**Sachin Chaudhary**

**230701402  
EX No.8  
 Producer Consumer Problem**Aim:

To write a program that implements a solution to the producer-consumer problem using semaphores.

Algorithm:

Initialize semaphore empty, full, and mutex.

Create two threads: producer thread and consumer thread.

Wait for target thread termination.

Call sem\_wait on the empty semaphore followed by the mutex semaphore before entering the critical section.

Produce/consume the item in the critical section.

Call sem\_post on the mutex semaphore followed by the full semaphore before exiting the critical section.

Allow the other thread to enter its critical section.

Terminate after looping ten times in both producer and consumer threads.

Program Code:

producer\_consumer.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

int in = 0, out = 0;

sem\_t empty, full, mutex;

void\* producer(void\* arg) {

int item;

for (int i = 0; i < 10; i++) {

printf("\nEnter your choice: ");

scanf("%d", &item);

if (item == 1) { // Produce item

sem\_wait(&empty); // Decrement empty semaphore

sem\_wait(&mutex); // Enter critical section

buffer[in] = i + 1; // Produce an item

printf("Producer produces the item %d\n", buffer[in]);

in = (in + 1) % BUFFER\_SIZE; // Update in pointer

sem\_post(&mutex); // Exit critical section

sem\_post(&full); // Increment full semaphore

} else if (item == 3) {

break; // Exit the producer thread

}

}

pthread\_exit(0);

}

void\* consumer(void\* arg) {

int item;

for (int i = 0; i < 10; i++) {

printf("\nEnter your choice: ");

scanf("%d", &item);

if (item == 2) { // Consume item

sem\_wait(&full); // Decrement full semaphore

sem\_wait(&mutex); // Enter critical section

if (in == out) {

printf("Buffer is empty!!\n");

} else {

printf("Consumer consumes item %d\n", buffer[out]);

out = (out + 1) % BUFFER\_SIZE; // Update out pointer

}

sem\_post(&mutex); // Exit critical section

sem\_post(&empty); // Increment empty semaphore

} else if (item == 3) {

break; // Exit the consumer thread

}

}

pthread\_exit(0);

}

int main() {

pthread\_t producer\_thread, consumer\_thread;

// Initialize semaphores

sem\_init(&empty, 0, BUFFER\_SIZE); // Initially empty slots = BUFFER\_SIZE

sem\_init(&full, 0, 0); // Initially no items are full

sem\_init(&mutex, 0, 1); // Mutex is initially unlocked

// Create producer and consumer threads

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

// Wait for both threads to finish

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

// Destroy semaphores

sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&mutex);

return 0;

}

Sample Output:

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 2

Enter your choice: 1

Producer produces the item 3

Enter your choice: 1

Buffer is full!!

Enter your choice: 3

Result:

The program successfully implements the producer-consumer problem using semaphores. It ensures that the producer and consumer processes are synchronized and prevents buffer overflow or underflow. The program also demonstrates mutual exclusion and inter-thread communication.