// Implementation of classical problems ( producer consumer ) using threads & semaphore

#include <stdio.h>package lab6;

import java.util.LinkedList;

import java.util.concurrent.Semaphore;

class ProducerConsumer {

private final LinkedList<Integer> buffer = new LinkedList<>();

private final int capacity = 5; // Size of the buffer

private final Semaphore mutex; // Mutual exclusion for buffer access

private final Semaphore empty; // Semaphore to count empty slots in buffer

private final Semaphore full; // Semaphore to count full slots in buffer

public ProducerConsumer() {

mutex = new Semaphore(1); // Binary semaphore for mutual exclusion

empty = new Semaphore(capacity); // Counting semaphore for empty slots

full = new Semaphore(0); // Counting semaphore for full slots

}

// Producer class

class Producer implements Runnable {

@Override

public void run() {

int item = 0; // Item to be produced

while (true) {

try {

empty.acquire(); // Wait if buffer is full

mutex.acquire(); // Protect buffer access

// Critical Section for Producer (Inserting item)

buffer.add(item);

System.out.println("Produced: " + item);

item++; // Increment item to produce new one

mutex.release(); // Release buffer access

full.release(); // Signal that there's a full slot now

// Simulate production time

Thread.sleep(1000);

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

// Consumer class

class Consumer implements Runnable {

@Override

public void run() {

while (true) {

try {

full.acquire(); // Wait if buffer is empty

mutex.acquire(); // Protect buffer access

// Critical Section for Consumer (Removing item)

int item = buffer.removeFirst();

System.out.println("Consumed: " + item);

mutex.release(); // Release buffer access

empty.release(); // Signal that there's an empty slot now

// Simulate consumption time

Thread.sleep(1500);

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

}

}

}

public static void main(String[] args) {

ProducerConsumer pc = new ProducerConsumer();

// Creating producer and consumer threads

Thread producer1 = new Thread(pc.new Producer());

Thread producer2 = new Thread(pc.new Producer());

Thread producer3 = new Thread(pc.new Producer());

Thread consumer1 = new Thread(pc.new Consumer());

Thread consumer2 = new Thread(pc.new Consumer());

// Start threads

producer1.start();

producer2.start();

producer3.start();

consumer1.start();

consumer2.start();

}

}