RLock

This lesson explains the reentrant lock in python's threading module.

Introduction

A reentrant lock is defined as a lock which can be reacquired by the same thread. A `RLock` object carries the notion of ownership. If a thread acquires a `RLock` object, it can chose to reacquire it as many times as possible. Consider the following snippet:

Reentrant lock

```
# create a reentrant lock
rlock = RLock()

# acquire the lock twice
rlock.acquire()
rlock.acquire()

# release the lock twice
rlock.release()
rlock.release()
```

In contrast to `Lock`, the reentrant lock is acquired twice in the above snippet without blocking. Note that it is imperative to release the lock as many times as it is locked, otherwise the lock remains in locked state and any other threads attempting to acquire the lock get blocked. This is shown with an example below:

```
1  from threading import RLock
2  from threading import Thread
3
4
5  def child_task():
6      rlock.acquire()
7      print("child task executing")
8      rlock.release()
9
10
11 rlock = RLock()
12
13 rlock.acquire()
14 rlock.acquire()
15
16 rlock.release()
17
18 # UNCOMMENT THE FOLLOWING LINE TO MAKE THE
19 # PROGRAM EXIT NORMALLY.
```

```
20  rlock.release()
21
22  thread = Thread(target=child_task)
```

Select lesson text to explore the Explain with AI Premium Offering

Run Save Reset

Output 0.65s

```
child task executing
```

If you uncomment **line 20** in the above code widget the child thread would be able to execute. We can have arbitrary nesting of acquiring and releasing of the reentrant lock. For instance, the following would work:

```
rlock = RLock()

rlock.acquire()
rlock.acquire()
rlock.acquire()
rlock.release()

rlock.acquire()
rlock.release()
rlock.release()
rlock.release()
```

The nested acquire/release calls are tracked internally by recursion level which is incremented on every `acquire()` and decremented on every `release()` by the same thread. When the recursion level is zero, the reentrant lock is in unlocked state.

Ownership

As explained, each reentrant lock is owned by some thread when in the locked state. Only the owner thread is allowed to exercise a `release()` on the lock. If a thread different than the owner invokes `release()` a RuntimeError is thrown as shown in the example below:

Releasing unowned reentrant lock

```
from threading import RLock
from threading import Thread


def perform_unlock():
```

```
    rlock.release()
    print("child task executing")
    rlock.release()
```



```
rlock = RLock()

# reentrant lock acquired by main thread
rlock.acquire()

# let's attempt to unlock using a child thread
thread = Thread(target=perform_unlock)
thread.start()
thread.join()
```

Recognize this isn't a problem with non-reentrant locks.

```
1  from threading import RLock
2  from threading import Thread
3
4
5  def perform_unlock():
6      rlock.release()
7      print("child task executing")
8      rlock.release()
9
10
11 rlock = RLock()
12
13 # reentrant lock acquired by main thread
14 rlock.acquire()
15
16 # let's attempt to unlock using a child thread
17 thread = Thread(target=perform_unlock)
18 thread.start()
19 thread.join()
20
```

Run Save Reset

Output 0.61s

```
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.5/threading.py", line 914, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.5/threading.py", line 862, in run
    self._target(*self._args, **self._kwargs)
  File "main.py", line 6, in perform_unlock
    rlock.release()
RuntimeError: cannot release un-acquired lock
```