Systems Qualifying Exam

January 18, 2000

1. [20 pts] Architecture (parts a–d)

   Modern CPUs, such as the Pentium III, use pipelining, speculation, and superscalar (*i.e.*, parallel) datapaths to achieve impressive performance at high clock rates for common-case code. Many current CPUs are able to run at clock rates of 750 MHz or greater and can complete 4-8 instructions per cycle by taking advantage of instruction-level parallelism. However, in realistic programs, the average number of cycles per instruction (CPI) is far greater than this peak performance.

   (a) [4 pts] The Pentium III has only 8 general purpose registers. Explain why having more registers might improve the average CPI.

   (b) [5 pts] Explain why having more general purpose registers might slow the execution of a program that frequently communicates with the operating system.

   (c) [5 pts] Explain why conditional branches limit instruction-level parallelism.

   (d) [6 pts] The Unix `memcpy` function copies a contiguous block of data from one area in memory to another, using a simple loop. On many systems, the CPI for two calls of the form `memcpy(dest, src, n)` can differ by a factor of 100 for different values of the memory pointers `dest` and `src`. This happens even when in both calls, `n` has the same small value, and `dest` and `src` point to word-aligned locations in ordinary memory. Why does this happen?

2. [20 pts] Security (parts a–e)

   It is often useful to perform procedure calls that cross protection domains. In general, the domains of the caller and callee will have different privileges (access rights).

   (a) [4 pts] Why might we want the callee not to possess all the privileges of the caller? Give an example.

   (b) [4 pts] Why might it be useful for the callee to have privileges not possessed by the caller? Give an example.

   (c) [4 pts] Why might it be useful for the protection domain of the callee to be augmented *for the duration of the call* with additional privileges transferred from the caller? Give an example.

   Ben Bitdiddle is designing a filesystem for a new version of Linux running on the new Intel Itanium chip. In his design, files are protected by capabilities that are stored on disk with the file. Each capability is represented by a distinct, randomly-generated 32-bit number, and the operating system permits a process to open a file only if that process can supply the correct capability for that file (The system call interface is extended to allow a capability to be supplied to the `open()` call.)

This design has the advantage that the owner of a file can give another party access to the file simply by transmitting the capability as a 32-bit number (encrypted, if the channel is insecure.) Each file actually has two capabilities: one to control reading and the other to control writing.

Assume that the numbers encoding capabilities are randomly generated with uniform distribution by special hardware whose output cannot be predicted or intercepted, that the disk cannot be read or inspected except through the operating system interface described, and that the owner of a file is able to obtain the necessary numbers protecting that file.

(d) [4 pts] Describe a straightforward attack on the security provided by this implementation, and propose a conceptually simple way to fix the vulnerability.

(e) [4 pts] Consider the domain-switching scenarios of 2(a)–2(c). Will Ben's capability-based filesystem design, in which capabilities are stored and transmitted by user-level processes, create any difficulties in implementing these scenarios? Discuss briefly.

3. [20 pts] Page Management (parts a–e)

   Suppose that a large file is organized as a binary tree of nodes and that you are using a procedure that searches the tree. Each node is the size of one page. Every operation on the tree involves a search that starts at the root. Suppose that this tree is the only paged data segment in your program.

   (a) [3 pts] Give a brief definition of the term "working set."

   (b) [3 pts] Which pages of the tree are likely to belong to the working set?

   (c) [5 pts] Suppose that the tree has depth 500, that most searches descend all the way to a leaf node, and that the operating system can assign only 250 page frames of physical memory to your program. What problem might arise if the LRU page replacement algorithm is used to manage this memory?

   (d) [4 pts] Suppose that conditions are as in part 3(c), except that now 750 pages of physical memory are available for your program. Now would LRU work significantly better as a page-replacement policy? Explain briefly.

   (e) [5 pts] Suppose you had the option of implementing your own page-replacement policy. What policy would you propose? Would it significantly out-perform LRU?

4. [20 pts] Concurrency (parts a–d)

   Below is a skeleton for a 2-process concurrent program in which process `Prod` communicates 1000 values to process `Cons`. In particular, `Cons` constructs a list in variable `out` using values written by `Prod`. A shared buffer, `buf`, which consists of a single word of memory, is used for inter-process communication between `Prod` and `Cons`.

Angle brackets are used below solely to delimit segments of code that are not expanded but might need to be.

```
var buf : integer

Prod:  process
       var i : integer
       for i:=1 to 1000
              <seg_1>
              Deposit: buf := i
              <seg_2>
              end
       end Prod

Cons:  process
       var j : integer
       for j := 1 to 1000
              <seg_3>
              Remove: out := out CONCAT buf
              <seg_4>
              end
       end Cons
```

Assume that load and store to shared memory locations are the *only* atomic operations on this processor and that busy-waiting is the only way to delay execution of a process.

(a) [5 pts] Describe, in English, constraints on execution of Deposit and Remove that ensure the variable out ultimately contains the sequence 1 2 3 4 . . . 1000.

(b) [6 pts] Now give code for <seg_1> through <seg_4> and define additional variables required to accomplish the synchronization that ensures the variable out ultimately contains the sequence 1 2 3 4 . . . 1000.

(c) [3 pts] The program of 4(b) is being used to benchmark a new, experimental uni-processor operating system kernel. For both Prod and Cons, you carefully measure the time it takes to execute one iteration of the respective loop body. Both loop bodies take about 1 usec when run in isolation. Based on this measurement, what total execution time (**in milliseconds**) would you expect for the concurrent execution of Prod and Cons?

(d) [6 pts] When Prod and Cons actually run together, the total measured execution time is considerably larger than predicted in 4(c) — seconds rather than milliseconds. What is the likely explanation, and what practical measures can be employed to remedy the problem?

5. [20 pts] Databases (parts a–e)

   (a) [4 pts] The recovery manager of a database management system performs logging so that recovery from system crashes is possible. Why does the recovery manager force-write the log to disk before a transaction commits, rather than writing the log in a way that maximizes overall system performance?

   (b) [4 pts] For a transaction T, let the set P(T) be the set of pages that T modifies. As a simple example, for a read-only transaction P(T) is empty. Consider a "no-steal" policy for the buffer manager: While a transaction T is active, no pages p in P(T) are written to disk. This policy permits a simplification of both the recovery manager and the information stored in transaction log records. What is this simplification?

   (c) [4 pts] Consider two relations, $R(X, Y)$ and $S(Y, Z)$; that is, R has attributes X and Y, and S has attributes Y and Z. The relation R resides on $|R|$ disk pages and S resides on $|S|$ disk pages. R and S share a common single attribute Y. Define (as a set of tuples) the natural join between R and S: R natjoin S.

   (d) [4 pts] Assume you want to compute the natural join between R and S. Describe how the sort-merge join algorithm works and describe how the hash-join algorithm works. You may assume that the buffer pool contains B pages of physical memory, and that an unlimited amount of disk space is available for temporary storage. Describe only the case in which B is large enough to perform the join in two passes with a total cost of $3 \cdot (|R| + |S|)$ disk page accesses. (This cost does not include the cost of writing the result of the join.)

   (e) [4 pts] Just considering the quantity B, when would you choose the sort-merge algorithm for best performance, and when the hash-join algorithm? As in 5(d), assume that the buffer pool has B pages of memory that are available for processing the natural join between R and S. Also assume that $|R| < |S|$ and that each tuple in R joins with approximately $|S|/|R|$ tuples in S.