010

Condition Variables

Operating Systems

Lab010 Condition Variables

(Zombie Bridge Crossing Challenge)

Objective

This lab challenges students to apply concepts of thread synchronization using locks and condition variables by simulating a "real-world" concurrency problem. The goal is to safely manage the flow of multiple people (threads) across a unidirectional bridge with a capacity limit. Students will implement logic to coordinate access to the bridge in a way that ensures both safety and fairness.

Scenario – The Zombie Outbreak

The university has been unexpectedly overrun by a horde of hungry zombies. Fortunately, agents from a mysterious organization called *Umbrella* have sealed the perimeter, trapping the zombies inside. Despite the chaos, the Operating Systems students continue attending class to complete their OS assignments.
To reach the classroom safely, they have constructed a narrow bridge from the entrance gate to the classroom. The bridge can only support traffic in **one direction at a time** and can hold **a maximum of 4 people** at once. Exceeding this limit risks collapsing the bridge — and feeding the zombies waiting below.

Assignment Tasks

1. Simulate student threads:

- Each student is a thread that waits for a random time (0–5s) before attempting to cross.
- Assign a crossing direction randomly: 0 (to the right) or 1 (to the left).

2. Implement a thread-safe bridge structure:

- Include accessBridge(direction) and exitBridge() methods.
- Use condition variables and locks to handle access control and synchronization.

3. Enforce bridge constraints:

- Allow only 4 students on the bridge at any given time.
- Only allow students crossing in the same direction together.
- Students require between 1 and 3 seconds to cross the bridge.
- If other students are crossing in the same direction and capacity allows, they may join the crossing.
- When a student finishes crossing, they signal others waiting to attempt access.

4. Log activity:

- Log arrival, entry, and exit from the bridge.
- Include the current queue status and crossing direction.

Print messages similar to the sample output provided below.

```
Inge 07 arrives wanting to go Left
Inge 07 crosses to the Left (on bridge: 1)
Inge 05 arrives wanting to go Left
Inge 05 crosses to the Left (on bridge: 2)
Inge 04 arrives wanting to go Right
Inge 08 arrives wanting to go Left
Inge 08 crosses to the Left (on bridge: 3)
Inge 01 arrives wanting to go Left
Inge 01 crosses to the Left (on bridge: 4)
Inge 02 arrives wanting to go Left
Inge 09 arrives wanting to go Left
Inge 07 exits bridge (on bridge: 3)
Inge 03 arrives wanting to go Right
Inge 10 arrives wanting to go Right
Inge 05 exits bridge (on bridge: 2)
Inge 06 arrives wanting to go Right
Inge 01 exits bridge (on bridge: 1)
Inge 08 exits bridge (on bridge: 0)
Inge 04 crosses to the Right (on bridge: 1)
Inge 06 crosses to the Right (on bridge: 2)
Inge 03 crosses to the Right (on bridge: 3)
Inge 10 crosses to the Right (on bridge: 4)
Inge 03 exits bridge (on bridge: 3)
Inge 04 exits bridge (on bridge: 2)
Inge 06 exits bridge (on bridge: 1)
Inge 10 exits bridge (on bridge: 0)
Inge 02 crosses to the Left (on bridge: 1)
Inge 09 crosses to the Left (on bridge: 2)
Inge 09 exits bridge (on bridge: 1)
Inge 02 exits bridge (on bridge: 0)
```

- Track and report the average waiting time of students before crossing.

Extra Points

- Implement a simple priority mechanism for students who've been waiting too long.
- Implement a starvation-prevention mechanism. Ensure that no direction waits indefinitely. If students on one side have been waiting for too long while others continue crossing in the opposite direction, implement logic to force a switch in direction once the bridge is empty, giving the waiting side a chance to cross.