Faculty of Computers and Artificial Intelligence
Cairo University
CS241: Operating System – 1

Assignment 2: Car Wash Simulation

Assignment description

In this assignment you are required to implement a classic **Producer-Consumer Problem**, using the Bounded Buffer pattern to simulate a busy Car Wash and Gas Station, taking into consideration managing the shared resources and preventing race conditions.

The simulation models a service station with a limited number of service bays and a fixed-size waiting area for vehicles.

- Each Car will enter a waiting queue fixed the waiting area depends on garage area. If the waiting area is full, the car must wait until space is available.
- A fixed number N of Pumps (or Service Bay) can work concurrently providing services. the pump only works when a car arrives.
- Queue sizes reported must always be ≥1 and ≤10
- Before starting service, the pump must acquire permission from the Service Bay.
- The pump releases the Service Bay upon completion, allowing another car to use a bay.
- Garage waiting and pumps numbers are inserted as input.

The output must clearly indicate shows all system activities, when a Car:

- Arrives
- Enters the queue
- Is taken by a Pump
- Starts service (Acquiring the Pumps)
- Finishes service (Releasing the Pumps)

Implementation Requirements

You must implement the following classes in Java:

1. ServiceStation (Main Class)

- Initializes the shared resources (Queue, Semaphores, Mutex, etc.)
- Creates and starts a background thread pool for the **Pumps** (Consumers)
- Creates and starts a continuous stream of **Car** (Producer) threads

2. Semaphore class

- As described in labs

3. Car (Producer)

- Extends Thread or implements Runnable

- Requires access to the Queue, the Mutex, and the Empty and Full Semaphores
- The run() method implements the logic described, ensuring correct semaphore.

3. Pump (Consumer)

- Extends Thread or implements Runnable.
- Requires access to the Queue, the Mutex, and ALL three semaphores (Empty, Full, and Pumps).
- The run() method implements the logic described, ensuring correct semaphore and mutex usage for resource access and queue modification.

Bonus:

Non-trivial Graphical User Interface (GUI) for the simulator, that shows the behavior of pumps when occupied or released by a particular car.

Running Example:

Sample Input

- Waiting area capacity: 5
- Number of service bays (pumps): 3
- Cars arriving (order): C1, C2, C3, C4, C5

Sample Output (log sequence)

- C1 arrived
- C2 arrived
- C3 arrived
- C4 arrived
- Pump 1: C1 Occupied
- Pump 2: C2 Occupied
- Pump 3: C3 Occupied
- C4 arrived and waiting
- C5 arrived
- C5 arrived and waiting
- Pump 1: C1 login
- Pump 1: C1 begins service at Bay 1
- Pump 2: C2 login
- Pump 2: C2 begins service at Bay 2
- Pump 3: C3 login
- Pump 3: C3 begins service at Bay 3
- Pump 1: C1 finishes service
- Pump 1: Bay 1 is now free
- Pump 2: C2 finishes service
- Pump 2: Bay 2 is now free
- Pump 1: C4 login
- Pump 1: C4 begins service at Bay 1

- Pump 3: C3 finishes service
- Pump 3: Bay 3 is now free
- Pump 2: C5 login
- Pump 2: C5 begins service at Bay 2
- Pump 1: C4 finishes service
- Pump 1: Bay 1 is now free
- Pump 3: C3 finishes service
- Pump 3: Bay 3 is now free
- Pump 2: C5 finishes service
- Pump 2: Bay 2 is now free
- All cars processed; simulation ends

Submission Rules:

1. One team member should submit only **one ".java"** file containing the source code
2. The submitted file name must follow this format: ID1_ID2_ID3_Group.java (e.g. 20210000_20210001_20210002_DS1.java)
3. The assignment is submitted in groups of **5**
4. Team Members MUST be within the **same group**
5. Cheating is totally prohibited and won't be tolerated (any similarity between your code and any other source will be assigned **NEGATIVE** without argument)
6. The deadline for submission **is 7th of November**
7. **No late** submissions

Grading criteria:

ServiceStation	**10**
Semaphore	**10**
Car	**15**
Pump	**15**
Valid output	**20**
GUI (Bonus)	**10**

Best Luck