Assignment No. 4A

Name: **Sahil Jagadale** Division: TE-11 Batch: L-11 Roll No.: **33327**

Title: Thread synchronization using counting semaphores.

Aim: Application to demonstrate producer-consumer problem with counting semaphores and mutex.

OBJECTIVE: Implement C program to demonstrate producer-consumer problem with counting semaphores and mutex.

THEORY:

Semaphores:

An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic: initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

Semaphores are the OS tools for synchronization. Two types:

1. Binary Semaphore. **2**. Counting Semaphore.

Counting semaphore

The counting semaphores are free of the limitations of the binary semaphores. A counting

semaphore comprises:

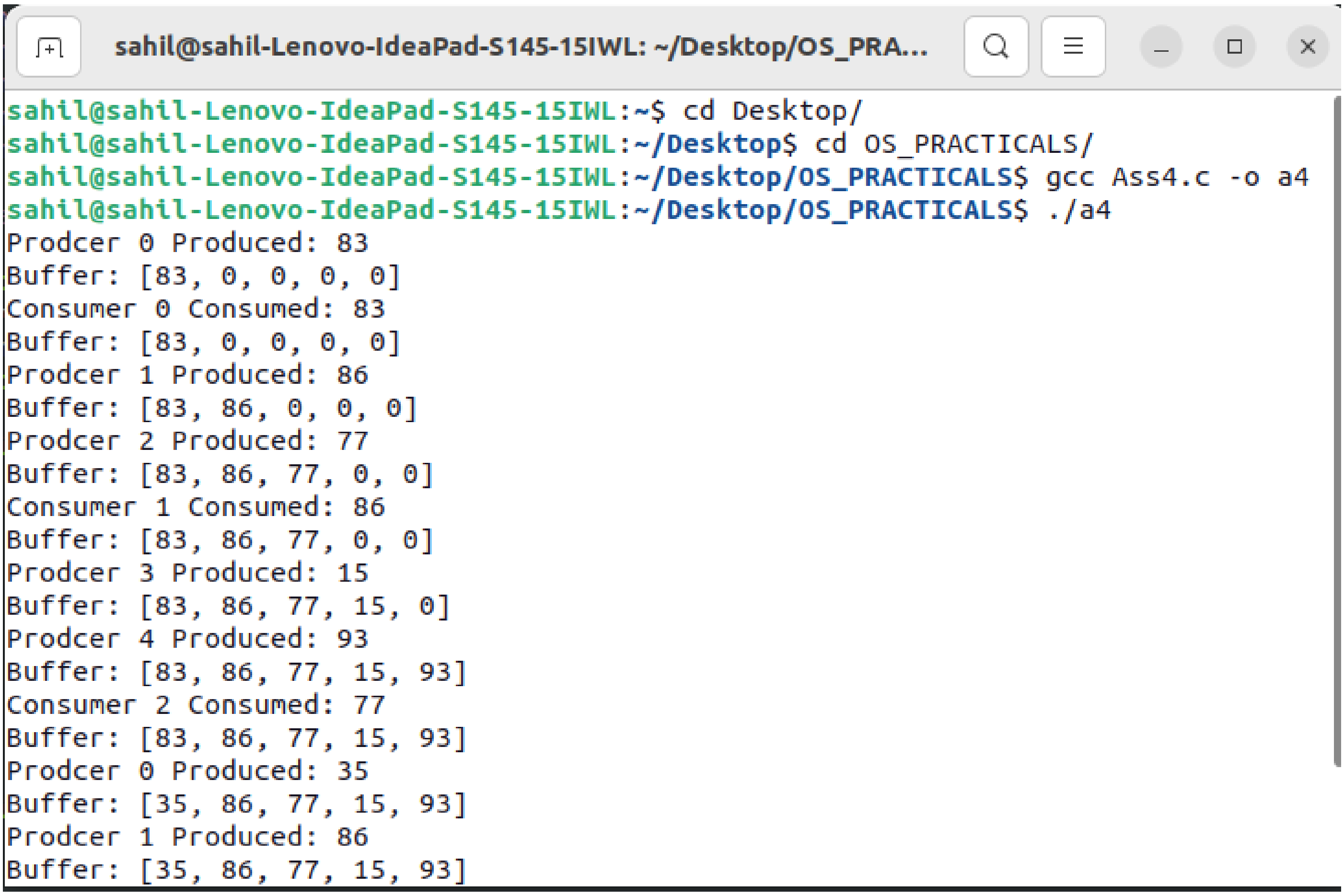
An integer variable, initialized to a value K (K>=0). During operation it can assume any value <= K, a pointer to a process queue. The queue will hold the PCBs of all those processes, waiting to enter their critical sections. The queue is implemented as a FCFS, so that the waiting processes are served in a FCFS order.

Code:

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
sem_t empty, full;
pthread_mutex_t mutex;
int in = 0, out = 0;

void print_buffer() {
   printf("Buffer: [");
   for (int i = 0; i < BUFFER_SIZE; i++) {
      printf("%d", buffer[i]);
      if (i < BUFFER_SIZE - 1) {
         printf(", ");
      }
   }
   printf("]\n");
}

void *producer(void *arg) {
   int item;

   while(1){
      item = rand() % 100;

      sem_wait(&empty);
      pthread_mutex_lock(&mutex);

      buffer[in] = item;
      printf("Prodcer %d Produced: %d\n",in, item);
      in = (in + 1) % BUFFER_SIZE;

      pthread_mutex_unlock(&mutex);
      sem_post(&full);

      print_buffer();

      sleep(2);
   }
   pthread_exit(NULL);
}

void *consumer(void *arg) {
   int item;
   while(1) {
      sem_wait(&full);
      pthread_mutex_lock(&mutex);

      item = buffer[out];
      printf("Consumer %d Consumed: %d\n",out,item);
      out = (out + 1) % BUFFER_SIZE;


      pthread_mutex_unlock(&mutex);
      sem_post(&empty);

      print_buffer();

      sleep(5);
   }
   pthread_exit(NULL);
}
```

```
int main() {
   pthread_t producer_thread, consumer_thread;

   sem_init(&empty, 0, BUFFER_SIZE);
   sem_init(&full, 0, 0);
   pthread_mutex_init(&mutex, NULL);

   pthread_create(&producer_thread, NULL, producer, NULL);
   pthread_create(&consumer_thread, NULL, consumer, NULL);

   pthread_join(producer_thread, NULL);
   pthread_join(consumer_thread, NULL);

   sem_destroy(&empty);
   sem_destroy(&full);
   pthread_mutex_destroy(&mutex);

   return 0;
}
```

Output:



```
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~$ cd Desktop/
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop$ cd OS_PRACTICALS/
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ gcc Ass4.c -o a4
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ ./a4
Prodcer 0 Produced: 83
Buffer: [83, 0, 0, 0, 0]
Consumer 0 Consumed: 83
Buffer: [83, 0, 0, 0, 0]
Prodcer 1 Produced: 86
Buffer: [83, 86, 0, 0, 0]
Prodcer 2 Produced: 77
Buffer: [83, 86, 77, 0, 0]
Consumer 1 Consumed: 86
Buffer: [83, 86, 77, 0, 0]
Prodcer 3 Produced: 15
Buffer: [83, 86, 77, 15, 0]
Prodcer 4 Produced: 93
Buffer: [83, 86, 77, 15, 93]
Consumer 2 Consumed: 77
Buffer: [83, 86, 77, 15, 93]
Prodcer 0 Produced: 35
Buffer: [35, 86, 77, 15, 93]
Prodcer 1 Produced: 86
Buffer: [35, 86, 77, 15, 93]
```