KARNAVATI UNIVERSITY

Operating Systems

Course Code: **71203002004**

Synchronization Tools & Critical Section Problem

by -
Minal Rajwar

Semaphores in Process Synchronization

- Semaphores help operating systems manage how processes share resources (like memory or devices) without conflicts.
- They ensure only a safe number of processes use a resource at the same time.

What is a Semaphore?

A semaphore is a synchronization tool used in concurrent programming to control access to shared resources.

- Works like a **lock** with a counter.
- Prevents issues like **race conditions** by controlling when processes can access data.

Operation in Semaphore

- **Wait (P)** → Decrease the semaphore value. If value is 0, the process waits until another process increases it.
- **Signal (V)** → Increase the semaphore value. This may unblock waiting processes.

If the initial value is N, up to N processes can pass the wait operation.

```
P(Semaphore s){
    while (S == 0);    /* wait until s = 0 */
  s = s - 1;
}

V(Semaphore s){
        s = s + 1;
}
```

Note that there is Semicolon after while. The code gets stuck Here while s is 0.

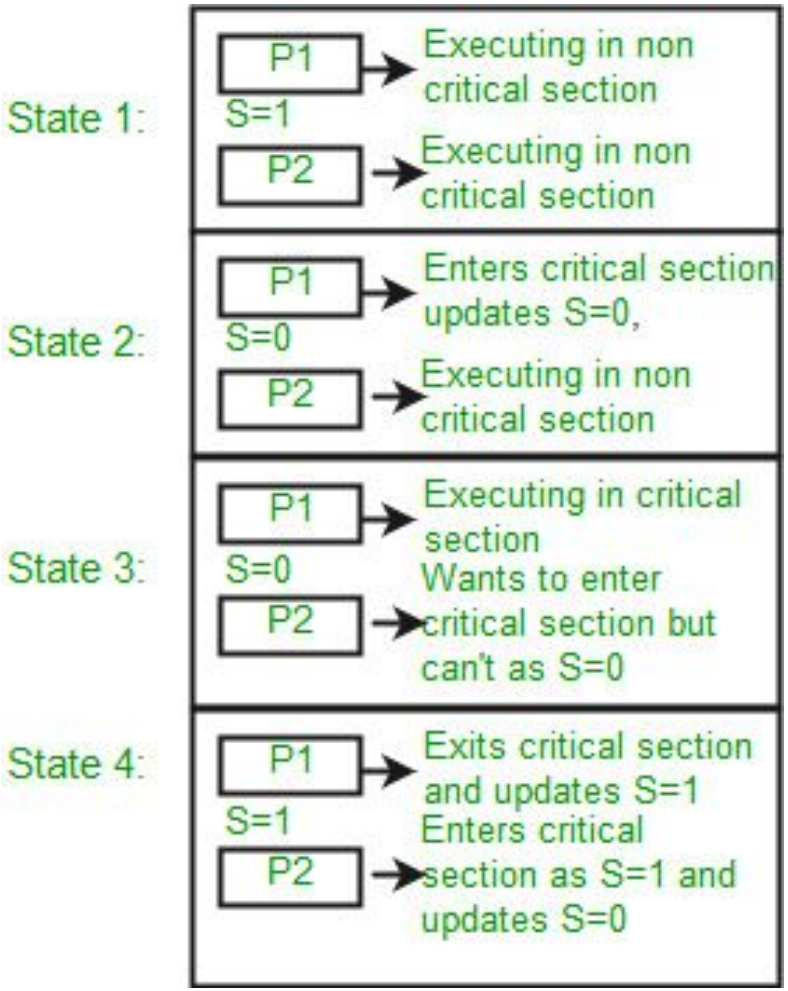
Types of Semaphore

- **Binary Semaphore (Mutex)** → Value 0 or 1; used for mutual exclusion (only one process at a time).
- **Counting Semaphore** → Value can be more than 1; used when multiple resource instances are available.

Types of Semaphore

How it Works

- Example: Semaphore $s = 1$.
  - If **P1** enters the critical section → s becomes 0.
  - If **P2** tries to enter → it waits until $s > 0$.
  - When P1 exits, it signals, and s becomes 1 again, allowing P2 to enter.



Uses

- **Mutual Exclusion** → Only one process accesses a resource at a time.
- **Process Synchronization** → Controls execution order.
- **Resource Management** → Limits use of finite resources (printers, devices).
- **Reader-Writer Problem** → Allows multiple readers but only one writer at a time.
- **Deadlock Avoidance** → Manages resource allocation order.

Advantages & Disadvantages

Advantages:

- Simple and effective.
- Supports coordination between processes.
- Prevents race conditions.
- Can help avoid deadlocks if used correctly.

Disadvantages:

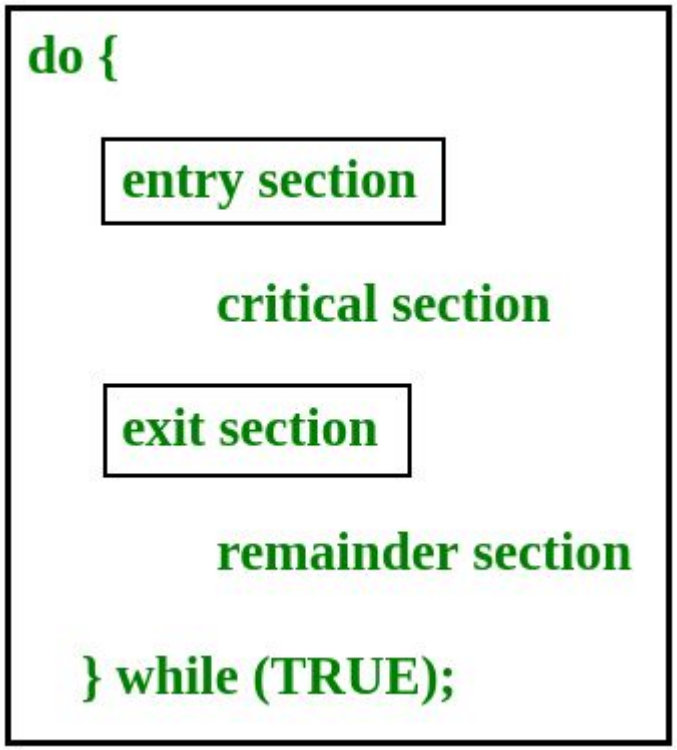
- Can cause performance issues due to overhead.
- Misuse may lead to **deadlock**.
- Can be hard to debug and maintain.
- May still cause race conditions if not used properly.
- Can be vulnerable to denial-of-service attacks.

Critical Section

- A critical section is a part of code that can be accessed by only one process at a time because it uses shared variables/resources.
- The critical section problem is finding a way for processes to share resources without causing data errors.

Example:

In banking, updating the balance variable in deposit or withdraw functions should be done in a critical section.



Requirement for solutions

- **Mutual Exclusion** → Only one process in the critical section at a time.
- **Progress** → If no process is in the critical section, waiting processes should decide fairly who goes next.
- **Bounded Waiting** → A process should not wait forever to enter the critical section.

Classic IPC Problems

Producer–Consumer Problem

- **Producer** → Creates data and puts it into a shared buffer.
- **Consumer** → Takes data from the buffer.

Problems:

- **Buffer Overflow** → Producer adds when buffer is full → must wait.
- **Buffer Underflow** → Consumer takes when buffer is empty → must wait.

Solution: Use **semaphores** to control access.

Classic IPC Problems

Readers–Writers Problem

- **Readers** → Only read data (no changes).
- **Writers** → Change the shared data.

Rules:

- Many readers can read at the same time.
- Only one writer at a time.
- No reading while writing is happening.

Classic IPC Problems

Dining Philosophers Problem

- 5 philosophers sit at a table with forks between them.
- To eat → a philosopher needs two forks (left and right).
- **Problem:** If each takes one fork and waits for the other → deadlock occurs.
- **Solution:** Use semaphores or rules to prevent deadlock and starvation.

Advantages & Disadvantages

Advantages:

- Maintains **data consistency**.
- Avoids **race conditions**.
- Prevents data corruption.
- Fair and efficient use of resources.

Disadvantages:

- **Extra overhead** → slows the system.
- Can reduce performance.
- Increases complexity.
- Risk of **deadlocks** if done incorrectly.

DISCUSSION & REVISION

1. What is the code segment accessed by only one process at a time called?
2. Which requirement ensures only one process is in the critical section at a time?
3. In the Producer–Consumer problem, what happens when the buffer is empty?
4. In the Readers–Writers problem, who modifies the shared data?
5. In the Dining Philosophers problem, what do philosophers need to eat?
6. What error occurs when each philosopher holds one fork and waits for another?

REFERENCES

1. https://www.geeksforgeeks.org/operating-systems/semaphores-in-process-synchronization/
2. https://www.geeksforgeeks.org/operating-systems/introduction-of-process-synchronization/