Kshitij Deshpande   K11   Roll No: 33312

Assignment 4 - A

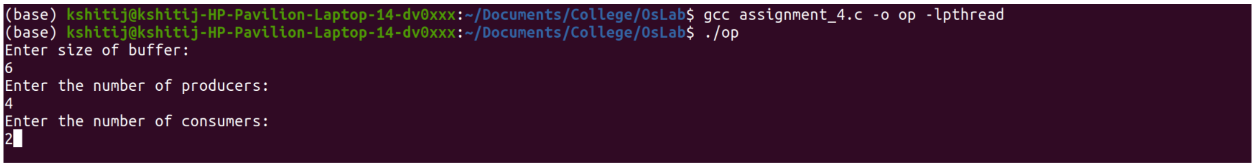
Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <math.h>
//#define BUF_SIZE 5

//Counting semaphores
sem_t empty,full;

//for mutual exclusion,binary semaphore
//sem_t S;

//Mutex Lock
pthread_mutex_t lock;

//Thread functions for producer and consumer
void *producer_fn(void *arg);
void *consumer_fn(void *arg);


int count=0;

//Current\ Indices
int pidx,cidx;
```

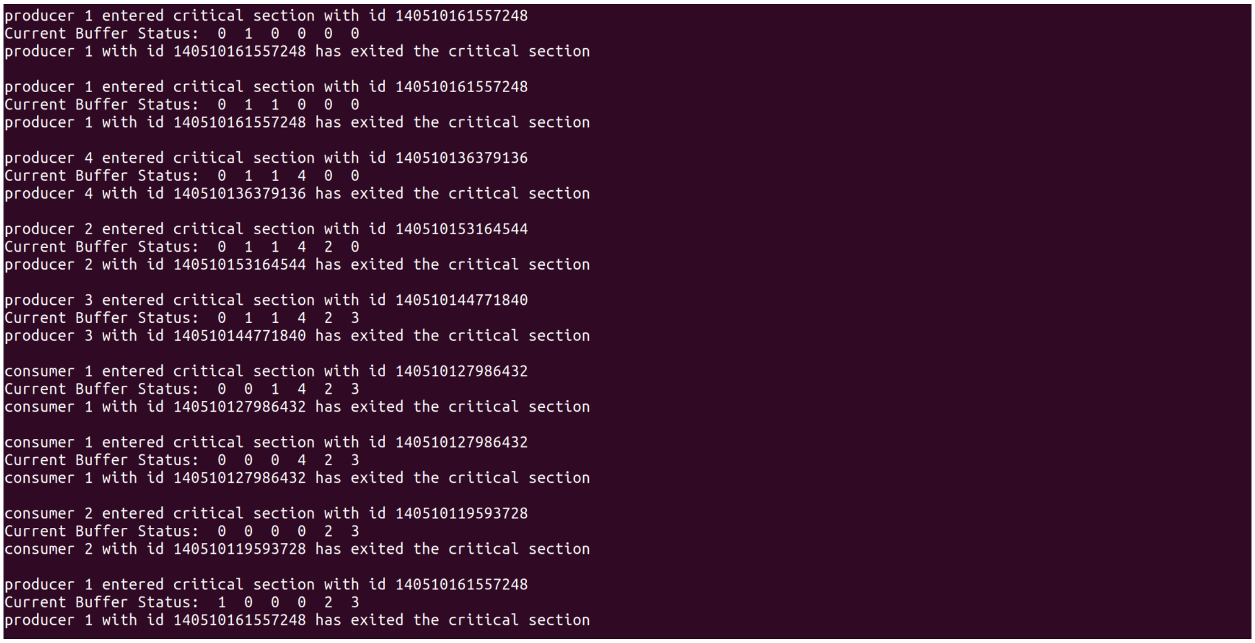
```
//Buffer array
int* buffer;

//size of buffer
int BUF_SIZE;

//Main function
int main() {

    int res,prod_count,cons_count,i;
    int *loc;
    printf("Enter size of buffer: \n");
    scanf("%d",&BUF_SIZE);
    //dyanmic buffer allocation
    buffer = (int*)malloc(BUF_SIZE* sizeof(int));

    printf("Enter the number of producers: \n");
    scanf("%d",&prod_count);
    printf("Enter the number of consumers: \n");
    scanf("%d",&cons_count);

    //Create threads
    pthread_t producers[prod_count], consumers[cons_count];

    //Initialise semaphores
    res=sem_init(&empty,0,BUF_SIZE);
    if(res!=0)
    {
        printf("Eror in semaphore initialisation! \n");
        exit(EXIT_FAILURE);
    }
    res=sem_init(&full,0,0);
    if(res!=0)
    {
        printf("Eror in semaphore initialisation! \n");
        exit(EXIT_FAILURE);
    }

    //initially resources available so set to 1
    //binary semaphore initialised to 1
    //res=sem_init(&S,0,1);
    //if(res!=0)
    //{
    //    printf("Eror in semaphore initialisation! \n");
    //    exit(EXIT_FAILURE);
    //}


    //initialise mutex
    res=pthread_mutex_init(&lock,NULL);
    if(res!=0)
    {
        printf("Eror in mutex initialisation! \n");
        exit(EXIT_FAILURE);
    }

    pidx=0;
    cidx=0;

    printf("producer threads are being created....\n");
    for(i = 1; i <= prod_count; i++) {
        printf("creating producer number %d \n",i);
        //allocate memory
        loc=(int *) malloc(sizeof(int));
        //associate value at memory
        *loc=i;
        //create thread
        res = pthread_create(&producers[i], NULL, producer_fn,
        loc);
        if (res != 0) {
```

```
                perror("Error in thread creation!");
                exit(EXIT_FAILURE);
        }

  }

  printf("consumer threads are being created....\n");
  for(i = 1; i <= cons_count; i++) {
        printf("creating consumer number %d \n",i);
        //allocate memory
        loc=(int *) malloc(sizeof(int));
        //associate value at memory
        *loc=i;
        //create thread
        res = pthread_create(&consumers[i], NULL, consumer_fn,
        loc);
        if (res != 0) {
                perror("Error in thread creation!");
                exit(EXIT_FAILURE);
        }
  }


  for(int i=0;i<prod_count;i++){
        res = pthread_join(producers[i],NULL);
        if(res!=0){
               printf("Error in thread join: \n");
               exit(EXIT_FAILURE);
        }
 }



for(int i=0;i<cons_count;i++){
        res = pthread_join(consumers[i],NULL);
        if(res!=0){
                printf("Error in thread join: \n");
                exit(EXIT_FAILURE);
        }
   }






    pthread_mutex_destroy(&lock);
    //sem_destroy(&S);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}

void *producer_fn(void *arg) {
    int *my_number = (int*) arg;
    while(1){
        int i;
        //int rand_num=rand();
        //sleep for random time...this ensure async behavior
        //sleep(rand_num);
        //decrements value as new item is produced it will take up one empty space in buffer
        sem_wait(&empty);
        //lock the mutex
        pthread_mutex_lock(&lock);
        //sem_wait(&S);

        //now critical section starts
        printf("\nproducer %d entered critical section with id %lu \n",*my_number,pthread_self());
        //store in buffer
        buffer[pidx]=*my_number;
        //print buffer
        printf("Current Buffer Status: ");
        for(int i=0;i<BUF_SIZE;i++)
            printf(" %d ",buffer[i]);
        printf("\n");
        //count++;
        //printf("Count is %d\n",count);
        //update index
        pidx=(pidx+1)%BUF_SIZE;
        printf("producer %d with id %lu has exited the critical section
\n",*my_number,pthread_self());
        //unlock the mutex,release the locks
        pthread_mutex_unlock(&lock);
        //sem_post(&S);
        //locks the semaphore,increments value
        sem_post(&full);
    }
}

void *consumer_fn(void *arg) {
    int *my_number = (int*) arg;
    while(1){
        int i;
        //int rand_num=rand();
        //sleep for random time...this ensure async behavior
        //sleep(rand_num);
        //decrements value as full elements would decrease after consuming
        sem_wait(&full);
        //lock the mutex
        pthread_mutex_lock(&lock);
        //sem_wait(&S);

        //now critical section starts
        printf("\nconsumer %d entered critical section with id %lu
\n",*my_number,pthread_self());
        //consume from buffer
        buffer[cidx]=0;
        //print buffer
        printf("Current Buffer Status: ");
        for(int i=0;i<BUF_SIZE;i++)
                printf(" %d ",buffer[i]);
        printf("\n");
        //count--;
        //printf("Count is %d\n",count);
        //printf("for consumer id %lu",pthread_self());
        //update index
        cidx=(cidx+1)%BUF_SIZE;
        printf("consumer %d with id %lu has exited the critical
section\n",*my_number,pthread_self());
        //unlock the mutex, release lock
        pthread_mutex_unlock(&lock);
        //sem_post(&S);
        //increments value as new empty space will be created after
consuming from buffer
        sem_post(&empty);
    }
}
```

Output:



```
(base) kshitij@kshitij-HP-Pavilion-Laptop-14-dv0xxx:~/Documents/College/OsLab$ gcc assignment_4.c -o op -lpthread
(base) kshitij@kshitij-HP-Pavilion-Laptop-14-dv0xxx:~/Documents/College/OsLab$ ./op
Enter size of buffer:
6
Enter the number of producers:
4
Enter the number of consumers:
2
```



```
producer 1 entered critical section with id 140510161557248
Current Buffer Status:  0  1  0  0  0  0
producer 1 with id 140510161557248 has exited the critical section

producer 1 entered critical section with id 140510161557248
Current Buffer Status:  0  1  1  0  0  0
producer 1 with id 140510161557248 has exited the critical section

producer 4 entered critical section with id 140510136379136
Current Buffer Status:  0  1  1  4  0  0
producer 4 with id 140510136379136 has exited the critical section

producer 2 entered critical section with id 140510153164544
Current Buffer Status:  0  1  1  4  2  0
producer 2 with id 140510153164544 has exited the critical section

producer 3 entered critical section with id 140510144771840
Current Buffer Status:  0  1  1  4  2  3
producer 3 with id 140510144771840 has exited the critical section

consumer 1 entered critical section with id 140510127986432
Current Buffer Status:  0  0  1  4  2  3
consumer 1 with id 140510127986432 has exited the critical section

consumer 1 entered critical section with id 140510127986432
Current Buffer Status:  0  0  0  4  2  3
consumer 1 with id 140510127986432 has exited the critical section

consumer 2 entered critical section with id 140510119593728
Current Buffer Status:  0  0  0  0  2  3
consumer 2 with id 140510119593728 has exited the critical section

producer 1 entered critical section with id 140510161557248
Current Buffer Status:  1  0  0  0  2  3
producer 1 with id 140510161557248 has exited the critical section
```