A file server uses caching, and achieves a hit rate of 80%. File operations in the server cost 5 ms of CPU time when the server finds the requested block in the cache, and take an additional 15 ms of disk I/O time otherwise.

Explaining any assumptions you make, estimate the server’s throughput capacity (average requests/sec) if it is:

i) single-threaded;

ii) two-threaded, running on a single processor;

iii) two-threaded, running on a two-processor computer.

*6.8 Ans.*

80% of accesses cost 5 ms; 20% of accesses cost 20 ms. average request time is 0.8\*5+.2\*20 = 4+4=8ms.

i) single-threaded: rate is 1000/8 = 125 reqs/sec

ii) two-threaded: serving 4 cached and 1 uncached requests takes 25 ms. (overlap I/O with computation). Therefore throughput becomes 1 request in 5 ms. on average, = 200 reqs/sec

iii) two-threaded, 2 CPUs: Processors can serve 2 rqsts in 5 ms => 400 reqs/sec. But disk can serve the 20% of requests at only 1000/15 reqs/sec (assume disk rqsts serialised). This implies a total rate of 5\*1000/15 = 333 requests/sec (which the two CPUs can service).

3. A file server uses caching, and achieves a hit rate of 80%. File operations in the server cost 5 ms of CPU time when the server finds the requested block in the cache, and take an additional 15 ms of disk I/O time otherwise. Explaining any assumptions you make, estimate the server's throughput capacity (average requests/sec) if it is:

(1) single-threaded;

(2) two-threaded, running on a single processor;

(3) two-threaded, running on a two-processor computer.

Ans:

80% of accesses cost 5 ms; 20% of accesses cost 20 ms.

average request time is 0.8 \* 5 + .2 \* 20 = 4 + 4 = 8 ms.

(1) single-threaded: rate is 1000/8 = 125 reqs/sec

(2) two-threaded: serving 4 cached and 1 uncached requests takes 25 ms.

(over lap I/O with computation).

Therefore throughput becomes 1 request in 5 ms.

on average, = 200 reqs/sec

(3) two-threaded, 2 CPUs: Processors can serve 2 rqsts in 5 ms =>

400 reqs/sec. But disk can serve the 20% of requests at only

1000/15 reqs/sec (assume disk rqsts serialised).

This implies a total rate of 5 \* 1000/15 = 333 requests/sec

(which the two CPUs can service).

In a certain system, each process typically uses a critical section many times before another process requires it. Explain why Ricart and Agrawala's multicast-based mutual exclusion algorithm is ine\_cient for this case, and describe how to improve its performance. Does your adaptation satisfy liveness condition in ME2 ?

Answer:

Ricart and Agrawala's algorithm multicast-based algorithm multicasts requests, and requires reply from all other processes before entering a critical section, which is expensive when one process needs several access to the critical section before any other process requests for it.

One possible solution is to change the state from HELD to TEMP instead of RELEASE when the process is done with the critical section. If it needs access again, it can change the state from TEMP to

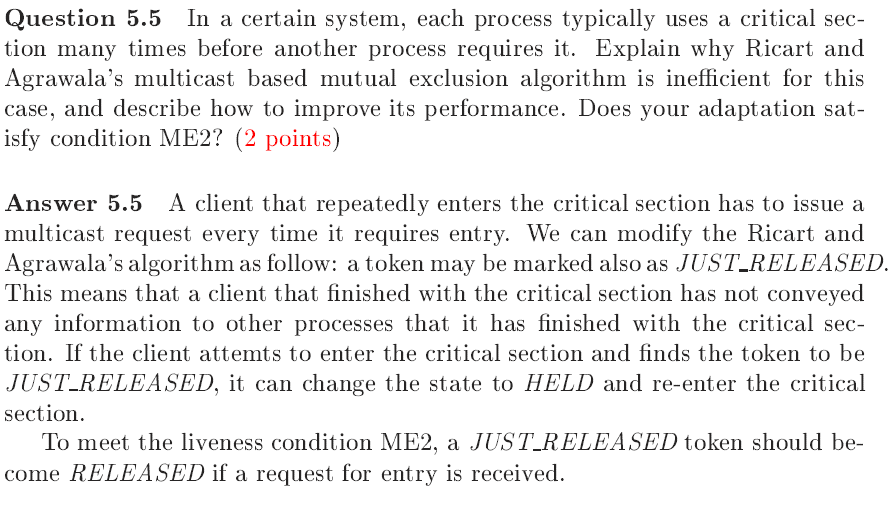
HELD without sending multicast message.

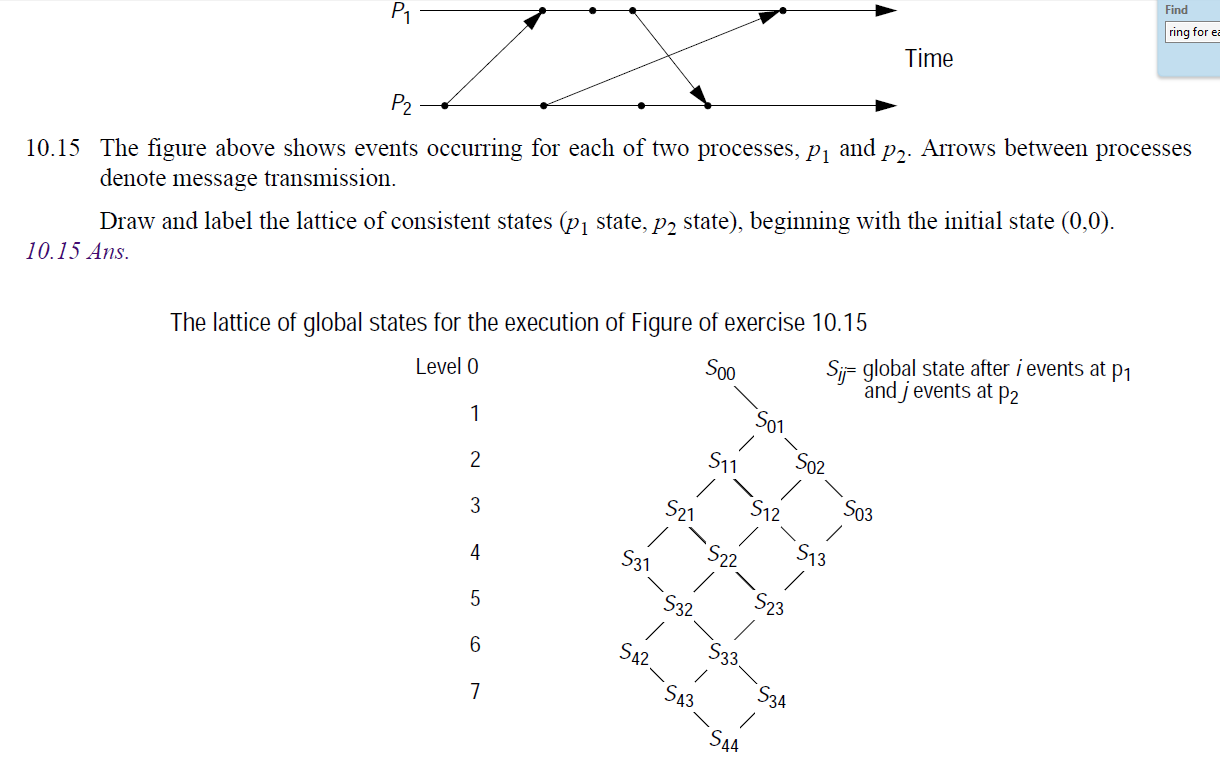
To satisfy ME2 the process has to change state TEMP to RELEASE if there is any request for the critical section, i.e., its queue is not empty.

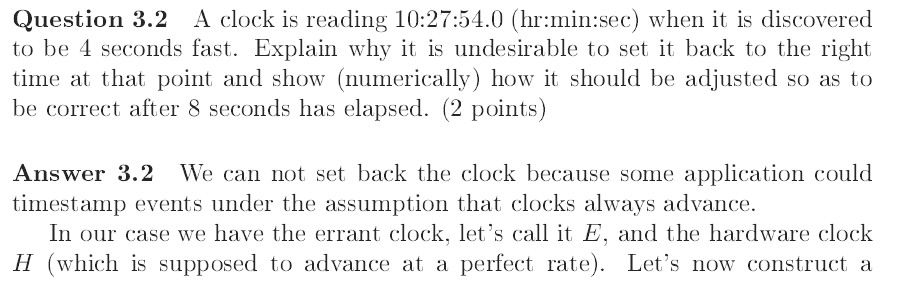
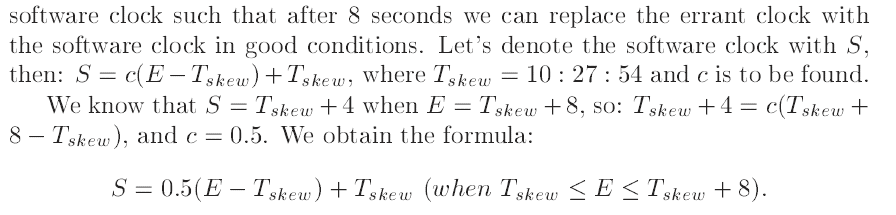
**Q3. (4 points)**. In a certain system, each process typically uses a critical section many times before another process requires it. Explain why Ricart and Agrawala’s mutual exclusion algorithm is inefficient for this case, and describe how to improve its performance.

**Key**: The RA algorithm requires that each entry to the CR needs permission from all other members. If a process needs to enter CR for n times, it is required to send the request n times to group members, which wastes the time and bandwidth.

A possible improvement is: Suppose process P just leaves its CR. If its wants to enter CR again, it can just simple do that if no process in its waiting queue.





10.2 A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast. Explain why it is undesirable to set it back to the right time at that point and show (numerically) how it should be adjusted so as to be correct after 8 seconds has elapsed.

*10.2 Ans.*

Some applications use the current clock value to stamp events, on the assumption that clocks always advance.

We use *E* to refer to the ‘errant’ clock that reads 10:27:54.0 when the real time is 10:27:50. We assume that *H* advances at a perfect rate, to a first approximation, over the next 8 seconds. We adjust our software clock *S* totick at rate chosen so that it will be correct after 8 seconds, as follows:

*S* = *c*(*E* - *Tskew*) + *Tskew*, where *Tskew* = 10:27:54 and *c* is to be found.

But *S* = *Tskew*+4 (the correct time) when *E* = *Tskew*+8, so:

*Tskew*+ 4 = *c*(*Tskew* + 8 - *Tskew*) + *Tskew*, and *c* is 0.5. Finally:

*S* = 0.5(*E* - *Tskew*) + *Tskew* (when *Tskew* ≤*E* ≤*Tskew*+8).