Stanford University Computer Science Department

Fall 2002 Comprehensive Exam in Software Systems

1. **CLOSED BOOK**: no notes, textbook, computer, Internet access, etc.
2. **WRITE ONLY IN BLUE BOOKS: No credit** for answers written on these exam pages.
3. **WRITE MAGIC NUMBER** on the cover of **EACH** blue book.
4. The exam is designed to take less than an hour. Most answers should be quite short.
5. If you need to make assumptions to answer a question, *state them clearly*.
6. For questions that refer generically to "the OS": if you think the answer would differ substantially depending on the OS in question, assume any popular flavor of Unix (Linux, BSD, Solaris, etc.).

Short Answers (3 each, 30 total)

1) With respect to transmission of a message from principal A to principal B, distinguish what is meant by authenticity, privacy, and integrity with respect to that message.

2) What is the point of multilevel paging?

3) Give an example of a high-level language that uses explicit memory management, and state what facilities are provided for doing it.

4) What is an idempotent operation? Give an example of an idempotent operation, and contrast it with an example of a non-idempotent operation.

5) Describe how failure recovery can be made easier by the use of idempotent operations.

6) What's the difference between a preemptive and a non-preemptive process scheduler?

7) If a physical page is shared between two different processes, is it possible for the page to be read-write for one process and read-only for the other? If so, how, and if not, why not?

8) Describe the difference between memory-mapped I/O and programmed I/O. Mention one typical application of each.

9) What is the difference between deadlock and starvation in a resource-allocation system?

10) A RAID system can fail if two or more of its drives crash within the same hour. Suppose that in a given hour, a drive can fail with probability p, and that drive failures are independent. What is the probability of a k-drive RAID system failing in a given hour?

Slightly Longer Answers (5 each, 30 total)

11) When a user program makes a (privileged) system call, in what important way(s) is the calling sequence different from that of calling another user-level procedure?

12) Describe the *priority inversion* problem that arises in process scheduling or thread scheduling. Give a concrete example of circumstances that might cause priority inversion to occur.

13) Suppose you have a word-sized variable *Foo* that is shared among many concurrent threads. You would like to serialize accesses to this variable, to prevent conflicting updates. Unfortunately, you're writing in a language that doesn't provide language-level constructs for writing a monitor procedure (i.e., doesn't have features like Java's *synchronized* keyword).
(a) Assume your OS provides a mutex/locking facility. Using any suitable pseudocode syntax, write the pseudocode for *updateFoo(int newFoo)* using mutexes and/or locks.
(b) Assume your OS provides simple nonblocking atomic operations, like test-and-set or compare-and-swap, on word-sized operands. Write the pseudocode for *updateFoo(int newFoo)* using nonblocking atomic operations. (Note! For part (b), *do not* use nonblocking operations to implement locks! Write true nonblocking code.)

14) Suppose now that *Foo* is not an int but a data structure *FooStruct* that is several words long, and *updateFoo(FooStruct newFoo)* must atomically update the entire contents of this data structure. Can you still write only nonblocking code for *updateFoo*? If so, show the pseudocode. If not, explain why it can't be done. (Note: as in part (b) of the previous question, using atomic instructions to implement locks *does not* count as nonblocking code! Nonblocking code avoids spinwaits.)

15) Suppose that a CPU scheduling algorithm favors those processes that have used the least processor time in the recent past. Explain why this algorithm will favor I/O-bound processes; then explain why it will *not* permanently starve CPU-bound processes.

16) Typically, a true LRU replacement policy can be used for managing blocks in the buffer cache of the file system, whereas only approximations of LRU are used to keep track of pages in a virtual memory system. Why is this so?

17) Suppose a new CPU design has the following feature: there are multiple copies of the register file; at any given time, exactly one of these copies or "banks" can be accessed by user code; and a privileged instruction must be executed to change which "bank" is being used. How might the OS exploit such a CPU feature, and what benefit(s) would be gained?