Operating systems

Sheet 5

(EED)

Name: Samuel Ayman Shawky

ID: 20010750

1. Define the followings:

a. Mutual exclusion

making sure that if one process is accessing critical section of shared resources, the other processes excluded from entering this critical section.

b. Critical section

It’s a part of code in program where shared resources are accessed and only one process can enter this section while others are excluded.

2. True or false:

a. Peterson’s algorithm cannot be generalized to the case of processes (false **)** can be generalized

3. Define the followings:

1. Race condition

Race condition happens when two or more processes or threads reading and writing on shared date item and the result is depending on which is faster.

4-What are the requirements for mutual exclusion?

1. Mutual exclusion must be enforced: Only one process at a time is allowed into its critical section

2. A process that halts in waiting to enter critical sections must do so without interfering with other processes.

3. It must not be possible for a process requiring access to a critical section to be delayed indefinitely: no deadlock or starvation.

4. When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay.

5. No assumptions are made about relative process speeds or number of processors.

6. A process remains inside its critical section for a finite time only.

5. Machine-instruction approach that enforces mutual exclusion has a number of disadvantages, mention them.

1. Busy waiting is employed: while process waiting it continuous consume processor time.
2. Starvation is possible: when there is a process In critical section and many processes are waiting to enter critical section some processes may be denied indefinitely
3. Deadlock is possible.

6-

ANSWER)

The two definitions of semaphores provided are similar but have a subtle difference. In the first definition, the semaphore's count cannot be negative. In the second definition, there isn't any condition preventing the count from being negative, due to this difference the second tells how many processes waiting and the first not.

7-

ANSWER)

There may be errors as there is no mutual exclusion so some results may not be recorded since these increments are not atomic operations, there can be race conditions where the two processes interfere with each other's updates.

The lower bound on the final value of tally would be 50.

The upper bound on the final value of tally would be 100. This would occur if the two processes' increments are perfectly interleaved.

b. If an arbitrary number of these processes are permitted to execute in parallel, the range of final values of tally would increase. The lower bound would still be 50. The upper bound would be 50N, assuming perfect interleaving of the processes' increments.

8-

```
boolean blocked [2];
int turn;
void P (int id) {
   while (true) {
         blocked[id] = true;
         while (turn != id) {
               while (blocked[1-id])
               /* do nothing */;
               turn = id;
         }
         /* critical section */
         blocked[id] = false;
         /* remainder */
   }
}
void main() {
   blocked[0] = false;
   blocked[1] = false;
   turn = 0;
   parbegin (P(0), P(1));
}
```

ANSWER)

With first value of turn =0

1-for process 0 blocked [0]=true.

2-while (turn! = id) false so enter critical section.

3-for process 1 blocked [1] =true.

4- while (turn! = id) false so enter the nested loop.

5- this loop while (blocked[1-id]) is false

6- then turn = 1 so while (turn! = id) false so enter critical section.

so both process 0 and process 1 both in critical section so this code is incorrect for turn =0