Ex. No.: 8
Date:

PRODUCER CONSUMER USING SEMAPHORES

Aim: To write a program to implement solution to producer consumer problem using semaphores.

Algorithm:

1. Initialize semaphore empty, full and mutex.
2. Create two threads- producer thread and consumer thread.
3. Wait for target thread termination.
4. Call sem_wait on empty semaphore followed by mutex semaphore before entry into critical section.
5. Produce/Consume the item in critical section.
6. Call sem_post on mutex semaphore followed by full semaphore
7. before exiting critical section.
8. Allow the other thread to enter its critical section.
9. Terminate after looping ten times in producer and consumer Threads each.

Program Code:

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

#define BUFFER_SIZE 3
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
int next_item = 1;  // Next item number to produce
int items_in_buffer = 0;  // Current items in buffer
sem_t empty;
sem_t full;
sem_t mutex;

void produce() {
    if (sem_trywait(&empty) == 0) {
        sem_wait(&mutex);

        buffer[in] = next_item;
        printf("Producer produces the item:%d\n", next_item);
        in = (in + 1) % BUFFER_SIZE;
        items_in_buffer++;

        // Only increment next_item if buffer wasn't empty before
        if (items_in_buffer >= 1) {
            next_item++;
        } else {
```

```
            // If buffer was empty, keep next_item as is (it's already 1)
        }

        sem_post(&mutex);
        sem_post(&full);
    } else {
        printf("Buffer is full!!\n");
    }
}

void consume() {
    if (sem_trywait(&full) == 0) {
        sem_wait(&mutex);

        int item = buffer[out];
        printf("Consumer consume product %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        items_in_buffer--;

        // Reset counter when buffer becomes empty
        if (items_in_buffer == 0) {
            next_item = 1;
        }

        sem_post(&mutex);
        sem_post(&empty);
    } else {
        printf("Buffer is empty!!\n");
    }
}

int main() {
    int choice;

    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    // Initialize buffer to 0
    for (int i = 0; i < BUFFER_SIZE; i++) {
        buffer[i] = 0;
    }

    do {
        printf("1. Produce\n2. Consume\n3. Exit\n");
        printf("Enter choice:");
```

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

        switch(choice) {
            case 1:
                produce();
                break;
            case 2:
                consume();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while(choice != 3);

    // Clean up
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
    }
```

Sample Output:

1. Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item
1 Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3

Result:

The producer-consumer problem has been successfully solved using semaphores, demonstrating the ability to synchronize processes and manage shared resources