**EX 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. This program demonstrates process synchronization where the producer adds items to a buffer and the consumer removes them, ensuring no overproduction or consumption occurs

**Algorithm:**

1. **Initialize Semaphores**:
   * mutex: Controls access to the shared buffer.
   * full: Counts filled slots in the buffer.
   * empty: Counts empty slots in the buffer.
2. **Producer Function**:
   * Wait on empty and mutex.
   * Produce an item and add it to the buffer.
   * Signal mutex and full.
3. **Consumer Function**:
   * Wait on full and mutex.
   * Remove an item from the buffer and consume it.
   * Signal mutex and empty.
4. **Create Threads**:
   * Create threads for producers and consumers.
   * Each thread executes the producer or consumer function.

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 5

#define NUM\_ITEMS 10

int buffer[BUFFER\_SIZE];

int in = 0;

int out = 0;

sem\_t empty;

sem\_t full;

pthread\_mutex\_t mutex;

void\* producer(void\* arg) {

for (int i = 0; i < NUM\_ITEMS; i++) {

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = i;

printf("Produced: %d\n", buffer[in]);

in = (in + 1) % BUFFER\_SIZE;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

sleep(1);

}

return NULL;

}

void\* consumer(void\* arg) {

for (int i = 0; i < NUM\_ITEMS; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumed: %d\n", item);

out = (out + 1) % BUFFER\_SIZE;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

sleep(1);

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

pthread\_mutex\_init(&mutex, NULL);

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

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

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

sem\_destroy(&empty);

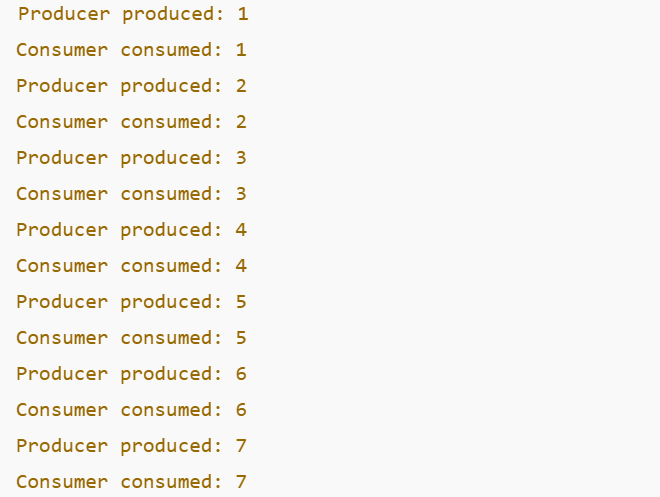
sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

**OUTPUT:**

****