A Semaphore is a synchronization primitive used in concurrent programming to control access to shared resources.

It acts as a counter with two main operations:

1. Acquire (Wait): Decreases the semaphore count. If the count becomes negative, the process/thread is blocked, waiting for the semaphore to become positive again.
2. Release (Signal): Increases the semaphore count. If there are waiting processes/threads, one of them is unblocked, allowing it to proceed.

Example of Semaphore:

Imagine a conference room with a capacity of 10 seats. A semaphore is used to represent the available seats:

- When a person enters the conference room, they call “Acquire” on the semaphore. If there are available seats (semaphore count > 0), they take a seat (decrement the count).
- If the conference room is full (semaphore count = 0), the next person who wants to enter the room will be blocked (wait) until someone inside the room leaves (releases a seat by calling “Release” on the semaphore).

This way, the semaphore ensures that only 10 people can be inside the conference room at any given time, preventing overcrowding and ensuring that everyone has a seat.