Some Classic Concurrency Problems

Queues

Joseph Kehoe[1]

[1]Department of Computing and Networking
Institute of Technology Carlow

CDD101, 2017

Queues

- Initial Value of semaphore is 0
- Code is written so that it is not possible to signal unless a thread is waiting
    - Value of semaphore is, therefore, never positive!
- Example
    - We want thread to proceed in pairs
    - Leaders and followers
    - Leaders cannot proceed unless a follower is waiting
    - Similarly for followers
        - Like Ballroom dancing

Hint

```
leaderQ=Semaphore(0)
followerQ=semaphore(0)
```

Solution

Leader

```
followerQ.signal()
leaderQ.wait()
dance()
```

Follower

```
leaderQ.signal()
followerQ.wait()
dance()
```

Issues

- It allows leaders and followers to proceed in pairs
- But does it force them to?
  - It is possible for any number of threads to accumulate before executing dance!
- Change the solution so that it solves this problem
  - Leader can invoke dance concurrently with only one follower and vice versa

Hint

```
Leaders=Followers=0
mutex=semaphore(1)
leaderQ=semaphore(0)
followerQ=semaphore(0)
rendezvous=semaphore(0)
```

Solution - Leaders

```
mutex.wait()
if followers>0:
    followers-
    followersQ.signal()
else:
    leaders++
    mutex.signal()
    leaderQ.wait()
dance()
rendezvous.wait()
mutex.signal()
```

Solution - Followers

```
mutex.wait()
if leaders>0:
    leaders-
    leaderQ.signal()
else:
    followers++
    mutex.signal()
    followerQ.wait()
dance()
rendezvous.signal()
```

FIFO Queue

- There is no way of telling which thread will be woken
  - This can lead to unfairness
  - A thread may wait forever!
- To ensure fairness we need to guarantee an ordering on which thread will be woken
- Design a fifo queue that preserves ordering on threads waiting
  - Create a class “fifo” with wait and signal methods that enforce these constraints

Hint

- Each thread has its own semaphore

```
mySem=semaphore(0)
class fifo:
    def __init__(self):
        self.queue=Queue()
        self.mutex=semaphore(1)
```

- Assume Queue class has add and remove methods
    - but is not thread safe!

Solution

```
class fifo:
    def __init__(self):
        self.queue=Queue()
        self.mutex=semaphore(1)
    def wait():
        self.mutex.wait()
        self.queue.add(mySem)
        self.mutex.signal()
        mySem.wait()
    def signal():
        self.mutex.wait()
        sem=self.queue.remove()
        self.mutex.signal()
        sem.signal()
```