(1) Which process scheduling algorithms can cause “process starvation” (select all that apply)?

(a) FCFS  
(b) RR  
(c) SJF  
(d) SRTF  

**Solutions:** (c) and (d)

(2) How can “race condition” happen? Show “how” using an example.

Race conditions happen when machine codes (or processor instructions) for updating the contents of a shared resource, shared by more than one process, are interrupted (or context-switched) in an unpredictable timing as shown below:
(3) What is “critical section”?

A critical section is a portion of (or a set of instructions in) a program/a process that can cause a race condition.

OR

A critical section is a portion of (or a set of instructions in) a program/a process that must be executed by at most one process at a time (to prevent a race condition).

(4) What does “signal” system call to a semaphore exactly performs?

When a process is leaving a critical section, the leaving process first sees (“lets the operating system to see”) the content of the semaphore. Then:

- If there is a process that is waiting to the semaphore, the OS removes the first process in the FIFO queue for the semaphore and lets the process enter the critical section (the semaphore remains unchanged).
- If there is no process waiting for the semaphore, the OS increases the content of the semaphore.

Note: both of the two descriptions (one for if at least one process is waiting and the other for if there is no process is waiting) are necessary for full credit.

Signal

- If no one waiting on S, set S = 1
- If some one waiting on S, let the first proceed to CS and leave S = 0

(5) Why must the two system calls for semaphores (“wait” and “signal”) be atomic operations (explain the reason)?

If “wait” system call is not an atomic, a race condition occurs when the operating system is updating the content of a semaphore (either for increasing or for decreasing the content of a semaphore). Thus it defeats the purpose of using a semaphore to prevent a race condition from occurring. The same for “signal” system call.