Second Readers-Writers Problem (Writer Priority)

Amrith M

April 9, 2018

1 Readers-Writers Problem

In computer science, the readers-writers problems are examples of a common computing problem in concurrency. There are at least three variations of the problems, which deal with situations in which many threads try to access the same shared resource at one time. Some threads may read and some may write, with the constraint that no process may access the shared resource for either reading or writing while another process is in the act of writing to it. (In particular, it is allowed for two or more readers to access the share at the same time.) A readers-writer lock is a data structure that solves one or more of the readers-writers problems. Here we deal with the second problem which gives priority to the writers.

All the readers has to wait when a writer comes and concurrency is established between multiple readers.

2 Solution Using Semaphore

```
from threading import Thread

class Semaphore(object):

    def __init__(self, initial_value):
        self.n = initial_value

    def wait(self):
        while(self.n <= 0):
#           print ("----- Blocked ------")
            pass
        self.n -= 1

    def signal(self):
        self.n += 1
```

```
class Reader_Writer(object):
    def __init__(self):
        self.rw_mutex = Semaphore(1)
        self.read_count = 0
        self.mutex = Semaphore(1)
        self.shared_memory = []

    def reader(self):
        self.mutex.wait()
        self.read_count += 1
       # if (self.read_count == 1):
        self.mutex.signal()

        print ("Printing the contents of the list --> ", self.shared_memory)

        self.mutex.wait()
        self.read_count -= 1
        #if (self.read_count == 0):
        self.mutex.signal()

    def writer(self):
        self.mutex.wait()
        self.rw_mutex.wait()
        print ("Enter a string which is to be added to the shared memory:-")
        update = raw_input()
        self.shared_memory += [update]

        self.rw_mutex.signal()
        self.mutex.signal()
        print ("Shared Memory Updated")

def test_thread(obj, t_id, num_operations = 10):
    for i in range(num_operations):
        if (i%10 == 3):
            print ("Thread : ", t_id, " executing Writer")
            print
            obj.writer()
        else:
            print ("Thread : ", t_id, " executing Reader")
            print
            obj.reader()

if __name__ == "__main__":
    test_obj = Reader_Writer()
    thread1 = Thread(target = test_thread, args = (test_obj, 1, 10))
```

```python
thread2 = Thread(target = test_thread, args = (test_obj, 2, 10))
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print ("\n -------------- EXIT --------------- \n")
```