Semaphore Homework
CECS 326 PM

Francisco Fierro
f.a.fierrojr@gmail.com

December 11, 2017

Problem 1

Algorithm 1 Problem 1

```
1: procedure Process 1
2:     loop:
3:         f1()
4:         V(GOF21) #notify P2
5:         V(GOF31) #notify P3
6:         P(GOF12) #wait for P2
7:         P(GOF13) #wait for P3
8:         g1()
9:         h1()
10:        V(GOH21) #notify P2
11:        V(GOH31) #notify P3
12:        P(GOH12) #wait for P2
13:        P(GOH13) #wait for P3
14:        m1()
15:        n1()
16:        V(GON31) #notify P3
17:        P(GON13) #wait for P3
18:    end loop
19: end procedure
```

Algorithm 2 Problem 1

```
1: procedure Process 2
2:     loop:
3:         f2()
4:         V(GOF12) #notify P1
5:         V(GOF32) #notify P3
6:         P(GOF21) #wait for P2
7:         P(GOF23) #wait for P3
8:         g2()
9:         V(GOG32) #notify P3
10:        P(GOG23) #wait for P3
11:        h2()
12:        V(GOH12) #notify P1
13:        V(GOH32) #notify P3
14:        P(GOH21) #wait for P2
15:        P(GOH23) #wait for P3
16:        m2()
17:        n2()
18:    end loop
19: end procedure
```

Algorithm 3 Problem 1

```
procedure Process 3
    loop:
        f3()
        V(GOF13) #notify P1
        V(GOF23) #notify P2
        P(GOF31) #wait for P1
        P(GOF32) #wait for P2
        g3()
        V(GOG23)
        P(GOG32) #wait for P2
        h3()
        V(GOH13) #notify P1
        V(GOH23) #notify P2
        P(GOH31) #wait for P1
        P(GOH32) #wait for P2
        m3()
        n3()
        V(GON13) #notify P1
        P(GON31) #wait for P1
    end loop
end procedure
```

The Basic Idea

Each semaphore is initialized to zero. In this solution, a semaphore represents a "Go" signal to a waiting process from a process that has just completed a required section. The first number represents the process receiving the "go" signal and the second number represents the process sending the "go" signal As a process completes a section and reaches the next halting point, it first notifies each of the others of completion with V operations and then it waits on the others for their completion signals. There is exactly one P operation and one V operation per other process with which progress is synchronized at that point.

Correctness

I'm using a separate set of semaphores for each sync point to hopefully make it easier to prove correctness. Each process will essentially signal the others that they have reached a sync point and then wait for them to do the same. They do this by perform two V operations that the other two processes perform P on and then performing P on each semaphore the other two processes will perform V on. This design will not work in every scenario. For instance, the example used in our study guide and exam could have a process loop around twice before the others got to the sync point if the other two processes did both their Vs and then the third process kept going as far as it could. But this design works in this scenario. The key here is that there are two points at which all three processes sync; with the very last point being one where all three sync. When the three processes sync at this point all other semaphores – not involved with this point's syncing – are zero. That's because the three processes have done all the Ps and Vs related to them. Similarly, when the three processes loop and all sync back at the top, the semaphores involved with the last sync point are all zero. This "resetting" makes it easy to observe the correctness of the program.

Starvation

The processes are interlocked. Each process will always eventually reach a point where they must wait on each of the other processes to proceed. So there is no possibility of some subset of processes going on indefinitely while another subset starves.

Deadlock

Though every semaphore starts at zero, because the processes give "go" signals with the V operation before waiting with a P operation, no amount of sneezing from Thor should cause a deadlock scenario.

Problem 2

Algorithm 4 Problem 2

```
1: procedure Robot A
2:     loop:
3:         compute briefly
4:         P(A0A1) #lock out A0 A1
5:         P(B0A0) #lock out B0 A0
6:         occupy A0
7:         return to base
8:         V(A0A1) #free A0 A1
9:         V(B0A0) #free B0 A0
10:        compute a little while
11:        P(B0A0) #lock out B0 A0
12:        occupy B0
13:        V(B0A0) #free B0 A0
14:        return to base
15:    end loop
16: end procedure
```

Algorithm 5 Problem 2

```
1: procedure Robot B
2:     loop:
3:         calculate for a moment
4:         P(B0A0) #lock out B0 A0
5:         occupy B0
6:         return to base
7:         V(B0A0) #free B0 A0
8:         calculate a little more
9:         P(B2B1) #lock out B2 B1
10:        occupy B1
11:        return to base
12:        V(B2B1) #free B2 B1
13:    end loop
14: end procedure
```

Algorithm 6 Problem 2

```
1: procedure Robot C
2:     loop:
3:         await a shipment
4:         P(A0A1) #lock out A0 A1
5:         occupy A1
6:         return to base
7:         V(A0A1) #free A0 A1
8:         await a 2nd shipment
9:         P(B2B1) #lock out B2 B1
10:        occupy B1
11:        return to base
12:        V(B2B1) #free B2 B1
13:    end loop
14: end procedure
```

Algorithm 7 Problem 2

```
1: procedure Robot D
2:     loop:
3:         P(A0A1) #lock out A0 A1
4:         P(B0A0) #lock out B0 A0
5:         occupy A0
6:         return to base
7:         V(B0A0) #free B0 A0
8:         V(A0A1) #free A0 A1
9:         intermission control
10:        P(B2B1) #lock out B2 B1
11:        P(B2A2) #lock out B2 A2
12:        occupy B2
13:        return to base
14:        V(B2A2) #free B2 A2
15:        V(B2B1) #free B2 B1
16:        download new instructions
17:    end loop
18: end procedure
```

Algorithm 8 Problem 2

```
1: procedure ROBOT E
2:     loop:
3:         check inventory
4:         P(A0A1) #lock out A0 A1
5:         occupy A1
6:         return to base
7:         V(A0A1) #free A0 A1
8:         do a quick calculation
9:         P(B2A2) #free B2 A2
10:        occupy A2
11:        return to base
12:        V(B2A2) #free B2 A2
13:    end loop
14: end procedure
```

The Basic Idea

Each semaphore is initialized to 1. In this solution, a semaphore represents the availability of a resource. In a sense, I've partitioned the work areas into four resource blocks: (A0, A1), (B0, A0), (B2, A2), (B2, B1). These resources can be grouped into two pairs of intersecting blocks: (A0, A1) and (B0, A0) intersect at A0, and (B2, A2) and (B2, B1) intersect at B2. When a robot attempts to occupy either A0 or B2 it will operate P and later V on both of their respective intersecting blocks.

Correctness

Occupying A0: If a robot attempts to occupy A0, it will only occupy it if (A0 A1), and (B0, A0) are unoccupied. Occupying A0 locks out A0, A1, and B0. Ocuppying A1: If a robot attempts to occupy A1, it will only occupy it if (A0, A1) is unoccupied. Occupying A0 locks out A0 and A1. A robot may concurrently occupy B0. Ocuppying A2: If a robot attempts to occupy A2, it will only occupy it if (B2, A2) is unoccupied. Occupying A2 locks out B2 and A2. A robot may concurrently occupy A1. Ocuppying B0: If a robot attempts to occupy B0, it will only occupy it if (B0, A0) is unoccupied. Occupying B0 locks out B0 and A0. A robot may concurrently occupy A1. Ocuppying B1: If a robot attempts to occupy B1, it will only occupy it if (B2, B1) is unoccupied. Occupying B1 locks out B1, and B2. A robot may concurrently occupy A2.Ocuppying B2: If a robot attempts to occupy B2, it will only occupy it if (B2, A2) and (B2, B1) are unoccupied. Occupying A2 locks out B2, B1, and A2. Therefore, the requirements are met.

Starvation

So long as each process eventually gets a turn, every process should continue to progress. There is not a condition where a process may become stuck indefinitely.

Deadlock

The most precarious scenario would be where robots attempt to occupy A0 and B2, but in both cases are interrupted between P operations either because the second resource is taken or Thor sneezed. But whether that second resource is available or in use, the second P operations will eventually return and allow the robots to occupy A0/B2. The key here is that the sets of resource blocks that intersect at either A0 or B2 are disjoint.