5.1 MUTUAL EXCLUSION: SOFTWARE APPROACHES

1-

- a- A requirement in which no more than 1 process is in a critical section that accesses shared resources.
- b- A section of code that accesses a shared resource. It mustn't be executed while another process is in its critical section.

2-

- a- False

5.2 PRINCIPLES OF CONCURRENCY

3-

- a- A situation in which the result of a shared item executed by multiple processes depends on the speed of execution of each process. The shared item will be modified according to the last process that modified it.

4-

- No deadlock or starvation.
- A process remains inside its critical section for a finite time only.
- Mutual exclusion must be enforced.
- A process in its noncritical section must not interfere with other processes.
- When no process is in a critical section, a process that is requiring access to its critical section must be granted access with no delay.

5.3 MUTUAL EXCLUSION: HARDWARE SUPPORT

5-

- machine instruction approach employs busy waiting. When a process is waiting to enter its critical section, it consumes the processor time just to check whether it is granted access or not.
- Starvation is possible. If the selection of the next process to enter the critical section is arbitrary, some process could wait indefinitely.
- Deadlock is possible.

5.4 SEMAPHORES

6- in the first definition, when a process calls semWait() it first checks for the value of s.count. if it is greater than 0, s.count is decremented by one. If s.count is less than or equal to 0, the process gets placed in the blocked queue. When a process calls semSignal(), it first checks whether there are any processes in the blocked queue. If there is, a process will be unblocked. Otherwise, s.count is incremented by 1.

In the second definition, the order of operations is different. When a process calls semWait, it first decrements the value of s.count and checks whether it is negative. If it is, this process will be held in the blocked queue. When a process calls semSignal, the first thing it does is that it increments the value of s.count by one. After that, it checks whether it is less than or equal to 0. If it is, a process is transferred to the ready list. If it is not, no action is taken.

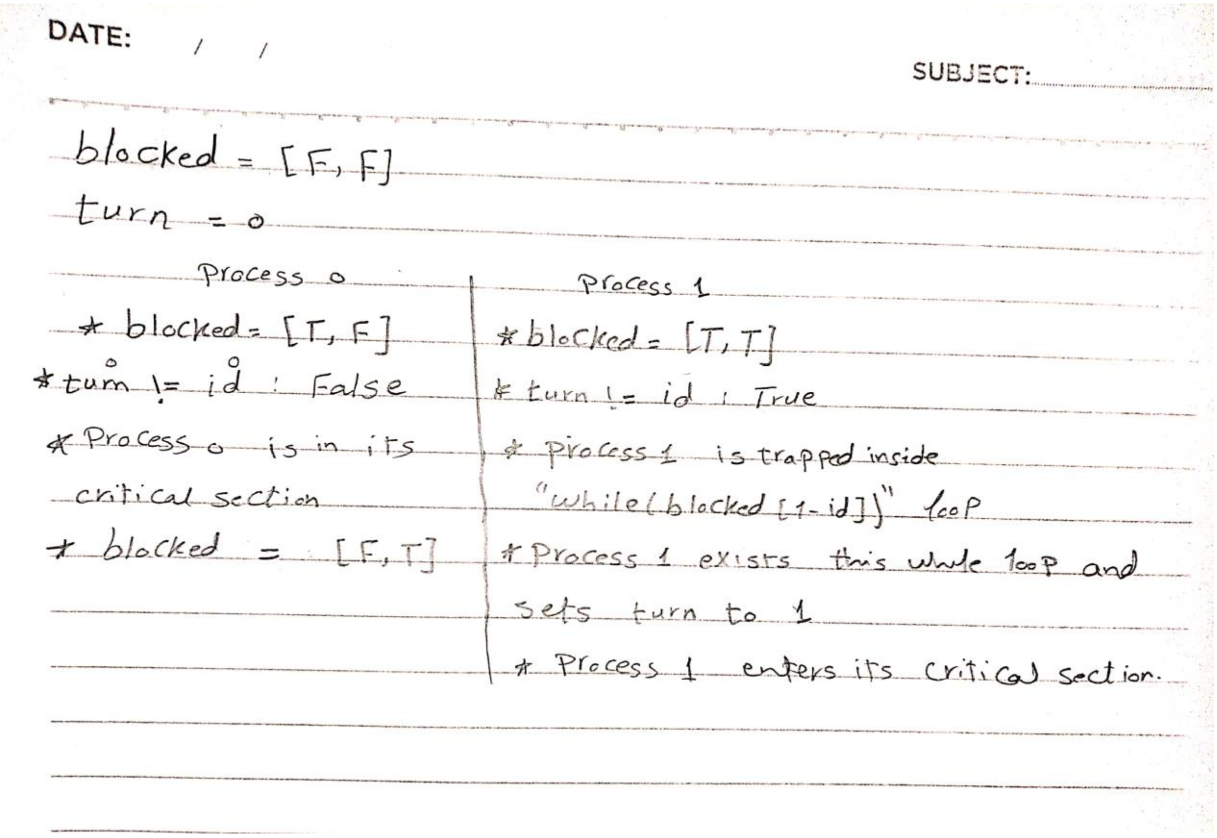
The 2 definitions are interchangeable.

GENERAL QUESTIONS

7-

- a- the lower bound is 50 and the upper bound is 100.
- b- as the number of concurrent processes increases, the range of values of the variable tally will also increase.

8- I don’t think that there is any flaw in this program according to my tracing to the program.



DATE: / /

SUBJECT:

blocked = [F, F]

turn = 0

Process 0

* blocked = [T, F]
* $\overset{0}{\text{turn}}$!= $\overset{0}{\text{id}}$: False
* Process 0 is in its critical section
* blocked = [F, T]

Process 1

* blocked = [T, T]
* turn != id : True
* process 1 is trapped inside "while(blocked [1-id])" loop
* Process 1 exists this while loop and sets turn to 1
* Process 1 enters its critical section.