Experiment No.6
Process Management: Synchronization A. Write a C program to implement the solution of the Producer consumer problem through Semaphore.
Name Of Student:-Bhagyashri Kaleni Sutar
Roll No:-75
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 6

Aim: Write a C program to implement solution of Producer consumer problem through Semaphore

Objective:

Solve the producer consumer problem based on semaphore

Theory:

The Producer-Consumer problem is a classical multi-process synchronization problem, that is we are trying to achieve synchronization between more than one process.

There is one Producer in the producer-consumer problem, Producer is producing some items, whereas there is one Consumer that is consuming the items produced by the Producer. The same memory buffer is shared by both producers and consumers which is of fixed-size.

The task of the Producer is to produce the item, put it into the memory buffer, and again start producing items. Whereas the task of the Consumer is to consume the item from the memory buffer.

Producer consumer problem is a classical synchronization problem. We can solve this problem by using semaphores.

A **semaphore** S is an integer variable that can be accessed only through two standard operations : wait() and signal().
The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1.

```
wait(S){

while(S<=0);   // busy waiting

S--;

}

signal(S){

S++;

}
```

To solve this problem, we need two counting semaphores – Full and Empty. "Full" keeps track of number of items in the buffer at any given time and "Empty" keeps track of number of unoccupied slots.

Initialization of semaphores –
mutex = 1
Full = 0// Initially, all slots are empty. Thus full slots are 0
Empty = n // All slots are empty initially

Solution for Producer –

```
do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)
```

When producer produces an item then the value of "empty" is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of "full" is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

Solution for Consumer –

```
do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

signal(empty);

// consumes item

}while(true)
```

As the consumer is removing an item from buffer, therefore the value of "full" is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this

moment. Now, the consumer has consumed the item, thus increasing the value of "empty" by 1. The value of mutex is also increased so that producer can access the buffer now.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h> // For sleep()

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void *producer(void *arg) {
   int item;
   for (int i = 0; i < 10; ++i) { // Produce 10 items
      item = rand() % 100; // Generate a random item
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
```

```
        buffer[in] = item;
        printf("Producer produced item %d at index %d\n", item, in);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1); // Simulate production time
    }
    pthread_exit(NULL);
}

void *consumer(void *arg) {
    int item;
    for (int i = 0; i < 10; ++i) { // Consume 10 items
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

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

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(2); // Simulate consumption time
    }
    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;
}
//Out[put
Producer produced item 83 at index 0
Consumer consumed item 83 from index 0
Producer produced item 86 at index 1
Consumer consumed item 86 from index 1
Producer produced item 77 at index 2
Producer produced item 15 at index 3
```

Consumer consumed item 77 from index 2

Producer produced item 93 at index 4

Producer produced item 35 at index 0

Consumer consumed item 15 from index 3

Producer produced item 86 at index 1

Producer produced item 92 at index 2

Consumer consumed item 93 from index 4

Producer produced item 49 at index 3

Producer produced item 21 at index 4

Consumer consumed item 35 from index 0

Consumer consumed item 86 from index 1

Consumer consumed item 92 from index 2

Consumer consumed item 49 from index 3

Consumer consumed item 21 from index 4

Conclusion:

What is Semaphore?

Ans:-A semaphore is essentially a variable or abstract data type that acts as a counter.

- **It's used to regulate access to a shared resource by multiple processes or threads, preventing them from interfering with each other.**
-

Key Features:

- **Synchronization Tool:**
  - **Semaphores help coordinate the actions of multiple processes, ensuring that they access shared resources in a controlled manner.**
  -

- **Preventing Race Conditions:**
  - **They are crucial for preventing race conditions, which occur when the outcome of a program depends on the unpredictable order in which multiple processes execute.**
  -

How Semaphores Work:

- **Wait (P) Operation:**
  - **Decrements the semaphore's value.**
  -
  - **If the value becomes negative, the process is blocked (waits) until the semaphore becomes available.**
  -
- **Signal (V) Operation:**
  - **Increments the semaphore's value.**
  -
  - **If there are any processes waiting, one of them is unblocked.**

What are different types of Semaphore?

Ans:-

- **Types of Semaphores:**
  - **Binary Semaphore:**
    - **Has only two values: 0 and 1.**
    -
    - **Often used to implement locks (also called mutexes), ensuring that only one process can access a critical section at a time.**
    -
  - **Counting Semaphore:**
    - **Can have non-negative integer values.**
    - **Used to control access to a resource with a limited number of instances.**