Aquino, Kurt Neil CSC611M

Choy, Matthew Seaver AdvanOS

Matias, Angelo

Yu Galan, Stanley

Homework

Problems Using Semaphores and Monitors

1. **Sleeping Barber - Semaphore**

public class SleepingBarberSemaphore extends Thread {

public static Semaphore customers = new Semaphore(0);

public static Semaphore barber = new Semaphore(0);

public static Semaphore accessSeats = new Semaphore(1);

public static final int CHAIRS = 3;

public static int numberOfFreeSeats = CHAIRS;

class Customer extends Thread {

public void run() {

try {

accessSeats.acquire();

if (numberOfFreeSeats > 0) {

System.out.println("Customer just sat down.");

numberOfFreeSeats--;

customers.release();

//accessSeats.release();

barber.acquire();

this.getHaircut();

} else {

System.out.println("No waiting room space, the customer left");

accessSeats.release();

}

} catch (InterruptedException ex)

}

public void getHaircut(){}

}

class Barber extends Thread {

public void run() {

while (true) {

try {

customers.acquire();

accessSeats.release();

numberOfFreeSeats++;

barber.accessSeats();

release.release();

cutHair();

} catch (InterruptedException ex)

}

}

public void cutHair(){}

}

}

1. **Sleeping Barber – Monitor**

public class SleepingBarberMonitor {

public static void main(String a[]) {

Bshop shop = new Bshop();

Barber barber = new Barber(shop);

CustomerGenerator cg = new CustomerGenerator(shop);

Thread barberThread = new Thread(barber);

Thread customerSpawner = new Thread(cg);

customerSpawner.start();

barberThread.start();

}

}

class Barber implements Runnable {

Bshop shop;

public Barber(Bshop shop) {

this.shop = shop;

}

public void run() {

System.out.println("Barberbot initialized");

while(true) {

shop.cutHair();

}

}

}

class Customer implements Runnable {

Bshop shop;

public Customer(Bshop shop) {

this.shop = shop;

}

public void run() {

goForHairCut();

}

private synchronized void goForHairCut() {

shop.add(this);

}

}

class CustomerGenerator implements Runnable {

Bshop shop;

public CustomerGenerator(Bshop shop) {

this.shop = shop;

}

public void run() {

while(true) {

Customer customer = new Customer(shop);

Thread thcustomer = new Thread(customer);

thcustomer.start();

try {

TimeUnit.SECONDS.sleep((long)(Math.random()\*10));

} catch(InterruptedException iex) {

iex.printStackTrace();

}

}

}

}

class Bshop {

int nchair;

List<Customer> listCustomer;

public Bshop() {

nchair = 3;

listCustomer = new LinkedList<Customer>();

}

public void cutHair() {

Customer customer;

synchronized (listCustomer) {

while(listCustomer.size()==0) {

System.out.println("Barber fell asleep");

try {

listCustomer.wait();

} catch(InterruptedException iex) {

iex.printStackTrace();

}

}

System.out.println("Barber found a customer in the queue.");

customer = (Customer)((LinkedList<?>)listCustomer).poll();

}

System.out.println("Yay a haircut has been completed");

}

public void add(Customer customer) {

synchronized (listCustomer) {

if(listCustomer.size() == nchair) {

System.out.println("No chair, customer left to get a taco");

return ;

}

((LinkedList<Customer>)listCustomer).offer(customer);

System.out.println("Customer entered the waiting room"

+ "");

if(listCustomer.size()==1)

listCustomer.notify();

}

}

}

1. **Roller Coaster – Semaphore**

public class RollerCoasterSemaphore {

private final Semaphore permissionToBoard = new Semaphore(0);

private final Semaphore allAboard = new Semaphore(0);

private final Semaphore permissionToUnload = new Semaphore(0);

private final Semaphore allDeparted = new Semaphore(0);

public synchronized void rollerCoaster() {

while(true) {

try {

board();

permissionToBoard.release();

allAboard.acquire();

runRollerCoaster();

unload();

permissionToUnload.release();

allDeparted.acquire();

} catch (InterruptedException ex) {}

}

}

public synchronized void passenger() {

while (true) {

try {

permissionToBoard.acquire();

board();

allAboard.release();

permissionToUnload.acquire();

unboard();

allDeparted.release();

} catch (InterruptedException ex) {}

}

}

public void unload(){}

public void board(){}

public void unboard(){}

public void runRollerCoaster(){}

}

1. **Roller Coaster – Monitor**

public class RollerCoasterMonitor {

int boarding = 0, riders = 0, unboarding = 0;

final int CAPACITY =5;

boolean loading = false, unloading = false;

public synchronized void Passenger(RollerCoasterMonitor coaster) {

while (true) {

coaster.passengerBoarding();

coaster.passengerUnboarding();

}

}

public synchronized void Car(RollerCoasterMonitor coaster) {

while (true) {

coaster.carBoarding();

//runCoaster(); // whee

coaster.carUnboarding();

}

}

public synchronized void passengerBoarding() {

while (boarding + riders+ unboarding == CAPACITY) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

++boarding;

if (boarding == CAPACITY) {

notify();

}

while (!loading) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

--boarding;

++riders;

if (riders == CAPACITY) {

notify();

}

}

public synchronized void carBoarding() {

while (boarding < CAPACITY) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

loading = true;

notifyAll();

while (riders < CAPACITY) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

loading = false;

}

public synchronized void passengerUnboarding() {

while (!unloading) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

--riders;

++unboarding;

if (unboarding == CAPACITY) {

notify();

}

}

public synchronized void carUnboarding() {

unloading = true;

notifyAll();

while (unboarding < CAPACITY) {

try {

wait();

} catch (InterruptedException ex) {

Logger.getLogger(RollerCoasterMonitor.class.getName()).log(Level.SEVERE, null, ex);

}

}

unloading = false;

unboarding = 0;

notifyAll();

}

}