CS350 Assignment 7

Chris-Emio (chrisr98@bu.edu)

April 16, 2020

Written Part

Problem 1
a)

```
semaphore P1 = 1
semaphore P2 = 1
semaphore P3 = 1
//Repeat for P1
Process P1:
    signal(P1) //Fetching data and write in on B1
    wait(P2) //Wait for P2 to finish reading and clear B1
forever

//Repeat for P2
Process P2:
    wait(P1)
    //Once P2 reads this data block from B1, performs some
    //encoding, and writes the encoded data block into a
    //network buffer B2. Once the content of B1 is
    //successfully written to B2, P2 clears the content of B1
    signal(P2)
    wait(P3) //Wait to clear B2
forever

//Repeat for P3
Process P3:
    wait(P2)
    //responsible for reading the content of B2 and
    //transmitting the read data block (if B2 contains one)
    //over the network
    signal(P3)
forever
```

b) I used a semaphore for each process. If required they wait and signal the previous process that the content has be cleared and can begin another process.

Without those semaphores some processes would overwrite buffers before the next process finish.

c)

```
int c1 = X
int c2 = Y
semaphore P1 = c1
semaphore P2 = c2
semaphore P3 = 1
//Repeat for P1
Process P1:
    for (int i < c1){
        signal(P1) //Fetching data and write in on B1
    }
    wait(P2) //Wait for P2 to finish reading and clear B1
forever

//Repeat for P2
Process P2:
    wait(P1)
    //Once P2 reads this data block from B1, performs some
    //encoding, and writes the encoded data block into a
    //network buffer B2. Once the content of B1 is
    //successfully written to B2, P2 clears the content of B1
    for (int i < c2){
        signal(P2)
    }
    wait(P3) //Wait to clear B2
forever

//Repeat for P3
Process P3:
    wait(P2)
    //responsible for reading the content of B2 and
    //transmitting the read data block (if B2 contains one)
    //over the network
    signal(P3)
forever
```

d) I did not have to introduce a new semaphore because I simply had to signal up to the amount of space in the buffer. Assuming if buffer is full next request goes in a queue and not dropped.

Problem 2

a) For extra credit: Those engineers should be sued and they should build another bridge.

Assuming they can't build another bridge and those engineers are nowhere to be found. Also assuming that citizens arrive in set amounts, it isn't a continuous flow/ an endless line crossing the bridge.

```
semaphore left = 1
semaphore right = 1
semaphore go = 1

//Repeat for left (Going from Pest to Buda)
Process TravelToBuda:
    wait(go)
    // People cross the bridge until done
    // or if there is no one that side
    signal(left)
    wait(right)
forever

//Repeat for Right (Going from Buda to Pest)
Process TravelToPest:
    wait(left)
    // People cross the bridge until done
    // or if there is no one that side
    signal(right)
forever

Process GoFirst:
    //If a group of travelers arrive at the bridge first
    //signal for that side accordingly

    //If both side arrive at the same time
    Signal(go) // Pest to buda always go first
forever
```

b) I think the problem is asking to solve the problem of continuous flow. Assuming citizens come in a continuous flow and block the path for the other city forever. This puts a limit on how many can cross before letting the other side go through.

```
semaphore left = 1
semaphore right = 1
semaphore go = 1
int K = 5
//Repeat for left (Going from Pest to Buda)
Process TravelToBuda:
    wait(go)
    // if K people cross or there is no remaining citizens
    // then lets the other side send K people
```

```
        signal(left)
        wait(right)
    forever

    //Repeat for Right (Going from Buda to Pest)
    Process TravelToPest:
        wait(left)
        // if K people cross or there is no remaining citizens
        // then lets the other side send K people
        signal(right)
    forever

    Process GoFirst:
        //If a group of travelers arrive at the bridge first
        //signal for that side accordingly

        //If both side arrive at the same time
        Signal(go) // Pest to buda always go first
    forever
```

Problem 3

Assuming philosophers can roll a die and get the same number meaning they do the same activity.

a) With 2 philosophers in the apartment. There wouldn't be any deadlock. There are enough resources for 2 philosophers to do any activity in the apartment at the same time.

b) There is a possibility for a deadlock. There are are only 2 plates and if all 3 philosophers roll to eat for the day there will not be enough plates

c) The will be a deadlock. Similar to the scenario for 3 philosophers. If 3 of the 4 end up rolling to eat for the day there will not be enough plates for all 3 to the same activity for the day. Additionally, if there 2 want to write and the other 2 want to think. There is not enough resources for all of them to complete their activity. Any combination of think, write, eat will for the 4 will result in a deadlock.

d)

```
    // Shared global variables
    semaphore pencil := 3;
    semaphore notebook :=3;
    semaphore laptop := 2;
    semaphore plate := 2;
    semaphore bed := 4;
Process Modern Philosopher:
    // Local variables
    enum activity chosen;
    Repeat:
```

```
    // Day begins!
    activity chosen = roll dice();
    switch (activity chosen) {
    case THINK:
        wait(pencil);
        wait(notebook);
        break;
    case WRITE:
        wait(pencil);
        wait(notebook);
        wait(laptop);
        break;
    case EAT:
        wait(laptop);
        wait(plate);
        break;
    case SLEEP:
        wait(bed);
        break;
    default:
        print("This is NOT supposed to happen !!\n");
        exit ();
    }


perform activity();
    if (activity chosen == THINK) {
        // Sharpen pencil
        signal(pencil);
        // Close notebook
        signal(notebook);
    }
    if (activity chosen == WRITE) {
        // Sharpen pencil
        signal(pencil);
        // Close notebook
        signal(notebook);
        // Shutdown laptop
        signal(laptop);
    }
    if (activity chosen == EAT) {
        // Shutdown laptop
        signal(laptop);
        // Wash plate
        signal(plate);
}
if (activity chosen == SLEEP) {
    //Prepare bed
    signal(bed);
}
Forever
```