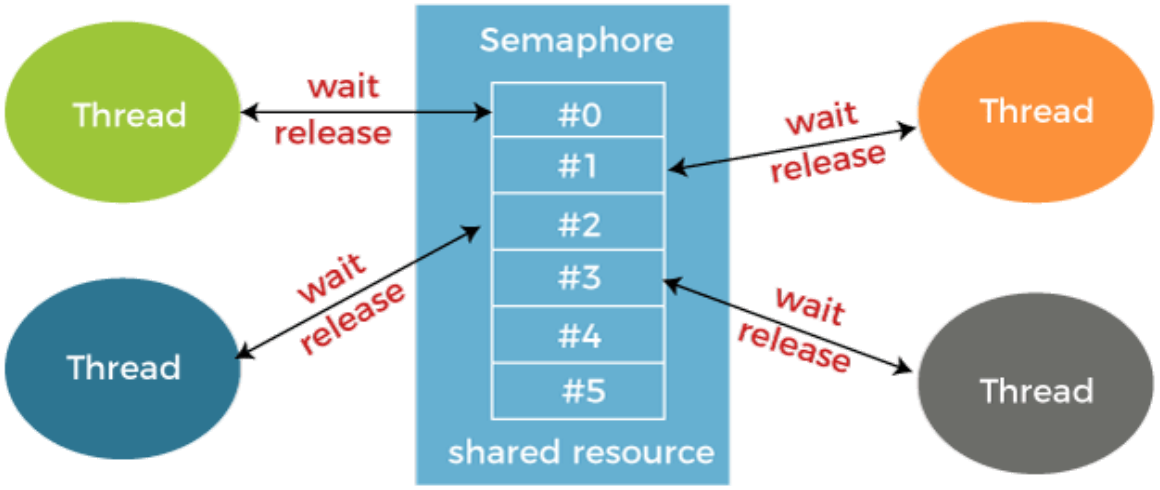
SEMAPHORES IN OPERATING SYSTEM

Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread.

It uses two atomic operations

1. Wait
2. Signal for the process synchronization

A semaphore either allows or disallows access to the resource, which depends on how it is set up.



CHARACTERISTIC OF SEMAPHORE

1. It is a mechanism that can be used to provide synchronization of tasks.
2. It is a low-level synchronization mechanism.
3. Semaphore will always hold a non-negative integer value.
4. Semaphore can be implemented using test operations and interrupts, which should be executed using file descriptors.

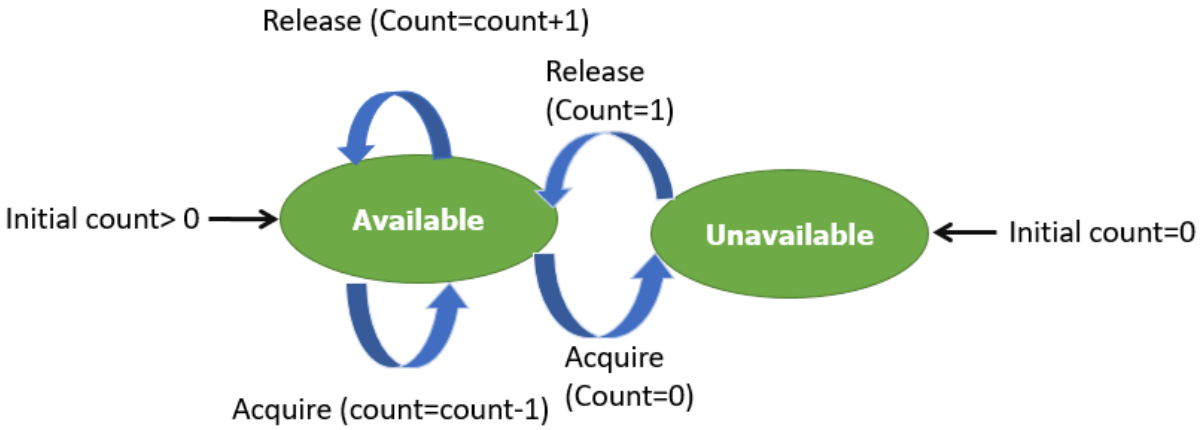
TYPES OF SEMAPHORES

The two common kinds of semaphores are

1. Counting semaphores
2. Binary semaphores

COUNTING SEMAPHORES

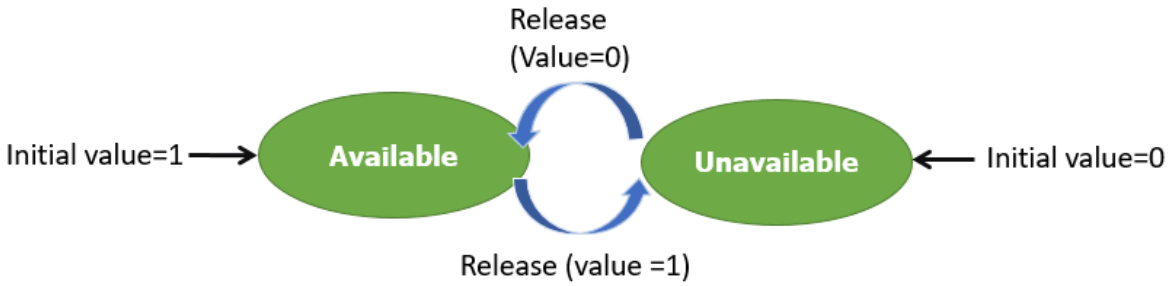
This type of Semaphore uses a count that helps task to be acquired or released numerous times. If the initial count = 0, the counting semaphore should be created in the unavailable state.



However, If the count is > 0, the semaphore is created in the available state, and the number of tokens it has equals to its count.

BINARY SEMAPHORES

The binary semaphores are quite similar to counting semaphores, but their value is restricted to 0 and 1. In this type of semaphore, the wait operation works only if semaphore = 1, and the signal operation succeeds when semaphore= 0. It is easy to implement than counting semaphores.



Here are some major differences between counting and binary semaphore

Counting Semaphore	**Binary Semaphore**
No mutual exclusion	Mutual exclusion
Any integer value	Value only 0 and 1
More than one slot	Only one slot

Provide a set of Processes	It has a mutual exclusion mechanism

Advantages of Semaphores

1. It allows more than one thread to access the critical section
2. Semaphores are machine-independent.
3. Semaphores are implemented in the machine-independent code of the microkernel.
4. They do not allow multiple processes to enter the critical section.
5. As there is busy waiting in semaphore, there is never a wastage of process time and resources.
6. They are machine-independent, which should be run in the machine-independent code of the microkernel.
7. They allow flexible management of resources.

Disadvantage of semaphores

1. One of the biggest limitations of a semaphore is priority inversion.
2. The operating system has to keep track of all calls to wait and signal semaphore.
3. Their use is never enforced, but it is by convention only.
4. In order to avoid deadlocks in semaphore, the Wait and Signal operations require to be executed in the correct order.
5. Semaphore programming is a complicated, so there are chances of not achieving mutual exclusion.
6. It is also not a practical method for large scale use as their use leads to loss of modularity.
7. Semaphore is more prone to programmer error.
8. It may cause deadlock or violation of mutual exclusion due to programmer error.