Semaphores

The Semaphore is just a normal integer. The Semaphore cannot be negative. The least value for a Semaphore is zero (0). The Maximum value of a Semaphore can be anything. The Semaphores usually have two operations. The two operations have the capability to decide the values of the semaphores.

The two Semaphore Operations are:

1. Wait ( )
2. Signal ( )

Wait Semaphore Operation

The Wait Operation is used for deciding the condition for the process to enter the critical state or wait for execution of process. Here, the wait operation has many different names. The different names are:

1. Sleep Operation
2. Down Operation
3. Decrease Operation
4. P Function (most important alias name for wait operation)
5. The Wait Operation works on the basis of Semaphore or Mutex Value.
6. Here, if the Semaphore value is greater than zero or positive then the Process can enter the Critical Section Area.
7. If the Semaphore value is equal to zero then the Process has to wait for the Process to exit the Critical Section Area.
8. This function is only present until the process enters the critical state. If the Processes enters the critical state, then the P Function or Wait Operation has no job to do.
9. If the Process exits the Critical Section we have to reduce the value of Semaphore

Signal Semaphore Operation

The Signal Semaphore Operation is used to update the value of Semaphore. The Semaphore value is updated when the new processes are ready to enter the Critical Section.

The Signal Operation is also known as:

1. Wake up Operation
2. Up Operation
3. Increase Operation
4. V Function (most important alias name for signal operation)
5. We know that the semaphore value is decreased by one in the wait operation when the process left the critical state. So, to counter balance the decreased number 1 we use signal operation which increments the semaphore value. This induces the critical section to receive more and more processes into it.
6. The most important part is that this Signal Operation or V Function is executed only when the process comes out of the critical section. The value of semaphore cannot be incremented before the exit of process from the critical section

## ypes of Semaphores

There are two types of Semaphores.

They are:

### 1. Binary Semaphore

Here, there are only two values of Semaphore in Binary Semaphore Concept. The two values are 1 and 0.

If the Value of Binary Semaphore is 1, then the process has the capability to enter the critical section area. If the value of Binary Semaphore is 0 then the process does not have the capability to enter the critical section area.

### 2. Counting Semaphore

Here, there are two sets of values of Semaphore in Counting Semaphore Concept. The two types of values are values greater than and equal to one and other type is value equal to zero.

If the Value of Binary Semaphore is greater than or equal to 1, then the process has the capability to enter the critical section area. If the value of Binary Semaphore is 0 then the process does not have the capability to enter the critical section area.

This is the brief description about the Binary and Counting Semaphores. You will learn still more about them in next articles.

Advantages

* Semaphores are machine independent since their implementation and codes are written in the microkernel's machine independent code area.
* They strictly enforce mutual exclusion and let processes enter the crucial part one at a time (only in the case of binary semaphores).
* With the use of semaphores, no resources are lost due to busy waiting since we do not need any processor time to verify that a condition is met before allowing a process access to the crucial area.
* Semaphores have the very good management of resources
* They forbid several processes from entering the crucial area. They are significantly more effective than other synchronization approaches since mutual exclusion is made possible in this way.

Disadvantages of a Semaphore

* Due to the employment of semaphores, it is possible for high priority processes to reach the vital area before low priority processes.
* Because semaphores are a little complex, it is important to design the wait and signal actions in a way that avoids deadlocks.
* Programming a semaphore is very challenging, and there is a danger that mutual exclusion won't be achieved.
* The wait ( ) and signal ( ) actions must be carried out in the appropriate order to prevent deadlocks.