Operating Systems Sample Questions

Concurrency: Race Conditions and Deadlocks

1. Define Concurrency. How does it differ from parallelism?

2. Explain the differences between apparent concurrency and true concurrency.

3. What is "atomicity" and why is it important?

4. Briefly explain with examples
   A. Critical Section
   B. Race condition
   C. Deadlock

5. What's wrong with associating locks with critical sections (code) rather than shared resources (data)?

6. Under what situations would you use the following locks and how? In your answer, consider whether the executing code is in process context or interrupt context, and also whether it is running on a multi-processor (SMP) machine.
   A. Blocking locks
   B. Non-blocking locks
   C. Spin locks.

9. When should you NOT use (a) blocking locks, (b) non-blocking locks, and (c) spin-locks?

10. (a) Consider (a) Blocking locks, (b) Non-blocking locks, and (c) Spin locks. Which of these locks can be used in interrupt handlers and how?

11. What is a deadlock? How can you prevent it?

12. What are the following? How can you prevent each of them?
    A. Race Conditions
    B. Deadlocks
    C. Priority Inversion

14. Explain how a deadlock can occur in the operating system between code executing in the user-context and code executing in interrupt handlers. Also explain how you can prevent such deadlocks.

15. Multiple processes are concurrently acquiring and releasing a subset of locks from a set of N locks L1, L2, L3, ….., LN. A process may try to acquire ***any subset*** of the N locks. (a) What is the rule that all processes must follow to guarantee that there would be no deadlocks? (b) Explain with an example where two processes need to acquire ***different but intersecting subsets*** of the N locks above.

16. Assume a multi-processor machine, i.e. a machine with more than one CPU. Consider a critical region function f() in kernel code, i.e. a kernel function that can be invoked concurrently from different parts of the kernel and which accesses shared resources. Explain what kind of locking mechanism — blocking, non-blocking, spin locks, etc — you would use before invoking f() under each of the following situations. If there is more than one possible answer, then explain the reason for your choice.
    (a) f() runs for a SHORT time and can be invoked concurrently from multiple system call functions in kernel, but f() is NEVER invoked from interrupt handlers (I.e. kernel functions that are triggered in response to hardware interrupts).
    (b) f() runs for a LONG time and can be invoked concurrently from multiple system call functions in kernel, but f() is NEVER invoked from interrupt handlers
    (c) f() runs for a SHORT time, and can be invoked from multiple system call functions and ALSO from interrupt handlers.