**NAME : MARIYAM PATEL**

**SRN:202201178**

**ROLL NO:34**

**ASSIGNMENT-5**

Q.Solve the Producers-Consumers problem:

a. using threads and semaphores -

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <stdlib.h> // Include for rand()

#define BUFFER\_SIZE 5

#define NUM\_ITEMS 10 // Define the number of items to produce/consume

int buffer[BUFFER\_SIZE];

int count = 0;

sem\_t empty;

sem\_t full;

pthread\_mutex\_t mutex;

void \*producer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Produce only 10 items

sleep(1);

item = rand() % 100;

sem\_wait(&empty); // Wait if buffer is full, Waits for an empty slot

pthread\_mutex\_lock(&mutex);

// Produce item

buffer[count++] = item;

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&full); // Signal that buffer is not empty

}

return NULL;

}

void \*consumer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Consume only 10 items

sleep(1);

sem\_wait(&full); // Wait if buffer is empty, Waits for a filled slot

pthread\_mutex\_lock(&mutex);

// Consume item

item = buffer[--count];

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty); // Signal that buffer is not full

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, BUFFER\_SIZE); // Initialize semaphore to the size of buffer

sem\_init(&full, 0, 0); // Initialize semaphore to 0 (no items initially)

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

// Create producer and consumer threads

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

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

// Wait for both threads to complete

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

// Clean up resources

pthread\_mutex\_destroy(&mutex);

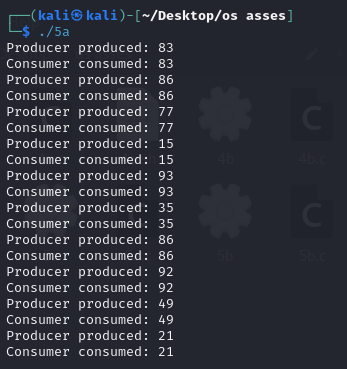
sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

OUTPUT-



b. using threads and mutex-

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <stdlib.h> // Include for rand()

#define BUFFER\_SIZE 5

#define NUM\_ITEMS 10 // Number of items to produce/consume

int buffer[BUFFER\_SIZE];

int count = 0; // Tracks the number of items in the buffer

pthread\_mutex\_t mutex; // Mutex to protect shared buffer

void \*producer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Produce only 10 items

sleep(1);

item = rand() % 100; // Generate a random item

// Critical section

pthread\_mutex\_lock(&mutex); // Lock the mutex before accessing the buffer

if (count < BUFFER\_SIZE) { // Only produce if buffer is not full

buffer[count++] = item;

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

} else {

printf("Buffer is full, producer is waiting...\n");

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex after producing the item

}

return NULL;

}

void \*consumer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Consume only 10 items

sleep(1);

// Critical section

pthread\_mutex\_lock(&mutex); // Lock the mutex before accessing the buffer

if (count > 0) { // Only consume if buffer is not empty

item = buffer[--count];

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

} else {

printf("Buffer is empty, consumer is waiting...\n");

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex after consuming the item

}

return NULL;

}

int main() {

pthread\_t prod, cons;

// Initialize the mutex

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

// Create producer and consumer threads

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

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

// Wait for both threads to finish

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

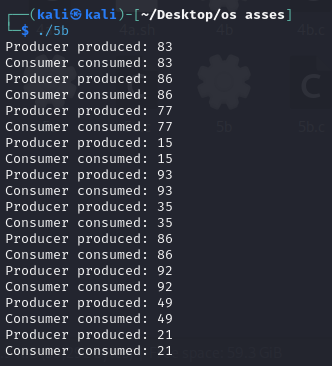
// Destroy the mutex

pthread\_mutex\_destroy(&mutex);

return 0;

}

OUTPUT-



**Easy codes-**

**A:** using threads and semaphores -

#!/bin/bash

# Create a temporary file to act as a lock

LOCK\_FILE="/tmp/con\_pro.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_CON=5

MAX\_PRO=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

consumer() {

local id=$1

for ((i = 1; i <= MAX\_CON; i++)); do

(

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Consumer $id: consumed = $data (iteration $i)"

) 200<$LOCK\_FILE

sleep 1

done

}

# Function for writer

producer() {

local id=$1

for ((i = 1; i <= MAX\_PRO; i++)); do

(

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Producer $id: produced = $data (iteration $i)"

) 200>$LOCK\_FILE

sleep 2

done

}

# Start readers and writers

for i in {1..3}; do

consumer $i &

done

for i in {1..2}; do

producer $i &

done

# Wait for all background processes to finish

Wait

**B:** using threads and mutex-

#!/bin/bash

# Create a temporary lock file

MUTEX="/tmp/mutex.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_CON=5

MAX\_PRO=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

consumer() {

local id=$1

for ((i = 1; i <= MAX\_CON; i++)); do

# Readers only acquire a shared lock

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Consumer $id: consumed = $data (iteration $i)"

flock -u 200

sleep 1

done 200<"$MUTEX"

}

# Function for writer

producer() {

local id=$1

for ((i = 1; i <= MAX\_PRO; i++)); do

# Writers need exclusive access

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Producer $id: produced to $data (iteration $i)"

flock -u 200

sleep 2

done 200>"$MUTEX"

}

# Start readers and writers

for i in {1..3}; do

consumer $i &

done

for i in {1..2}; do

producer $i &

done

# Wait for all background processes to finish

wait