605.629: Programming Languages
Assignment 11
Sabbir Ahmed

November 22, 2021

1. [60 pts, concurrency]
   Consider the following Python program:

```
from threading import Thread, Lock, current_thread
from time import sleep

status, status_lock, count, count_lock = 1, Lock(), 1, Lock()

def fun1():
    global count, status
    while 1:
        with count_lock:
            print (current_thread().name, 'count lock acquired') # line 10
            count += 1
            with status_lock:
                print (current_thread().name, 'status lock acquired') # line 13
                status = count
                print (current_thread().name, 'updated status= %d' % (status))
                sleep(0.5)

def fun2():
    global count, status
    while 1:
        with status_lock:
            print (current_thread().name, 'status lock acquired') # line 22
            count += 1
            with count_lock:
                print (current_thread().name, 'count lock acquired') # line 25
                status = count
                print (current_thread().name, 'updated status= %d' % (status))
                sleep(0.5)

if __name__=="__main__":
    f1 = Thread(target=fun1)
    f2 = Thread(target=fun2)
    f1.start()
```

```
    f2.start()
```

(a.) Run it and observe that the program hangs shortly after it started running. Explain what is happening and what causes the hang?

Answer

Both Thread 1 and Thread 2 depend on acquiring 2 locks that are nested within each other at alternating orders. For the threads to continue, they have to acquire the first lock, increment the count, acquire the second lock then release both of the locks. The program hangs when the outer lock of both of the threads are available and acquired at the same time, i.e. when `Thread-1` acquires `count_lock` and `Thread-2` acquires `status_lock` simultaneously, both of the threads starves while waiting for their inner locks to be freed by the other thread which does not happen.

(b.) Comment lines 10, 13, 22 and 25 and run it again. Explain what is happening and why it stopped hanging? Or did it?

Answer

It did not stop hanging since the threads are still being starved. Commenting out the specified lines simply surpresses the output of the threads' statuses.

(c.) Explain how would you fix the problem.

Answer

Replacing the `status_lock` or the `count_lock` from the program with a single lock will allow the threads to increment the shared variable properly.

```
from threading import Thread, Lock, current_thread
from time import sleep

status, count, lock = 1, 1, Lock()


def fun1():
    global count, status
    while 1:
        with lock:
            print(current_thread().name, "lock acquired")  # line 10
            count += 1
            status = count
            print(current_thread().name, "updated status= %d" % (status))
            sleep(0.5)


def fun2():
    global count, status
    while 1:
        with lock:
```

```
            print(current_thread().name, "lock acquired")  # line 22
            count += 1
            status = count
            print(current_thread().name, "updated status= %d" % (status))
            sleep(0.5)


if __name__ == "__main__":
    f1 = Thread(target=fun1)
    f2 = Thread(target=fun2)
    f1.start()
    f2.start()
```

---

2. [40 pts, concurrency]
   Imagine you need a producer/consumer design with multiple producers and consumers to process some data. Sketch a design (pseudocode) that uses a counting semaphore for critical section management.

Answer

```
prog ProducerConsumer:

BUFFER_SIZE: int = n;
buffer: bytes[BUFFER_SIZE] = [];
sem: Semaphore = 1;

proc Producer:
    while true:
        // critical section
        sem.acquire();
        if buffer.size() == BUFFER_SIZE:
            print("Buffer is full");
        else:
            buffer.push(item);
        sem.release();
        // critical section

proc Consumer:
    while true:
        // critical section
        sem.acquire();
        if buffer.size() == 0:
            print("Buffer is empty");
        else:
            buffer.pop();
        sem.release();
        // critical section
```

```
proc main:
    for i = 1 to x:
        p: Producer = Producer();
        c: Consumer = Consumer();
        pt: Thread = Thread(p);
        ct: Thread = Thread(c);
        pt.start();
        ct.start();
```
