

# Mutex vs Semaphore

Mutex and Semaphore both provide synchronization services but they are not the same. Details about both Mutex and Semaphore are given below:

## Mutex

Mutex is a mutual exclusion object that synchronizes access to a resource. It is created with a unique name at the start of a program. The Mutex is a locking mechanism that makes sure only one thread can acquire the Mutex at a time and enter the critical section. This thread only releases the Mutex when it exits the critical section.

This is shown with the help of the following example:

```
wait (mutex);  
    ....  
Critical Section  
    ....  
signal (mutex);
```

A Mutex is different than a semaphore as it is a locking mechanism while a semaphore is a signalling mechanism. A binary semaphore can be used as a Mutex but a Mutex can never be used as a semaphore.

## Semaphore

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that called the wait function.

A semaphore uses two atomic operations, wait and signal for process synchronization.

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--;  
}
```

The signal operation increments the value of its argument S.

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

There are mainly two types of semaphores i.e. counting semaphores and binary semaphores.

Counting Semaphores are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources.

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.