

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

AIM:

To construct a C program to simulate the Producer-Consumer problem using semaphores, ensuring synchronization between the producer and consumer processes while preventing race conditions and buffer overflows or underflows.

ALGORITHM:

1. Initialization:

- Define a shared buffer with a fixed size.
- Initialize three semaphores:
 - empty: Counts the number of available slots in the buffer (initially equal to the buffer size).
 - full: Counts the number of filled slots in the buffer (initially zero).
 - mutex: Ensures mutual exclusion for buffer access (initialized to 1).

2. Producer Process:

- Repeatedly execute the following steps:
 1. Wait (sem_wait) on the empty semaphore to ensure a free slot is available.
 2. Wait (sem_wait) on the mutex semaphore to gain exclusive access to the buffer.
 3. Produce an item and place it in the buffer.
 4. Signal (sem_post) the mutex semaphore to release the buffer.
 5. Signal (sem_post) the full semaphore to indicate a filled slot.

3. Consumer Process:

- Repeatedly execute the following steps:
 1. Wait (sem_wait) on the full semaphore to ensure a filled slot is available.
 2. Wait (sem_wait) on the mutex semaphore to gain exclusive access to the buffer.
 3. Remove an item from the buffer for consumption.
 4. Signal (sem_post) the mutex semaphore to release the buffer.
 5. Signal (sem_post) the empty semaphore to indicate a free slot.

4. Concurrent Execution:

- Create separate threads for the producer and consumer processes.
- Ensure both threads run concurrently and modify the shared buffer as per their respective logic.

5. Termination:

- Stop the producer and consumer threads after a predefined number of operations or based on user input.
- Destroy all semaphores to release system resources.

PROCEDURE:

1. Start:

Initialize necessary variables, shared buffer, and semaphores.

2. Define Semaphores:

- Create a semaphore empty initialized to the buffer size to track available slots.
- Create a semaphore full initialized to 0 to track filled slots.

- Create a semaphore mutex initialized to 1 to enforce mutual exclusion.
- 3. Define Shared Buffer:**
 - Set up a circular buffer with a fixed size.
 - Use in and out pointers to manage the producer and consumer operations.
- 4. Create Producer Thread:**
 - In the producer thread:
 - Wait on empty and mutex semaphores.
 - Produce an item and insert it into the buffer at the in index.
 - Update the in index to the next position in a circular manner.
 - Signal the mutex and full semaphores to indicate a successful operation.
- 5. Create Consumer Thread:**
 - In the consumer thread:
 - Wait on full and mutex semaphores.
 - Consume an item from the buffer at the out index.
 - Update the out index to the next position in a circular manner.
 - Signal the mutex and empty semaphores to indicate a successful operation.
- 6. Run Threads Concurrently:**
 - Execute both producer and consumer threads concurrently using pthread_create.
- 7. Synchronization:**
 - Ensure that both threads operate in sync by using semaphores to handle mutual exclusion and resource tracking.
- 8. Stop and Cleanup:**
 - Terminate the threads after a fixed number of operations.
 - Destroy the semaphores to release resources.
- 9. End:**

Stop the program after all operations are completed.

CODE:

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

#define MAX_BUFFER_SIZE 5
#define NUM_ITEMS 10

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

sem_t empty;
sem_t full;
sem_t mutex;

void *producer(void *arg) {
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        item = rand() % 100;
        sem_wait(&empty);
```

```

    sem_wait(&mutex);

    buffer[in] = item;
    printf("Producer produced: %d\n", item);
    in = (in + 1) % MAX_BUFFER_SIZE;

    sem_post(&mutex);
    sem_post(&full);
}
pthread_exit(NULL);
}

void *consumer(void *arg) {
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        sem_wait(&full);
        sem_wait(&mutex);

        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % MAX_BUFFER_SIZE; // Move to the next item

        sem_post(&mutex);
        sem_post(&empty);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, MAX_BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);


    pthread_join(prod, NULL);
    pthread_join(cons, NULL);


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

    return 0;
}

```

OUTPUT:

**OnlineGDB**
online compiler and debugger for c/c++

[Welcome, Siva Shirish](#) 

[Create New Project](#)


[My Projects](#)











[Classroom](#) new

[Learn Programming](#)







[Programming Questions](#)

[Upgrade](#)


[Logout](#) 

 Run  Debug  Stop  Share  Save  Beautify  

main.c



input



Producer produced: 83

Consumer consumed: 83

Producer produced: 86

Producer produced: 77

Producer produced: 15

Producer produced: 93

Producer produced: 35

Consumer consumed: 86

Consumer consumed: 77

Consumer consumed: 15

Consumer consumed: 93

Consumer consumed: 35

Producer produced: 86

Producer produced: 92

Producer produced: 49

Producer produced: 21

Consumer consumed: 86

Consumer consumed: 92

Consumer consumed: 49

Consumer consumed: 21