CS 314-001 Operating Systems, Spring 2024
Quiz #6 on February 15, 2024

List of the Possible Questions

#1: Operating systems use "queue (FIFO data structure)" for managing processes blocked on a semaphore. Why is FIFO-queue used (the best reason for using FIFO structure)?

#2: What is "process deadlock"? How is it different from "process starvation"?

#3: If more than one mutex (binary) semaphore is used, can process deadlock occur? If no, explain why not. If yes, explain how using an example.

#4: Why is it difficult to eliminate the condition of "hold & wait" (by applying "request all after you drop what all what you currently hold" method) to prevent a process deadlock from occurring?

#6: Why is it difficult to eliminate the condition of "mutual exclusion" to prevent a process deadlock from occurring?

#6: Why is it difficult to eliminate the condition of "non preemptive resources" to prevent a process deadlock from occurring?

#7: Why is it difficult to eliminate the condition of "hold & wait" (by applying "request all after you drop what all what you currently hold" method) to prevent a process deadlock from occurring?

#8: Which of the following sentence is the correct definition of "safe state" in deadlock avoidance (select the best option)?

(a) From the current state, deadlock will happen, even if some process(es) can finish.
(b) From the current state, there is at least one particular way that lets all the processes complete without causing a deadlock.
(c) The deadlock has already happened.
(d) If deadlock has not happened yet that is a safe state.
(e) None of the above

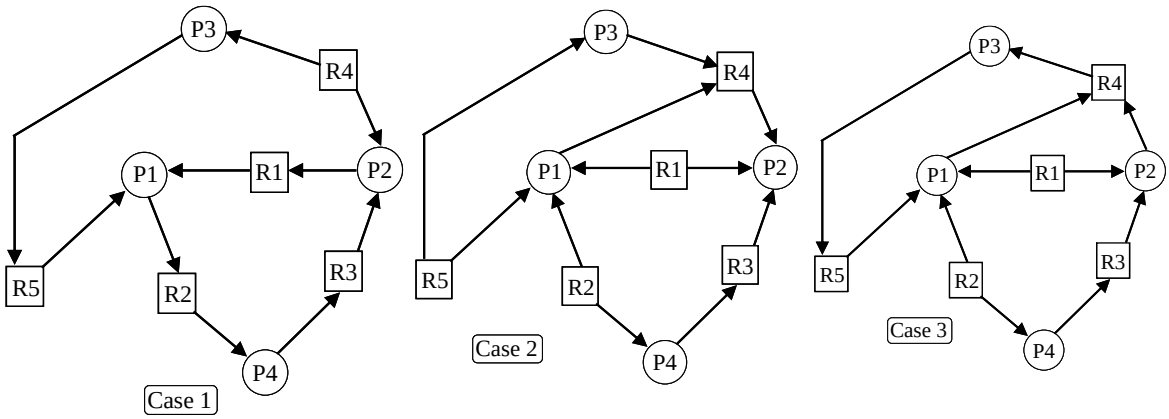
#9: Why is it difficult to eliminate the condition of "circular wait" (by applying "assign resource IDs and require all processes to make requests in the ascending or the descending order" method) to prevent a process deadlock from occurring?

#10: What are the two states in "deadlock avoidance"?

#11: In deadlock prevention, one of the solutions is not to allow any process to "hold & wait" (if a process that holds some non-preemptive resources needs additional resource(s), the process must release all the resources it currently holds and then it requests all the resources (both what it has released and what it additionally needs). Explain how this will prevent deadlocks.

#12: In the classroom, we discussed what we can do to make sure one of the four necessary conditions for a deadlock is not satisfied. Is it possible to have a technique that never causes "circular wait" (other than "all or nothing approach")? If yes, describe how.

#13: For the following situations, which one is (are) in deadlock?



In the above figures, Ri = Resource, Pi = Process, a directional edge from R to P is assignment and a directional edge from P to R is a blocked request (one edge for one resource). You don't need to describe anything. If none of the three is in deadlock, clearly mention so. If the same resource is assigned to more than one process, it means that more than one instance of the resource are available.