Operating systems

Sheet 6

(EED)

Name: Samuel Ayman Shawky

ID: 20010750

5.4 cont + 5.5 MONITORS

1-

a-

p1 or p2 execute completely without being interrupted
x will be 10

OR

P2	**x=x-1**	**9**
P1	**X=x-1**	**8**
P2	**X=x+1**	**9**
P2	if (x! =10)	**9**
P1	**X=x+1**	**10**
P2	printf("x is %d",x)	**10**

b-

process	Instructions	x	P1-R0	P2-R0
P1	LD R0, X	**10**	**10**	**-**
P1	Dec R0	**10**	**9**	**-**
P1	STO R0, X	**9**	**9**	**-**
P2	LD R0, X	**9**	**9**	**9**
P2	DEC R0	**9**	**9**	**8**
P2	STO R0, X	**8**	**9**	**8**
P1	LD R0, X	**8**	**8**	**8**
P1	INCR R0	**8**	**9**	**8**
P2	LD R0, X	**8**	**9**	**8**
P2	INCR R0	**8**	**9**	**9**
P2	STO R0, X	**9**	**9**	**9**
P1	STO R0, X	**9**	**9**	**9**

P2	if(x != 10) printf("x is %d", x)	**9**	**9**	**9**
	X is 9	**9**	**9**	**9**
P2	if(x != 10) printf("x is %d", x)	**9**	**9**	**9**
	X is 9	**9**	**9**	**9**
P1	LD R0, X	**9**	**9**	**9**
P1	Dec R0	**9**	**8**	**9**
P1	STO R0, X	**8**	**8**	**9**
P2	LD R0, X	**8**	**8**	**8**
P2	DEC R0	**8**	**8**	**7**
P2	STO R0, X	**7**	**8**	**7**
P1	LD R0, X	**7**	**7**	**7**
P1	INCR R0	**7**	**8**	**7**
P1	STO R0, X	**8**	**8**	**7**
P2	if(x != 10) printf("x is %d", x)	**8**	**8**	**7**
	X is 8	**8**	**8**	**7**

2- Show where/how to add semaphores to the program in the preceding problem to insure that the printf() is never executed. Your solution should allow as much concurrency as possible.

```
Semaphore s=1;
    P1: {
            shared int x;
            x = 10;
            while (1) {
                    semwait(s)
                    x = x - 1;
                    x = x + 1;
                    if (x != 10){
                            printf("x is %d",x);
                            semsignal(s);
                    }
            }
    }
    P2:{
            shared int x;
            x = 10;
            while ( 1 ) {
                    semwait(s)
                    x = x - 1;
                    x = x + 1;
                    if (x!=10){
                            printf("x is %d",x);
                            semsignal(s);
                    }
            }
    }
    Int main()
    {
            Parbegin(p1,p2);
    }
```

3-

ANSWER)

Semaphores are low-level synchronization primitives but Monitors are higher-level synchronization constructs. monitor wait and signal operations are different from those for the semaphore. If a process in a monitor signals and no task is waiting on the condition variable, the signal is lost.

4- What is massage passing? How can it be used to enforce mutual exclusion?

Message passing is a communication mechanism used in concurrent programming, where processes or threads exchange data by sending and receiving messages. In the context of enforcing mutual exclusion.

1- When a process/thread wants to access a shared resource, it sends a request message.
2-The coordinator receives the request message and determines whether the requesting process can access the resource without violating mutual exclusion.
3-If the coordinator determines that granting access would violate mutual exclusion, it blocks the request
4-When a process finishes using the shared resource, it sends a release message

5. Explain what is the problem with this implementation of the one-writer many readers problem?

The problem of this implementation that readers have higher priority than writers so if there are many readers there will be starvation for writers as wrt is held until number of readers =0.

6. Write the pseudo code for solving one writer and many readers problem using monitor?

```
Monitor Read_Write_Monitor {
    int readerCount = 0;
   int writerActive = 0;

    startRead() {
        while (writerActive) { //check if there is write process
            wait();  //wait until write process end
        }
        readerCount++;
    }

    endRead() {
        readerCount--; //one reader finished
        if (readerCount == 0) {  // check if there are no readers
            signal();
        }
    }

    startWrite() {
        while (writerActive || readerCount > 0) { //wait if writer is write or reader is read
            wait();
        }
        writerActive = 1;
    }

    endWrite() {
        writerActive = 0;  // set  writer as inactive
        signal();              //sending signal to wait writers or readers
    }
}
```

7-

a. Describe the algorithm in words:

if a process wants to enter critical section, it takes a number and this number is calculated by adding one to the max number taken by a waiting process, after process takes number, it waits and the smallest number held by process its process enter the critical first If two processes pick the same number, then their IDs are used as tiebreakers. The algorithm ensures that once a process is assigned a number, it will eventually enter the critical section.

b. How this Algorithm Avoids Deadlock:

This algorithm avoids deadlock because each process is assigned a unique ticket number based on its order of arrival, ensuring progress and avoiding situations where processes are indefinitely waiting for each other.

c. Proof that the Algorithm Enforces Mutual Exclusion:

Mutual exclusion is enforced as at any given time, only one thread can have the smallest ticket number, allowing only one thread to enter its critical section at any given time. This ensures that no two processes are executing their critical sections at the same time, thus enforcing mutual exclusion.