

CS 574

Midterm Exam

October 12, 2001

Please note the following instructions. There will be a **ten point deduction** for failure to comply with them:

- This exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no calculators** are allowed.
- start each problem on a new sheet of paper.
- write your social security number, but not your name, on each sheet of paper you turn in.
- be succinct. I will take points for facts that, while true, are not relevant to the question at hand.

You have until 10:30 to finish the exam. The questions are equally weighted.

1. In the Clouds paper, a comment is made to the effect that file abstractions such as those used by Unix and Plan 9 are “hardware-based.” Discuss whether or not this comment is accurate.

In my opinion, no. It's not true of real files; the file abstraction presented to the user is pretty far removed from the actual hardware representation. It's even less true when you consider things like procfs and devfs; the “file” abstraction is really just a hierarchical name space with a uniform low-level data interface. On the other hand, files are restricted to being modelled as linear arrays of bytes, which is an abstraction derived from disk storage.

I suspect that a lot of students hadn't really thought about what it means for the different systems to provide different abstractions; at any rate, I got a lot of answers that had nothing to do with the abstractions. In particular, the fact that the Unix file system is implemented using hardware doesn't imply that the abstraction is hardware-based!

A number of students described the implementation of the Unix file system (i-nodes and stuff), and claimed that this implies that the abstraction provided is hardware-based. No, this shows that the implementation is a data structure, which has nothing to do with whether the abstraction is hardware-based

Some others pointed out that Plan 9 supports heterogenous networks, as an argument that it isn't hardware-based. Again, this has nothing to do with the abstraction.

Oddly, I got answers that used the facts in these previous arguments to argue for the exact opposite conclusions, too!

- Both FFS and Reiserfs set a goal of better performance on small files. Which is more successful in achieving this goal? Why?

Reiserfs - FFS fragments are still pretty big; Reiserfs would fit several small files into a single fragment, giving better utilization and better bandwidth.

Does the balanced tree structure really improve performance of small files?

- The DES standard makes the point that the algorithm is well-suited to a hardware implementation. What features of the algorithm make this true? Why is RSA less well-suited to a hardware implementation than DES?

DES applies a sequence of simple operations - just fixed permutations (easy to implement with nothing but wires) and exclusive-ors (easy to implement in a couple of gates). It doesn't do any complicated arithmetic. On the other hand, RSA is all about complicated arithmetic; you'd really want to avoid doing it in hardware (though a "hardware" implementation really done in firmware would be feasible).

I got a number of answers which imply to me that some people in the class aren't clear on the nature of public-key encryption. The fact that one of the two keys is public doesn't reduce security at all. For as long as the only known ways of factoring large numbers amount to a brute force attack, saying that the algorithm depends on the difficulty of doing this doesn't imply that the algorithm isn't secure!

- Netscape supports the use of encryption for transmission of sensitive information (such as credit card numbers). At one time, the keys it used were based on requesting the operating system for the current time of day. Speculate on why this turned out to be a bad idea.

It's too easy to guess. Lots of programs use current time of day as a random number seed, so it's no big stretch for somebody to guess that Netscape might. Even without a guess, running a strace on Netscape would reveal that gettimeofday was being called, which would inspire someone malicious to make the guess. For that matter, disassembling the code isn't that big a problem. Once you know the key is based on the current time of day, you've got a vastly reduced set of possible keys to try in decrypting a message.

It should be mentioned that this was in Netscape 1.1 - a very long time ago!