SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department Of Artificial Intelligence and Data Science

Course Code & Name –23ITB203 & Operating Systems

II Year / IV Semester

Unit 2 - Semaphores

Semaphores

- Semaphore proposed by Edsger Djikstra, is a technique to manage concurrent processes by using a simple integer value, which is known as a semaphore.
- Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multi processing environment.
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal()

wait() → P [from the Dutch word proberen, which means to “to test”]

signal() → V [from the Dutch word verhogen, which means “to increment”]

Definition of wait();

```
P (Semaphore S) {
          while (S<=0)
          ; // no operation
S- -;
}
```

Definition of signal();

```
V (Semaphore S)
          S++;
}
```

All the modifications to the integer value of the semaphore in the wait() and signal() operations must be executed indivisibly. That is, when one Process modifies the semaphore value, no other process can simultaneously Modify that same semaphore value.

Types of Semaphores

- **Binary Semaphore:**
  The value of a binary semaphore can range only between 0 and 1. On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.
- **Counting semaphore:**
  Its value can range over an unrestricted domain.
  It is used to control access to a resource that has multiple instances.

Disadvantage of Semaphores

- The main disadvantage of semaphore definition that was discussed is that it requires busy waiting.
- While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code.
- Busy waiting wastes CPU cycles that some other process might be able to use productively.
- This type of semaphore is called a spinlock because the process “spins” while waiting for the lock.

To overcome the need for busy waiting, we can modify

the definition of the **wait() and signal()** semaphore operations.

- When a process executes the **wait()** operation and finds that the semaphore value is not positive, it must wait.
- However, rather than engaging in **busy waiting**, the process can block itself.
- The **block operation** places a process into waiting queue associated with the semaphore, and the state of the process is switched to the waiting state.
- The control is transferred to the **CPU scheduler**, which selects another process to execute.

$\rightarrow$

$\rightarrow$

Deadlocks and starvation

- The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
- The event in question is the execution of a signal() operation. When such a state is reached, the processes are said to be deadlocked.

P0	P1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
.	.
signal(S);	signal(Q);
signal(Q);	signal(S);

- Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended

- Priority Inversion - Scheduling problem when lower-priority process holds a lock needed by higher-priority process