CS 6/73201 # Homework #2 **Advanced OS**

Due in class on Monday 22 March 1999
(note new due date)

---

1. **(Exercise 6.5 from *Distributed Operating Systems*) Differentiate between internal synchronization and external synchronization of clocks in a distributed system. Externally synchronized clocks are also internally synchronized, but the converse is not true. Explain why this is so.**

   The goal of internal synchronization is to synchronize all the clocks to each other; the goal of external synchronization is to synchronize one particular clock to an external authoritative time source. If a set of clocks are all externally synchronized to the same time source, then they will become internally synchronized. However, if a group of clocks are internally synchronized, there's no guarantee that they're synchronized to an authoritative time source. In either case, if a set of clocks are synchronized, they will eventually start to drift away from each other, and may need to be resynchronized.

   (Note that this question asked you to differentiate between internal and external synchronization, not to define each of them. If you just defined the two without explaining how they were different, you didn't get very much credit on this problem.)

2. **Must the happened-before space-time diagram be acyclic? Explain your answer.**

   Yes, if there is a cycle between A, B, and C, and A happens before B, B before C, and C before A, then not only does A happen before C (by transitivity) but C also happens before A, which is not allowed. Either one must happen before the other, or they must be concurrent. Furthermore, a cycle would imply that A happens before A, and since "happens before" is an irreflexive partial order, this is not allowed.

3. **(Exercise 6.9 from *Distributed Operating Systems*) Using the space-time diagram of Figure 6.20, list all pairs of concurrent events according to the happened-before relation.**

e1 e5			
e2 e5	e2 e6	e2 e7	e2 e8
e3 e5	e3 e6	e3 e7	e3 e8
e4 e8	e4 e9		

4. **The Ricart and Agrawala mutual-exclusion algorithm can cause a process that only occasionally wants to enter the critical section to waste a lot of time replying to messages from other processes. Explain why this is so, and suggest a solution to this problem.**

   With this algorithm, a process wanting to enter the critical section must obtain replies (indicating permission to enter the critical section) from every other process in that critical section's request set. This means that every process in the request set must reply to every request from any process in that set, even if that process has no desire to enter the critical set itself.

One possible solution to this problem would be to modify the interpretation of the reply message so that a reply means “you have my permission to enter the critical section as much as you want until you get a request message from me”.

Another solution would be to maintain a dynamic request set, rather than the static set discussed in class. However, giving everyone an up-to-date copy of that set is difficult. A central coordinator could maintain it, but then we have very little fault tolerance, performance issues, etc. Instead, maybe some other mechanism could be used to pass the info around.

5. **(Exercise 6.36 from *Distributed Operating Systems*) Initiation of an election is actually needed only when the current coordinator process fails. However, this is not the case in the bully algorithm, in which an election is also initiated whenever a failed process recovers. Is this really necessary? If yes, explain why. If not, suggest a modification to the bully algorithm in which an election is initiated only when the current coordinator fails.**

It is only necessary if it is required that the coordinator is the process with the highest priority. The bully algorithm chooses the process with the highest priority at the time of the election as the coordinator, and as long as everyone agrees that process is the coordinator, it may not be necessary to take that role away from it just because a higher priority process recovers. In this case the recovering process can send out some sort of “who is the coordinator?” message, to which the coordinator will respond.