Assignment 5, Synchronization

Kevin Brandstatter

Problem 1:

I solved the golfing problem with the use of locks. I created locks for modifying the stash and the balls on field so as to not have loss due to overlapping transactions. Then I used a semaphore of 0 to cause the cart to initially wait. Then, when a golfer requested a bucket it would release the cart and wait for it to finish using another semaphore. Also, this was held in the stash mutex which prevented other golfers from asking for a bucket until the cart was done. The cart locked the balls on the field while it collects them.

Problem 2:

For the mixer, I used classes for the leaders and followers and two global queues. In the line_up function the dancer adds itself to the appropriate queue. In enter floor, the dancer waits for a signal. The main thread pops the first leader and signals it to dance. The leader then pops the first follower and signals to start dancing. Once they've danced for a random amount of time they line back up and repeat.

Problem 3:

To make the baboon program finite, I added a global variable for max crossings and a local counter for number of times crossed, thus the baboon loop terminated when this limit was reached.

Problem 4:

My hypothesis was that implementing a draining process wouldn't increase performance as it doesn't maximize concurrency. Instead it's focused on eliminating starvation if a continuous stream of baboons arrives on either side preventing the other side from going. In order to implement this draining, I added an extra parameter to the Lightwitch lock where if it was the MAX_ROPEth + 1 baboon it had to wait to acquire the semaphore. Thus the baboons would file off the rope and both sides could contend for the rope.

From comparison to the original version I saw a performance degradation in terms of time. This is likely due to the fact that the solution penalizes full concurrency by draining the rope.

Problem 5:

For my own optimization, I implemented a similar draining process, however instead of being related to the rope being full, it was instead relative to the number of baboons that have crossed in the stream from one side. After N baboons have crossed from one side, the rope would be drained. By doing this I allow each side to gather an amount of baboons so that when the rope is drained, the other side could potentially fill the rope. Also, this count is reset if the rope empties from lack of baboons on the crossing side.

My approach compared with the other two yielded results that showed my solution similarly matched the original and was better than the draining solution. Though it should be noticed that while my solution didn't directly outperform the original solution, it does address the problem of starvation which is an issue in an open system.