Operating Systems Sample Questions

Threads

1. What are threads? How do they differ from processes? How are they similar?

2. What state do threads share? What state is different?

3. Why does context-switching between threads incur less overhead than between processes?

4. Briefly explain
   (a) User-level threads
   (b) Kernel-level threads
   (c) Local thread scheduling
   (d) Global thread scheduling

5. Threads vs processes
   A. When would you (as a programmer) prefer to use multiple threads instead of multiple processes?
   B. When would you prefer to use multiple processes instead of multiple threads (in one process)?

6. What are the benefits and disadvantages of using user-level and kernel-level threads?

7. What combinations or user/kernel threads and global/local scheduling are feasible and why?

8. For what kind of applications would you prefer to use (i) user-level threads? (ii) kernel-level threads?

9. Explain how a web server could use threads to improve concurrency when serving client requests.

10. What happens if a thread in a multi-threaded process crashes? How can you improve the robustness (fault-tolerance) of a multi-threaded application?

11. Event-driven programming
    (a) What is the "event-driven" programming model?
    (b) What does the structure of a typical event-driven program look like?
    (c) When would you prefer an event-driven programming model over a thread-based programming model?
    (d) When would you prefer thread-based programming model over event-driven

programming model?

12. What is the problem with long-running event handlers? How do threads solve this problem?

13. What type of applications would be more suitable for thread-based programming compared to event-driven programming?

14. What are callbacks and what problems can they cause when used with threads?

15. (a) How are threads better than processes?
    (b) How are processes better than threads?

16. Assume a single-CPU system. You are given three multi-threaded processes P1, P2, and P3. P1 does a lot of computation, but little I/O. P2 does lots of I/O but little computation. P3 does a equal mix of both computation and I/O. What type of threads (kernel-level, user-level, or hybrid) would you prefer to use for each process? Explain why.

17. Traditional operating systems support the fundamental abstractions of Processes/Threads which are stateful, blocking, and (typically) long-running. Most OS services, such as the CPU scheduler, have evolved to manage process transitions across the process lifecycle. Even event-driven programming is currently implemented using a process that monitors different events in a while loop.

    Consider an alternative OS design in which there is no process or thread abstraction. Instead the fundamental OS abstraction is “event-driven” tasks which are stateless, non-blocking, and (typically) short-lived. Describe how this abstraction could be supported by the OS and what would be the key changes in the OS, particularly for CPU scheduler, I/O processing, and concurrency primitives.