Computer Operating Systems, Practice Session 8

Example Synchronization Problems 2

Mustafa Ersen (ersenm@itu.edu.tr)

Istanbul Technical University
34469 Maslak, İstanbul

02 April 2014

Today

Computer Operating Systems, PS 8

Roller Coaster Problem

Multi-car Roller Coaster Problem

Santa Claus Problem

Roller Coaster Problem

Suppose there are `n` passenger threads and a car thread. The passengers repeatedly wait to take rides in the car, which can hold `C` passengers, where `C < n`. The car can go around the tracks only when it is full.

- Passengers should invoke `board` and `unboard`.
- The car should invoke `load`, `run` and `unload`.
- Passengers cannot `board` until the car has invoked `load`
- The car cannot depart until `C` passengers have boarded.
- Passengers cannot `unboard` until the car has invoked `unload`.

Write code for the passengers and car that enforces these constraints.

Required Variables and Initial Values

```
1  // Counters
2  boarders = 0
3  unboarders = 0
4  // Mutex semaphores for counters
5  mutex = Semaphore(1)
6  mutex2 = Semaphore(1)
7  // Passenger queues
8  boardQueue = Semaphore(0)
9  unboardQueue = Semaphore(0)
10 // The status of the car
11 allAboard = Semaphore(0)    // full
12 allAshore = Semaphore(0)    // empty
```

Car Thread

Before each ride:

- The car signals `C` passengers to board, then waits for the last one to signal `allAboard`

After each ride:

- The car signals all `C` passengers to unboard, then waits the last one to signal `allAshore`

```
1 load()
2 boardQueue.signal(C)
3 allAboard.wait()
4
5 run()
6
7 unload()
8 unboardQueue.signal(C)
9 allAshore.wait()
```

Passenger Thread

Passengers wait for the car before boarding, naturally, and wait for the car to stop before leaving. The last passenger to board/unboard signals the car and resets the relevant passenger counter.

```
1  boardQueue.wait()
2  board()
3  mutex.wait()
4     boarders += 1
5     if boarders == C:
6        allAboard.signal()
7        boarders = 0
8  mutex.signal()
9
10 unboardQueue.wait()
11 unboard()
12 mutex2.wait()
13    unboarders += 1
14    if unboarders == C:
15       allAshore.signal()
16       unboarders = 0
17 mutex2.signal()
```

Multi-car Roller Coaster Problem

Some additional constraints need to be satisfied for the case of multiple cars:

- Only one car can be boarding at a time.
- Multiple cars can be on the track concurrently.
- Since cars can't pass each other, they have to unload in the same order they boarded.
- All the threads from one carload must disembark before any of the threads from subsequent carloads.

Modify the previous solution to handle these additional constraints. You can assume that there are `m` cars, and that each car has a local variable named `i` that contains an identifier between 0 and `m - 1`.

Required Variables and Initial Values

- Two lists of semaphores are defined to keep the cars in order:
  - `loadingArea` represents the loading area where passengers board a car
  - `unloadingArea` represents the unloading area where passengers unboard a car
- Each list contains one semaphore for each car.
- To enforce the ordering constraints among the cars, only one semaphore in each list is unlocked at any time.
  - Initially, only the semaphores for Car 0 are unlocked.
  - Before loading/unloading, each car waits on its own semaphore, and it signals the next car as it leaves.

```
1 loadingArea = [Semaphore(0) for i in range(m)]
2 loadingArea[1].signal()
3 unloadingArea = [Semaphore(0) for i in range(m)]
4 unloadingArea[1].signal()
```

The function `next` computes the identifier of the next car in the sequence (wrapping around from `m - 1` to 0):

```
1 def next(i):
2   return (i + 1) % m
```

Modified Car Thread

No need to modify the passenger thread as using 2 mutex semaphores allows a passenger to board a car while another passenger is unboarding the next car.

```
1  loadingArea[i].wait()
2  load()
3  boardQueue.signal(C)
4  allAboard.wait()
5  loadingArea[next(i)].signal()
6
7  run()
8
9  unloadingArea[i].wait()
10 unload()
11 unboardQueue.signal(C)
12 allAshore.wait()
13 unloadingArea[next(i)].signal()
```

Santa Claus Problem

Santa Claus sleeps in his shop at the North Pole and can only be awakened by either

1. all nine reindeer being back from their vacation in the South Pacific,
2. or some of the elves having difficulty making toys.

To allow Santa to get some sleep, the elves can only wake him when three of them have problems. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return.

If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready. (It is assumed that the reindeer do not want to leave the tropics, and therefore they stay there until the last possible moment.)

Santa Claus Problem

- After the ninth reindeer arrives, Santa must invoke `prepareSleigh`, and then all nine reindeer must invoke `getHitched`.
- After the third elf arrives, Santa must invoke `helpElves`. Concurrently, all three elves should invoke `getHelp`.
- All three elves must invoke `getHelp` before any additional elves enter.

Santa should run in a loop so he can help many sets of elves. We can assume that there are exactly 9 reindeer, but there may be any number of elves. Write code for Santa, reindeer and elf threads that satisfies the given constraints.

Required Variables and Initial Values

```
1  // Counters
2  elves = 0
3  reindeer = 0
4  // Mutex semaphore for counters
5  mutex = Semaphore(1)
6  // To wake Santa up
7  santaSem = Semaphore(0)
8  // For reindeer to get hitched
9  reindeerSem = Semaphore(0)
10 // To prevent additional elves from entering
11 // while three elves are being helped
12 elfTex = Semaphore(1)
```

Santa Thread

- When Santa is woken up by a signal (`santaSem`), he checks which of the two conditions holds and either deals with the reindeer or the waiting elves.
- If there are nine reindeer waiting, Santa invokes `prepareSleigh`, then signals `reindeerSem` nine times, allowing the reindeer to invoke `getHitched`.
- If there are elves waiting, Santa just invokes `helpElves`.

```
1 santaSem.wait()
2 mutex.wait()
3   if reindeer == 9:
4     prepareSleigh()
5     reindeerSem.signal(9)
6   else if elves == 3:
7     helpElves()
8 mutex.signal()
```

Reindeer Thread

- The ninth reindeer being back from vacation signals Santa and then joins the other reindeer waiting on `reindeerSem`.
- When Santa signals, the reindeer all execute `getHitched`.

```
1 mutex.wait()
2   reindeer += 1
3   if reindeer == 9:
4     santaSem.signal()
5 mutex.signal()
6
7 reindeerSem.wait()
8 getHitched()
```

Elf Thread

- The last elf to enter holds `elfTex`, stopping other elves from entering until all three elves have invoked `getHelp`.
- The last elf to leave releases `elfTex`, allowing the next batch of elves to enter.

```
1  elfTex.wait()
2  mutex.wait()
3     elves += 1
4     if elves == 3:
5       santaSem.signal()
6     else
7       elfTex.signal()
8  mutex.signal()
9
10 getHelp()
11
12 mutex.wait()
13    elves -= 1
14    if elves == 0:
15      elfTex.signal()
16 mutex.signal()
```

References

- Downey, A. B. (2008). The little book of semaphores. Version 2.1.5. Green Tea Press.