5.4 cont + 5.5 MONITORS

1- a)

P1 decrements x (9)

P1 increments x (10)

P2 decrements x (9)

P1 checks x (9 != 10 true)

P2 increments x (10)

P1 prints x is 10

b)

P1 decrements x (9)

P2 decrements x (8)

P1 increments x (9)

P2 reaches the decrement statement again before p1 checks x (8)

P1 checks x (8 != 10)

P1 prints x is 8.

2-

```
wait(lock);

printf(“x is %d”, x);

signal(lock);
```

3- semaphores are lower-level primitives that provide synchronization between processes. On the other hand, monitors provide synchronization in a high-level manner.

5.6 MESSAGE PASSING

4- message passing is a technique in which processes are able to communicate with each other through information exchange. To clarify how it can be used to enforce mutual exclusion, suppose a process needs access to a shred resource.

The process will send a request to another process that is responsible for coordinating access to shared resources. When the process is granted access to the shared resource, no other process can access it.

5.7 READERS/WRITERS PROBLEM

5- the problem is that the semaphore mutex is not protecting the critical section of the readers functions.

General questions

7-

a) The algorithm uses two arrays: choosing and number, to track the progress of each process as it enters the critical section. Each process has a unique identifier i and maintains a flag choosing[i] to indicate that it needs to access the critical section. Each process also has a number[i] value, which is used to determine the order of access to the critical section. This algorithm guarantees that only one process can enter its critical section at a given time and ensures that all processes eventually enter the critical section in a fair manner.

b) The algorithm uses a "wait-for" loop to ensure that all other processes have finished choosing their numbers before entering the critical section. This guarantees that no two processes can simultaneously enter the critical section, which eliminates the possibility of a deadlock.

c) the algorithm enforces mutual exclusion through 2 arrays, choosing and number. Mutual exclusion is guaranteed according to the following steps:

1- when a process needs access to the critical section, it sets a flag in choosing[i]

2- the process calculates a unique number for itself through getmax()

3- the process broadcasts that it finished choosing a number by setting a flag choosing[i]

4- the process waits for other processes to finish choosing numbers

5- the process acquires the smallest number among all processes indicating its priority.

6- the algorithm orders accesses to critical sections based on a unique number which guarantees mutual exclusion.