4]);

        /* Initialize the synchronization tools */
        sem_init(&full, 0, 0);
        sem_init(&empty, 0, BUFFER_SIZE);
        pthread_mutex_init(&mutex, NULL);
```

```
	/* Create the producer and consumer threads */
	pthread_t pro, con;
	for(int i=0; i<producerThreads;i++){
		pthread_create(&pro, NULL, producer, START_NUMBER); //create producers
	}
	for(int j=0; j<consumerThreads;j++){
		pthread_create(&con, NULL, consumer, NULL); //create consumers
	}

	/* Sleep for user specified time */
	sleep(sleepTime);

	return 0;
}

void *producer(void *param)
{
	/* Implementation of the producer thread -- refer to Figure 5.26 on page 256 */
	buffer_item item;

	while(TRUE){
		sleep(1);
		item=START_NUMBER;

		sem_wait(&empty); //lock empty semaphore if !0
		pthread_mutex_lock(&mutex); //mutex lock

		if(insert_item(item)){
			fprintf(stderr, "Insert into buffer failed\n");
		}
```

```
                else{
                        printf("Producer %u produced %d \n",(unsigned int)pthread_self(), item);
                        START_NUMBER++;
                }

                pthread_mutex_unlock(&mutex); //mutex unlock
                sem_post(&full); //increment full semaphore
        }
}

void *consumer(void *param)
{
        /* Implementation of the consumer thread -- refer to Figure 5.26 on page 256 */
        buffer_item item;

        while(TRUE){
                sleep(2);
                sem_wait(&full); //lock full semaphore
                pthread_mutex_lock(&mutex); //mutex lock

                if(remove_item(&item)){
                        fprintf(stderr, "Removal from buffer failed\n");
                }
                else
                        printf("Consumer %u consumed %d \n",(unsigned int)pthread_self(),
item);

                pthread_mutex_unlock(&mutex); //mutex unlock
                sem_post(&empty); //increments empty semaphore
        }
}
```

2) Screenshots:

Scenario 1: More Producers than Consumers

Scenario 2: More Consumer than Producers

Scenario 3: Equal number of Consumers and Producers

```
jimmy@jimmy-VirtualBox:~$ ./buffer 10 5 5 10
Producer 2534795008 produced 10
Producer 2526402304 produced 11
Producer 2518009600 produced 12
Producer 2509616896 produced 13
Producer 2501224192 produced 14
Producer 2534795008 produced 15
Producer 2526402304 produced 16
Producer 2518009600 produced 17
Consumer 2492831488 consumed 10
Producer 2509616896 produced 18
Consumer 2484438784 consumed 11
Producer 2501224192 produced 18
Consumer 2476046080 consumed 12
Consumer 2467653376 consumed 13
Consumer 2459260672 consumed 14
Producer 2534795008 produced 20
Producer 2526402304 produced 21
Producer 2518009600 produced 22
Consumer 2492831488 consumed 15
Producer 2509616896 produced 23
Consumer 2484438784 consumed 16
Producer 2501224192 produced 23
Consumer 2467653376 consumed 17
Consumer 2476046080 consumed 18
Producer 2526402304 produced 23
Producer 2534795008 produced 23
Consumer 2459260672 consumed 18
Producer 2518009600 produced 23
Consumer 2492831488 consumed 20
Producer 2509616896 produced 28
Consumer 2484438784 consumed 21
Producer 2501224192 produced 28
Consumer 2467653376 consumed 22
Consumer 2476046080 consumed 23
Producer 2526402304 produced 28
Producer 2534795008 produced 28
```