Thread Synchronization

Many threads trying to access the same object can lead to problems like making data inconsistent or getting unexpected output So When a thread is already accessing an object, preventing any other thread accessing the same object is called Thread Synchronization.

The object on which the threads are synchronized is called Synchronized Object or Mutually Exclusive Lock(mutex).

Thread Synchronization is recommended when multiple threads are acting on the same object simultaneously.

There are following techniques to do Thread Synchronization:

- Using Locks
- Using RLock (Re-Entrant Lock)
- Using Semaphores

Locks

Locks are typically used to synchronize access to a shared resource. Lock can be used to lock the object in which the thread is acting. A Lock has only two states, locked and unlocked. It is created in the unlocked state.

acquire( )

This method is used to changes the state to locked and returns immediately. When the state is locked, acquire() blocks until a call to release() in another thread changes it to unlocked, then the acquire() call resets it to locked and returns.

Syntax:- acquire(blocking=True, timeout = -1)

- True – It blocks until the lock is unlocked, then set it to locked and return True.
- False - It does not block. If a call with blocking set to True would block, return False immediately; otherwise, set the lock to locked and return True.
- Timeout - When invoked with the floating-point timeout argument set to a positive value, block for at most the number of seconds specified by timeout and as long as the lock cannot be acquired. A timeout argument of -1 specifies an unbounded wait. It is forbidden to specify a timeout when blocking is false.
- The return value is True if the lock is acquired successfully, False if not (for example if the timeout expired).

release( )

This method is used to release a lock. This can be called from any thread, not only the thread which has acquired the lock.

When the lock is locked, reset it to unlocked, and return. If any other threads are blocked waiting for the lock to become unlocked, allow exactly one of them to proceed.

When invoked on an unlocked lock, a RuntimeError is raised.

There is no return value.

Syntax:- release( )

RLock

A reentrant lock is a synchronization primitive that may be acquired multiple times by the same thread.

The standard Lock doesn’t know which thread is currently holding the lock. If the lock is held, any thread that attempts to acquire it will block, even if the same thread itself is already holding the lock. In such cases, RLock (re-entrant lock) is used.

A reentrant lock must be released by the thread that acquired it. Once a thread has acquired a reentrant lock, the same thread may acquire it again without blocking; the thread must release it once for each time it has acquired it.

Semaphore

This is one of the oldest synchronization primitives in the history of computer science, invented by the early Dutch computer scientist Edsger W. Dijkstra,

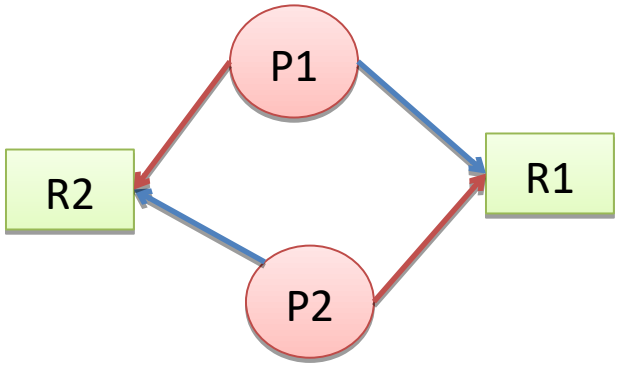
A semaphore manages an internal counter which is decremented by each acquire() call and incremented by each release() call.

The counter can never go below zero; when acquire() finds that it is zero, it blocks, waiting until some other thread calls release().

It’s usually better to use the BoundedSemaphore class, which considers it to be an error to call release more often than you’ve called acquire.

Dead Lock

A Deadlock is a situation where each of the process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.



Terrorist and army example