1. What is the difference between binary and counting semaphores?
A binary semaphore is a counting semaphore that has a count of either 1 or 0. This ensures that only one thread/process will be running at a time. A counting semaphore can have a count of any positive integer and can set a max number of threads running at once

2. What are the operations that can be performed on a semaphore. Explain in brief about each operation and what it does in your own words. A semaphore can either be incremented (semSignal/semPost) or decremented (semWait). Incrementing will increase a semaphore's counter and unblock a process if the semaphore's counter is 0, while decrementing will keep block a process/thread until a different process/ thread increments it.

3. **5.6.** Consider the following processes P1 and P2 that update the value of the shared variables, x and y, as follows:

```
Process P1 :                    Process P2 :
(  performs the operations:     (  performs the operations:
      x := x * y                      x ++
      y ++                            y := x * y
)                               )
LOAD R1, X                      LOAD R3, X
LOAD R2, Y                      INC R3
MUL R1, R2                      LOAD R4, Y
STORE X, R1                     MUL R4, R3
INC R2                          STORE X, R3
STORE Y, R2                     STORE Y, R4
```

Assume that the initial values of x and y are 2 and 3 respectively. P1 enters the system first and so it is required that the output is equivalent to a serial execution of P1 followed by P2. The scheduler in the uniprocessor system implements a pseudo-parallel execution of these two concurrent processes by interleaving their instructions without restricting the order of the interleaving.

a. If the processes P1 and P2 had executed serially, what would the values of x and y have been after the execution of both processes?

b. Write an interleaved concurrent schedule that gives the same output as a serial schedule.

c. Write an interleaved concurrent schedule that gives an output that is different from that of a serial schedule.

A. After Process 1 the value of X is 6 and Y is 4. After Process 2, X is 7 and Y is 28.

B. sem_t *semaphore; something to initialize a semaphore
sem_init(sem, 1, 0); start semaphore at 0
Do process 1;
sem_post(sem); signal it's done

sem_wait(sem); make semaphore wait for other process to finish
Do process 2;
sem_destroy;
return 0; finished program

C. sem_t *semaphore; // something to initialize a semaphore
sem_init(sem, 1, 0); start semaphore at 0
Do x:= x*y from process 1;
sem_wait(sem); wait for process 2 to finish
increment X to finish process 1;
Do process 2;
sem_post(sem); to allow process 1 to finish
sem_destroy;
return 0; finished program

4. The following three functions are run on a shared processor by three processes. They can coordinate their execution via shared semaphores that respond to the standard signal(`sem_signal()`) and wait(`sem_wait()`) procedures. In order to produce the output HELLO, add respective `sem_signal()/sem_post()` and `sem_wait()` comands in the code. Create your own semaphores as needed.

- Is printing HELLO possible
- Number of semaphores - 2
- Names of semaphores - sem1, sem2
- Initial values of sempahores - sem_init(&Sem, 0, 0)

Function#1	Function#2	Function #3
	sem_wait(&sem1)	sem_wait(&sem2)
`print("H")`	`print("L")`	`print("O")`
`print("E")`	`print("L")`	
sem_post(&sem1)	sem_post(&sem2)	

終わる