Assignment: Concurrent Programming, Part I

LINGHAN ZHAO A20313766

SECTION 2

CS 450

Concurrent Programming, Part I

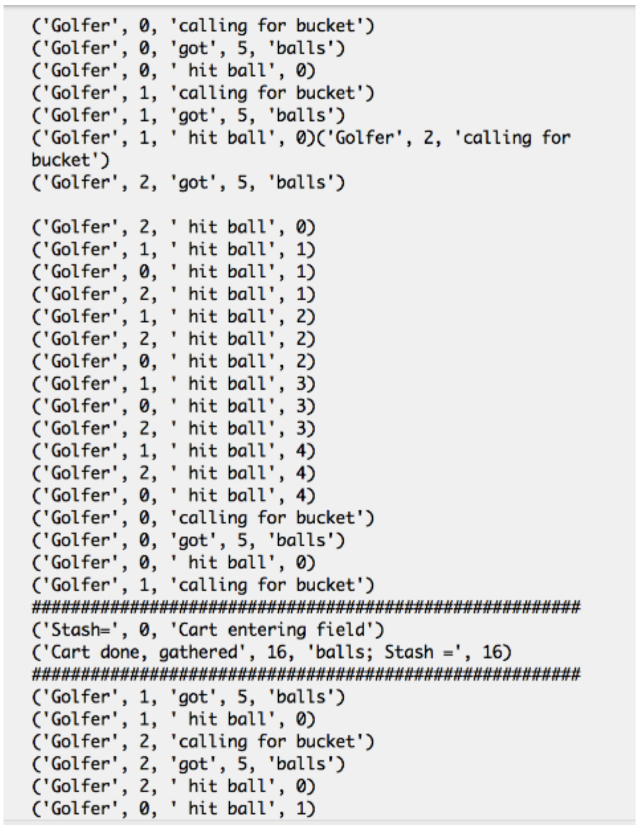
I run all source codes in TextMate Using Python 2.7.2

At the IIT Driving range

According to requirement, it is not hard to find that the Golfing problem is similar to problem Dinning Savage. We need cart to fill in the stash when a golfer can not get balls from stash.

My solution is a combination of the scoreboard pattern with a rendezvous like the Dinning Savage. I use a scoreboard to keep track of the number of stash. If a golfer finds the counter is less than N, which he needs one to fulfill his basket. He wakes the cart and waits for a signal that the stash is full.

A part of outputs:



```
('Golfer', 0, 'calling for bucket')
('Golfer', 0, 'got', 5, 'balls')
('Golfer', 0, ' hit ball', 0)
('Golfer', 1, 'calling for bucket')
('Golfer', 1, 'got', 5, 'balls')
('Golfer', 1, ' hit ball', 0)('Golfer', 2, 'calling for
bucket')
('Golfer', 2, 'got', 5, 'balls')

('Golfer', 2, ' hit ball', 0)
('Golfer', 1, ' hit ball', 1)
('Golfer', 0, ' hit ball', 1)
('Golfer', 2, ' hit ball', 1)
('Golfer', 1, ' hit ball', 2)
('Golfer', 2, ' hit ball', 2)
('Golfer', 0, ' hit ball', 2)
('Golfer', 1, ' hit ball', 3)
('Golfer', 0, ' hit ball', 3)
('Golfer', 2, ' hit ball', 3)
('Golfer', 1, ' hit ball', 4)
('Golfer', 2, ' hit ball', 4)
('Golfer', 0, ' hit ball', 4)
('Golfer', 0, 'calling for bucket')
('Golfer', 0, 'got', 5, 'balls')
('Golfer', 0, ' hit ball', 0)
('Golfer', 1, 'calling for bucket')
########################################################
('Stash=', 0, 'Cart entering field')
('Cart done, gathered', 16, 'balls; Stash =', 16)
########################################################
('Golfer', 1, 'got', 5, 'balls')
('Golfer', 1, ' hit ball', 0)
('Golfer', 2, 'calling for bucket')
('Golfer', 2, 'got', 5, 'balls')
('Golfer', 2, ' hit ball', 0)
('Golfer', 0, ' hit ball', 1)
```

Problems about my code and result output:

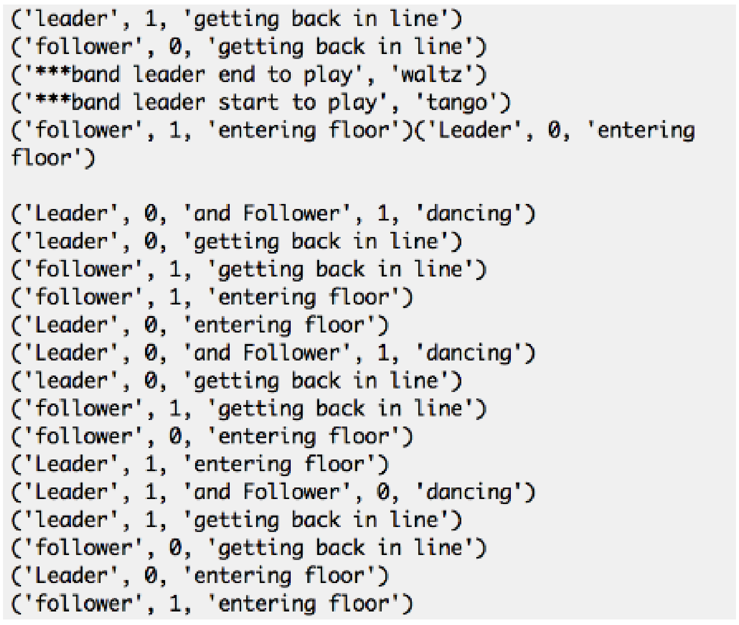
The output satisfies basic demand and do not have quite big problems.

However, it seems all golfers hit a ball as an inconspicuous sequence. The ‘for’ loop and thread could be improved.

Dance mixer

In this problem, I use a queue theory and semaphore to simulate the dance mixer problem. Rendezvous guarantees that leaders and followers are allowed to proceed in pairs. Leaders and followers are counters that keep track of the number of dancers of each kind that are waiting. Rendezvous is used to check that both threads are done dancing. When a leader arrives, it gets the mutex that protects leaders and followers. If there is a follower waiting, the leader decrements followers, signals a follower, and then invokes dance, all before releasing mutex. That guarantees that there can be only one follower thread running dance concurrently. If there are no followers waiting, the leader has to give up the mutex before waiting on leaderQueue.

A part of outputs:



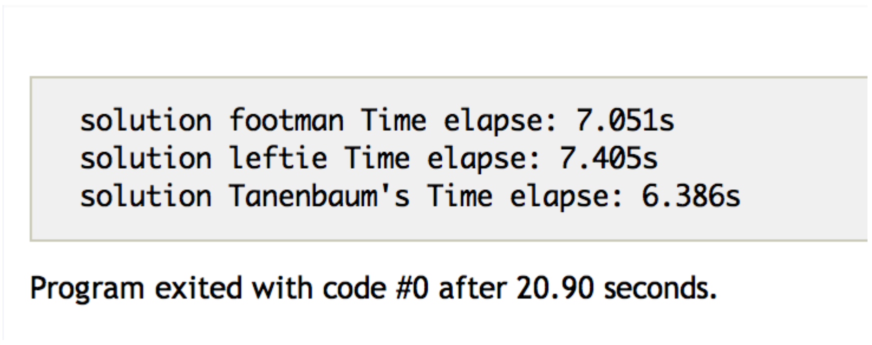
```
('leader', 1, 'getting back in line')
('follower', 0, 'getting back in line')
('***band leader end to play', 'waltz')
('***band leader start to play', 'tango')
('follower', 1, 'entering floor')('Leader', 0, 'entering
floor')

('Leader', 0, 'and Follower', 1, 'dancing')
('leader', 0, 'getting back in line')
('follower', 1, 'getting back in line')
('follower', 1, 'entering floor')
('Leader', 0, 'entering floor')
('Leader', 0, 'and Follower', 1, 'dancing')
('leader', 0, 'getting back in line')
('follower', 1, 'getting back in line')
('follower', 0, 'entering floor')
('Leader', 1, 'entering floor')
('Leader', 1, 'and Follower', 0, 'dancing')
('leader', 1, 'getting back in line')
('follower', 0, 'getting back in line')
('Leader', 0, 'entering floor')
('follower', 1, 'entering floor')
```

Problems about my code and result output:

Some semaphores do not act accurately, which leads some problems about pairing up and music change. In the output, I find the queue is not very useful to ensure dance waiting as expected. I try very hard to modify and improve, but the result is not distinctive.

Dining philosophers

According to this part, I have to calculate the elapse time of each solution about problem dinning philosopher and compare them. I run the code specifying 20 philosophers, with a requisite 10 meals eaten by each, produce the following results Result (each philosopher will sleep for random time after eating):



```
solution footman Time elapse: 7.051s
solution leftie Time elapse: 7.405s
solution Tanenbaum's Time elapse: 6.386s
```

Program exited with code #0 after 20.90 seconds.

In this output, I can find footman and leftie is higher than Tanenbaum. Leftie is the highest of three solutions.

Because in footman and leftie, the philosopher has to wait if others are holding a resource (fork) but do not use it. In the leftie, it will add a condition branch which will cause a little more response time.

However, in Tanenbaum’s solution no thread executes wait while holding mutex. So, it will save time so the response time is lower than footman and leftie solution.