Objectives

- To demonstrate the working of producer and consumer functions.
- To create a producer thread and consumer thread to perform producer and consumer routines respectively.
- To show synchronization among the producer and consumer threads using Semaphores.

Pre-Lab Theory:

Producer-Consumer Problem Concept:

The producer-consumer problem is a synchronization problem. There is a fixed-size buffer and two processes called as 'producer' and 'consumer'. The producer produces items and enters them into the buffer. The consumer removes the items from the buffer by consuming/using them. These two processes will be synchronized and generate the correct size of no. of products after developed and consumed if and only if during the execution of both the processes no interruption (high priority process) occurs or in other words, no preemption occurs otherwise if the preemption occurs the buffer size will have the incorrect result. The solution of this problem is synchronization by applying the Semaphore.

Synopsis:

```
sem_wait(sem_t * semaphore)
```

```
sem_post(sem_t * semaphore)
```

In-Lab Tasks:

Task 1: *Carefully read the code snippet for the producer function and in the contrary develop a consumer function.*

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

// Initialize a mutex to 1
int mutex = 1;

// Number of full slots as 0
int full = 0;

// Number of empty slots as size
// of buffer
int empty = 10, x = 0;

// Function to produce an item and
// add it to the buffer
```

```
void producer()
{
    // Decrease mutex value by 1
    --mutex;

    // Increase the number of full
    // slots by 1
    ++full;

    // Decrease the number of empty
    // slots by 1
    --empty;

    // Item produced
    x++;
    printf("\nProducer produces"
           "item %d",
           x);

    // Increase mutex value by 1
    ++mutex;
}
```

Task 2: *Create a driver code for the producer and consumer functions to invoke with the following menu output.*

Enter the Choice:

1. *Producer*
2. *Consumer*
3. *Exit*

Make sure the condition for Producer

```
if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }
```

Make sure the condition for Consumer

```
if ((mutex == 1)
                && (full != 0)) {
```

```
        consumer();
    }
```

Driver Code(main() function)

Output:

Task 3: *Given the declaration of shared buffer and "in", and "out" variables to control the index of the buffer. The Consumer function is given below, write the Producer function on the same pattern.*

```
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int count = 0;  // Number of items in the buffer
int in = 0;     // Index for producer to insert
int out = 0;    // Index for consumer to remove
consumer() {
        // Remove item from buffer
        int item = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
        printf("Consumed item %d\n", item);
}
```

Producer Code:

Task 4: *Attach a driver code(main function) similar to Task 2 to run the above producer and consumer functions.*

Driver Code:

Output (After Running a complete program)

Task 5: *Create a Producer thread and Consumer thread to run the respective functions(routines) Modify the signature of the Producer and Consumer function to adhere to the POSIX thread routine standard.*

Modified Signature of Producer() and Consumer() routines

Necessary code to create above threads and pthread_create() calls

Pthread_join() calls:

Highlight the problem (if any) in the output with reason

Producer and Consumer Threads Synchronization:

Task 6 and onwards Pre-requisites:

```
#include <semaphore.h>
#define RAND_DIVISOR 100000000
#define TRUE 1
/* The mutex lock */
pthread_mutex_t mutex;

/* the semaphores */
sem_t full, empty;

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

/* buffer counter */
int counter;

pthread_t tid;       //Thread ID
pthread_attr_t attr; //Set of thread attributes

void *producer(void *param); /* the producer thread */
void *consumer(void *param); /* the consumer thread */

void initializeData() {

   /* Create the mutex lock */
   pthread_mutex_init(&mutex, NULL);
```

```
   /* Create the full semaphore and initialize to 0 */
   sem_init(&full, 0, 0);

   /* Create the empty semaphore and initialize to BUFFER_SIZE */
   sem_init(&empty, 0, BUFFER_SIZE);

   /* Get the default attributes */
   pthread_attr_init(&attr);

   /* init buffer */
   counter = 0;
}
```

Task 6: *Given the add item function insert_Item() below. Write the function remove_Item() function on the same pattern, considering the above variables.*

```
/* Add an item to the buffer */
int insert_item(int item) {
   /* When the buffer is not full add the item
      and increment the counter*/
   if(counter < BUFFER_SIZE) {
      buffer[counter] = item;
      counter++;
      return 0;
   }
   else { /* Error the buffer is full */
      return -1;
   }
}
```

remove_item() code

***Task 7**:Given the Producer() function code below calling insert_item() function. Write the Consumer() function code to call the remove_item() function.*

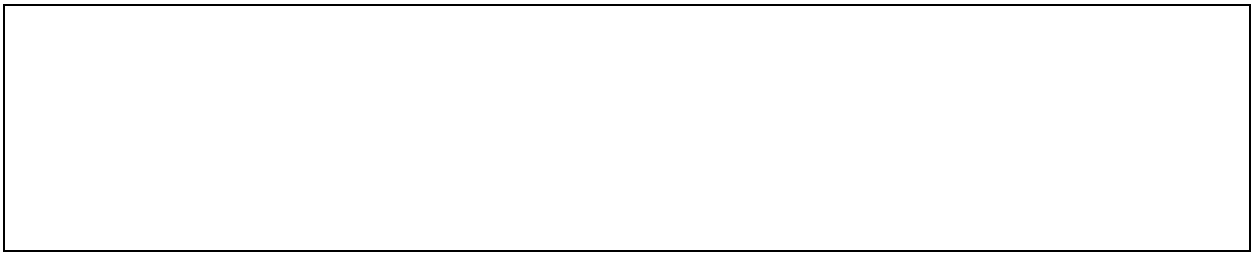
```
/* Producer Thread */
void *producer(void *param) {
   buffer_item item;

   while(TRUE) {
      /* sleep for a random period of time */
      int rNum = rand() / RAND_DIVISOR;
      sleep(rNum);

      /* generate a random number */
```

```
      item = rand();

      /* acquire the empty lock */
      sem_wait(&empty);
      /* acquire the mutex lock */
      pthread_mutex_lock(&mutex);

      if(insert_item(item)) {
         fprintf(stderr, " Producer report error condition\n");
      }
      else {
         printf("producer produced %d\n", item);
      }
      /* release the mutex lock */
      pthread_mutex_unlock(&mutex);
      /* signal full */
      sem_post(&full);
   }
}
```

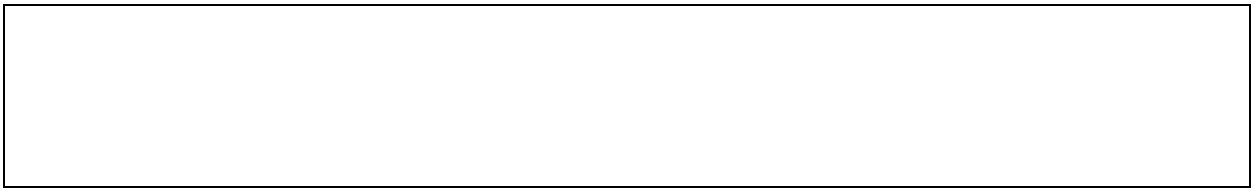
Consumer() function Code:

Task 8: *Complete the Driver Code given below to create the threads and join the producer and consumer threads to execute the respective producer and consumer routines*

```
int main(int argc, char *argv[]) {
   /* Loop counter */
   int i;

   /* Verify the correct number of arguments were passed in */
   if(argc != 4) {
      fprintf(stderr, "USAGE:./main.out <INT> <INT> <INT>\n");
   }

   int mainSleepTime = atoi(argv[1]); /* Time in seconds for main to
sleep */
   int numProd = atoi(argv[2]); /* Number of producer threads */
   int numCons = atoi(argv[3]); /* Number of consumer threads */

   /* Initialize the app */
   initializeData();

   /* Create the producer threads */
```



```
   /* Create the consumer threads */
```



```
    /* Sleep for the specified amount of time in milliseconds */
    sleep(mainSleepTime);

    /* Exit the program */
    printf("Exit the program\n");
    exit(0);
}
```