Mutexes Vs Semaphores

Introduction

As we have learnt so far, process synchronization techniques play a key role in maintaining the consistency of shared data. The techniques that we have used so far are Locks, Conditional variables and Semaphores. Locks and Conditional variables use mutexes to implement their solutions, whereas Semaphores employ semaphore which is an integer variable.

One of the foundational differences between semaphore and mutex is that semaphore is an integer variable, on which process or threads have to perform wait() and signal() operations to point out whether the resource is being acquired or released, while Mutex is an object, on which process has to acquire the lock if it want to use the resource. Let us discuss the differences between semaphore and mutex in more detail in the next section.

Comparison

Key Differences Between Semaphore and Mutex:

1. Semaphore is a **signalling** mechanism as the process has to perform wait() and signal() operation on a semaphore variable to point out whether the resource is being acquired or released. On the other hand, the mutex is a **locking** mechanism. In the locking mechanism, the process has to acquire the lock on a mutex object if it wants to acquire the resource and similarly, it has to release the lock to make the resource available for usage.

2. Semaphore is typically an **integer** variable whereas mutex is an **object**.

3. Semaphore allows multiple program threads to access the **finite instance of resources**. On the other hand, Mutex allows multiple program threads to access a **single shared resource** but one at a time.

4. Semaphore variable value can be modified by **any** process that acquires or releases resource by performing wait() and signal() operation. On the other hand, lock acquired on the mutex object can be released **only** by the process that has acquired the lock on the mutex object.

5. Semaphore variable value is modified by **wait()** and **signal()** operation apart from initialization. However, the mutex object is locked or unlocked by the process of acquiring or releasing the resource.

6. If all the resources are acquired by the process, and no resource is free then the process desiring to acquire resource performs wait() operation on the semaphore variable and **blocks** itself till the count of semaphore becomes greater than 0. But if a mutex object is already locked then the process desiring to acquire resource **waits** and gets **queued** by the system till the resource is released and the mutex object gets unlocked.

7. Semaphore is useful when there are multiple instances of resource, while mutex is preferable when there is a single instance of the resource.