Stanford University
Computer Science Department

Fall 2004 Comprehensive Exam in Software Systems

1. **Closed book/ no laptops & notes.Write only in the Blue Book.**
2. **The exam is timed for one hour.**
3. **Write your Magic Number on this sheet & on the Blue Book.**

The following is a statement of the Stanford University Honor Code:

A. The Honor Code is an undertaking of the students, individually and collectively:
   1. that they will not give or receive aid in examinations; that they will not give or receive un-permitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

By writing my Magic Number below, I certify that I acknowledge and accept the Honor Code.

Magic Number----------

Stanford University Computer Science Department

Fall 2004 Comprehensive Exam in Software Systems

1. **CLOSED BOOK:** no notes, textbook, computer, PDA, Internet access, etc.
2. **WRITE ONLY IN BLUE BOOKS: No credit** for answers written on these exam pages.
3. **EXPLAIN YOUR REASONING.** Answers with no explanation are insufficient.
4. The exam is designed to take about an hour if you budget 1 point per minute.
5. If you need to make assumptions to answer a question, *state them clearly*.

---

1) [5] Consider a virtual memory system with a single-level page table. The (dimensionless) page fault rate for a particular workload is *r;* the average DRAM access time is *d;* the average time to service a page fault is *f;* the (dimensionless) TLB hit rate is *h*. Write the expression for the average memory access time in this system.

2) [5] Consider three CPU scheduling disciplines: shortest-job-first, preemptive round-robin, and first-come-first-served. For each of these, if it is *starvation-free*, explain why; if it's not, *briefly* describe a scenario in which starvation could occur.

3) [4] Suppose a particular process makes a total of p page references, of which $n<=p$ are to distinct pages. (The ordering of page references is not known.) The process is allocated m frames of physical memory, initially all empty. In terms of these variables, give a *lower bound* and an *upper bound* on the number of page faults this process will experience, no matter what page-replacement strategy is used, and *explain your reasoning.*

4) [4] Debuggers like *gdb* let you set breakpoints that are triggered whenever a program variable, say *V,* is accessed or modified. How is this implemented? Under what circumstances, if any, does the usual implementation incur a performance cost when variables *other* than *V* are accessed? (To simplify the explanation, you may assume that we're only referring to global variables whose memory placement is known at load time.)

5) [4] Give an example of a scenario where memory-mapped I/O makes more sense than programmed I/O, and *why* memory-mapped would be better. Then give an example of the opposite case.

6) [4] To successfully prevent user programs from causing damage to other programs or the OS, hardware support is required. Name **two** hardware mechanisms in modern CPU's that supports this goal, and for each one, describe what specific kind(s) of damage it prevents.

7) [4] Why would some OS's support multiple page sizes instead of just one page size? What additional page-management issues does this raise?

8) [4] Describe the mechanism of *priority inheritance* and give an example of the kind of problem it's intended to solve.

9) [3] With respect to remote procedure calls (RPC), what is serialization (or marshalling) and why is it necessary? Are there cases where it is unnecessary?

10) [3] Describe one failure mode that might occur if a non-preemptive scheduler is used, and how it would be avoided with a preemptive scheduler.

11) [3] A particular email message you're sending is so sensitive that you wish to both encrypt and sign it (both using public-key cryptography). Under what circumstances, if any, would you encrypt it first and then sign it? Under what circumstances, if any, would you sign it first and then encrypt it? (In other words, what's the practical effect of doing it one way vs. the other?)

12) [3] True or false: all side-effect-free operations are idempotent, but not all idempotent operations are side-effect-free. Explain your answer concisely but completely (i.e. if true, explain why each part is true; if false, explain which part(s) are false and/or give counterexamples).

13) [2] To avoid replay attacks, one can either use a randomly-generated nonce or a physical timestamp. Give one advantage of using a nonce over a timestamp, and one advantage of using a timestamp over a nonce. (You may assume that the resolution of physical timestamps is sufficient to avoid timestamp value collisions.)

14) [2] Describe one type of file access control that can be performed with access-control lists (such as AFS uses) but cannot be performed with file permissions (such as traditional Unix filesystems use).

15) [2] You're asked to take an existing Java program and rewrite it in C++. What benefit would you *gain* by doing so? What benefit, if any, would you *lose* by doing so?

16) [2] What's the difference between a credential and a capability?

17) [2] What's the difference between thrashing and deadlock?

18) [2] Give one example of how a filesystem might become corrupted, ***other than corruption of data in the files themselves***

19) [2] Your C program has a bug that causes it to accidentally dereference a nonexistent array element, e.g. a[10] where array a[] has been statically declared as containing 8 elements. What happens when this bug occurs at runtime, and why?

THE END