Computer Science Department

Stanford University

Comprehensive Examination in Software Systems

Autumn 1996

READ THIS FIRST!!!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.
2. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even if you feel you could expound at length upon it.
3. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.
4. This exam is CLOSED BOOK. You may NOT use notes, books, computers, other people, etc.
5. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out. You can also get credit for realizing that certain approaches are incorrect.
6. If you are convinced you need to make an assumption to answer a question, state your assumptions(s) as well as your answer.
7. Be sure to provide justification for your answers.

1996 Comprehensive Exam: Software Systems (60 points total)

1. (15 points) Solaris 2 uses "adaptive mutexes" to protect access to every critical data item. An adaptive mutex starts as a standard semaphore implemented as a spinlock. If the data are locked (already in use), the adaptive mutex does one of two things. If the lock is held by a thread that is currently running, the thread waits for the lock to become available. If the thread holding the lock is not currently in the run state, the thread blocks, going to sleep until it is awakened by the lock being released.

   a. Why are both options (spinning and sleeping) provided when the data are already locked, and why does the decision depend on whether the thread holding the lock is running?

   b. For what sort of application workload is this feature beneficial?

   c. On a uniprocessor system, how do adaptive mutexes behave?

2. (10 points) Page replacement in a virtual memory system can be either global or local. In global page replacement a page fault is satisfied by selecting a page from the set of all physical pages. In local page replacement, the page fault can only be satisfied from the faulting process' own set of physical pages. The number of physical pages allocated to a process in the local scheme does not change over its lifetime.

   a. What are the potential disadvantages of each scheme?

   b. Why is global replacement more commonly found in existing systems?

3. (10 points) Monitors are a popular mechanism for ensuring synchronization of processes that access system data structures and resources. However, monitors do not guard against starvation.

   a. Briefly describe why monitors do not guard against starvation.

   b. Given this problem, why is it practical to use monitors in real-world systems? (How is the starvation problem handled in practice?)

4. (20 points) File cache update policies can be categorized by the amount of delay they allow before file updates must be written to disk. Compare the following policies based on reliability, process latencies, and disk throughput. For each policy, also describe a workload that would benefit from the specific policy.

   •Write-through immediately (no delayed writes)

   •Write-through on close (write modified data to disk when the file is closed. The close operation doesn't return until the write finishes. If a process with open files exits, the system calls the close operation on these files before the exit completes.)

   •Delay for 30 seconds (write out modified data from the cache when it's 30 seconds old)

   •Delay for 5 minutes (write out modified data from the cache when it's 5 minutes old)

5. (5 points) In UNIX, there are separate system calls to start a new process in the image of its running parent (`fork`) and to load into a process's address space fresh code and data from an executable that it should use for execution (`exec`). Why doesn't UNIX use just one system call that both creates a new process and loads the desired image (code and data)?