Mutex Locks

- In this approach in the entry section of the code the lock is acquired over critical resources which are modified and used inside the critical section. In the exit section the lock is released.
- As the resource is locked while a process executes its critical section hence no other process can access it.

Semaphores

- Semaphore was proposed by Dijkstra in 1965 which is a very significant 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 multiprocessing environment.
- Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, ***wait*** and ***signal*** that are used for process synchronization.

The definitions of wait and signal are as follows −

- **Wait**
  The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
  while (S<=0);

  S--;
}
```

- **Signal**
  The signal operation increments the value of its argument S.

```
signal(S)
{
  S++;
}
```

Semaphores are of two types:

1. **Binary Semaphore –** This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.

2. **Counting Semaphore –** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances. The semaphore count is the number of available resources. If the resources are added, semaphore count is automatically incremented and if the resources are removed, the count is decremented.

Advantages of Semaphores

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.

Disadvantages of Semaphores

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for large scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.