

**Name:** K.SaiKrishna

**Reg-No:** 192311106

18. Construct a C program to simulate producer-consumer problem using semaphores.

### **Aim:**

The aim of the program is to simulate the producer-consumer problem using semaphores. The producer creates data and puts it in a buffer, while the consumer consumes the data from the buffer. Semaphores are used to synchronize access to the shared buffer.

### **Algorithm:**

1. **Initialization:**
  - Initialize two semaphores: `empty` (to track the number of empty slots) and `full` (to track the number of full slots).
  - Initialize a mutex semaphore for mutual exclusion.
2. **Producer:**
  - Wait on the `empty` semaphore (to ensure there is space).
  - Wait on the mutex semaphore (for mutual exclusion).
  - Add an item to the buffer.
  - Signal the `full` semaphore (indicating a full slot).
  - Signal the mutex semaphore to release mutual exclusion.
3. **Consumer:**
  - Wait on the `full` semaphore (to ensure there is data).
  - Wait on the mutex semaphore (for mutual exclusion).
  - Consume an item from the buffer.
  - Signal the `empty` semaphore (indicating an empty slot).
  - Signal the mutex semaphore to release mutual exclusion.

### **Procedure:**

- The producer creates data and puts it into the buffer.
- The consumer retrieves data from the buffer and consumes it.
- The semaphores ensure that the buffer is accessed in a synchronized manner.

### **Code:**

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

```
#include <stdbool.h>
```

```
int main() {
```

```
int n, m, i, j, k;
```

```
printf("Enter number of processes: ");
```

```
scanf("%d", &n);
```

```
printf("Enter number of resources: ");
```

```
scanf("%d", &m);
```

```
int Allocation[n][m], Maximum[n][m], Need[n][m], Available[m];
```

```
printf("Enter Allocation matrix:\n");
```

```
for (i = 0; i < n; i++)
```

```
    for (j = 0; j < m; j++)
```

```
        scanf("%d", &Allocation[i][j]);
```

```
printf("Enter Maximum matrix:\n");
```

```
for (i = 0; i < n; i++)
```

```
    for (j = 0; j < m; j++)
```

```
        scanf("%d", &Maximum[i][j]);
```

```
printf("Enter Available resources:\n");
```

```
for (j = 0; j < m; j++)
```

```
    scanf("%d", &Available[j]);
```

```
for (i = 0; i < n; i++)
```

```
    for (j = 0; j < m; j++)
```

Need[i][j] = Maximum[i][j] - Allocation[i][j];

bool Finish[n];

for (i = 0; i < n; i++)

Finish[i] = false;

int SafeSequence[n], work[m];

for (j = 0; j < m; j++)

work[j] = Available[j];

int count = 0;

while (count < n) {

bool found = false;

for (i = 0; i < n; i++) {

if (!Finish[i]) {

for (j = 0; j < m; j++)

if (Need[i][j] > work[j])

break;

if (j == m) {

for (k = 0; k < m; k++)

work[k] += Allocation[i][k];

SafeSequence[count++] = i;

```

        Finish[i] = true;

        found = true;

    }

}

}

if (!found) {

    printf("System is in an unsafe state.\n");

    return 0;

}

}

printf("System is in a safe state.\nSafe sequence is: ");

for (i = 0; i < n; i++)

    printf("%d ", SafeSequence[i]);

printf("\n");


return 0;

}

```

**Output:**

The screenshot displays the OnlineGDB interface. On the left is a sidebar with navigation links: 'Welcome, K Sai Krishna', 'Create New Project', 'My Projects', 'Classroom' (marked as new), 'Learn Programming', 'Programming Questions', 'Upgrade', and 'Logout'. The main area shows a C++ program in 'main.c' with the following code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5
6 #define BUFFER_SIZE 5
7
8 int buffer[BUFFER_SIZE];
9 int in = 0, out = 0; // Circular buffer indices
10
11 sem_t empty; // Semaphore to track empty slots
12 sem_t full; // Semaphore to track filled slots
13 pthread_mutex_t mutex; // Mutex to protect buffer
14
15 void *producer(void *arg) {
16     int item;
17     for (int i = 0; i < 10; i++) {
18         item = rand() % 100; // Generate a random item
19         sem_wait(&empty); // Wait if no empty slot
20         pthread_mutex_lock(&mutex); // Lock the buffer
21         buffer[in] = item;
22         in = (in + 1) % BUFFER_SIZE;
23         pthread_mutex_unlock(&mutex);
24         printf("Producer produced: %d\n", item);
25     }
26 }
27
28 void *consumer(void *arg) {
29     for (int i = 0; i < 10; i++) {
30         pthread_mutex_lock(&mutex); // Lock the buffer
31         int item = buffer[out];
32         out = (out + 1) % BUFFER_SIZE;
33         pthread_mutex_unlock(&mutex);
34         printf("Consumer consumed: %d\n", item);
35     }
36 }
37
38 int main() {
39     pthread_t producer_thread, consumer_thread;
40     pthread_create(&producer_thread, NULL, producer, NULL);
41     pthread_create(&consumer_thread, NULL, consumer, NULL);
42     pthread_join(producer_thread, NULL);
43     pthread_join(consumer_thread, NULL);
44     return 0;
45 }
```

The output window shows the execution results:

```
Producer produced: 15
Producer produced: 93
Consumer consumed: 77
Producer produced: 35
Producer produced: 86
Consumer consumed: 15
Producer produced: 92
Producer produced: 49
Consumer consumed: 93
Producer produced: 21
Consumer consumed: 35
Consumer consumed: 86
Consumer consumed: 92
Consumer consumed: 49
Consumer consumed: 21
...Program finished with exit code 0
```

## Result:

- The producer generates random numbers and places them in the buffer.
- The consumer retrieves and consumes these numbers.
- The semaphores ensure that the producer and consumer operate without conflicts, and the buffer is accessed safely.