Concurrency Homework

*Name*____ *Student ID*________________________

1) The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of Process P_i (i == 0 or 1) is shown on the next page. The other process is P_j (j == 1 or 0). Argue that this algorithm satisfies the three goals: safety, progress, and bounded waiting.

```
do {
     flag[i] = TRUE;

     while (flag[j]) {
          if (turn == j) {
               flag[i] = FALSE;
               while (turn == j)
                    ; // do nothing
               flag[i] = TRUE;
          }
     }

     // CRITICAL SECTION

     turn = j;
     flag[i] = FALSE;

     // REMAINDER SECTION
} while (TRUE);
```

Answer: This algorithm satisfies the three conditions of mutual exclusion.
(1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed, namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn.
(2) Progress is provided, again through the flag and turn variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It sets turn to the value of the other process only upon exiting its critical section. If this process wishes to enter its critical section again—before the other process—it repeats the process of entering its critical section and setting turn to the other process upon exiting.
(3) Bounded waiting is preserved through the use of the turn variable. Assume two processes wish to enter their respective critical sections. They both set their value of flag to true; however, only the thread whose turn it is can proceed; the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered—and exited—its critical section. However, Dekker's algorithm has a process set the value of turn to the other process, thereby ensuring that the other process will enter its critical section next.

2) Why are spinlocks not used on uniprocessor systems?

Answer: Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock could be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

3) For the following schedule, tell whether it is conflict serializable or not.

a) *T1:R(X), T2:R(X), T1:R(Y), T2:W(X), T1:W(Y), T2:R(Y), T1:Commit, T2:Commit*

Answer: Yes, this is conflict serializable.

The order of T1:R(x) and T2:W(x) is the same as with the serial schedule T1, T2. The order of T1:W(y) and T2:R(y) is the same as with the serial schedule T1, T2. This schedule is conflict equivalent to T1, T2 and is therefore conflict serializable.

4) (4 points) Fill in the below table.

Fill in the below table using timestamp protocol. Assume the timestamp of the transaction is the same as the Transaction ID. Identify which transactions complete, and which are rolled back. Once a transaction has been rolled back, you do not have to restart it.

T1	T2	T3	T4	R-TS(Q)	W-RS(Q)	R-TS(R)	W-TS(R)
				0	0	0	0
R(Q)				1			
		R(Q)		3			
			W(Q)		4		
	R(R)					2	
	W(R)						2
W(Q)	Since T(i) = 1 and R(Q) = 3 and W(Q) = 4, this transaction is rolled back as a violation of both rules.						
R(R)	Not executed, since T1 was rolled back.						
	W(R)						2
		W(Q)	Since T(i) = 3 and W(Q) = 4, this transaction is rolled back.				
			R(Q)	4			
	W(R)						2
			W(Q)		4		

5) *(4 points) Give a short answer to the following question.*

Given the following schedule, show the locks that will occur and the subsequent schedule. Assume that strict 2PL is in effect, with no deadlock prevention and no starvation prevention. Assume that upgrading locks are allowed. Assume that requests will be satisfied in the order of arrival if possible.

T1:R(X), T2:R(Q), T3:R(Q), T(2):R(X), T3:R(X), T2:W(X), T2:Commit, T3:Abort, T1:W(X), T1:Commit

T1	T2	T3
RL(x)		
R(x)		
	RL(q)	
	R(q)	
		RL(q)
		R(q)
	RL(x)	
	R(x)	
		RL(x)
		R(x)
	WL(x) - holds	
		Abort
		UL(x)
		UL(q)
WL(x) – holds		
The system is now in deadlock.		