Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Spring 2004

Midterm Exam Answers

The average score was 86, the standard deviation was 12.

I Porcupine

Consider the Porcupine e-mail server described in *Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service*. The Porcupine User Map is a table that maps the hash of a user name to the identity of the server responsible for that user's profile and fragment list. Porcupine uses code like this to find the host name of the server that is responsible for a user:

```
int UserMapSize = 20;
String[] UserMap = new String[UserMapSize];
// ...
String user2server(String username) {
  int h = username.hashCode();
  h = h % UserMap.length; // % is the modulo operator
  return UserMap[h];
}
```

1. [10 points]: Suppose, by an amazing coincidence, all users' names hashed to 13 modulo UserMapSize. Compared to an even distribution of hashes, would this significantly reduce the rate at which Porcupine could receive messages? Why or why not?

Not significantly. It is true that one server would have to do all the lookups to find out where users' fragments and profiles were stored. But the load of storing and retrieving messages would still be spread over all the servers. Figure 6 in the Porcupine paper shows that overall performance is hardly affected by extreme imbalances in the hashing.

Suppose that Porcupine didn't have a User Map at all, but rather directly hashed the user name to decide which server to use. That is, suppose Porcupine numbered its N servers 0 through N-1 and used this code to decide which server was responsible for a user's profile and fragment list:

```
int user2server(String username) {
  int h = username.hashCode();
  int s = h % NServers;
  return s;
}
```

2. [10 points]: Why might this scheme adversely affect Porcupine's efficiency? Hint: think about what would have to happen when a new server was added and NServers was increased.

A new server would cause most of the user to server mappings to change, so that most fragment lists and profiles would have to be copied from one server to another.

II Echo

You're using a workstation with files stored on an Echo file server, as described in *A Coherent Distributed File Cache with Directory Write-Behind* by Mann *et al*. Your workstation is named W1, your current directory is /d, and /d starts out with no files or subdirectories in it. You type commands which cause the following file system operations to occur, one at a time (this notation is the same as in Figures 4 through 7 of the Echo paper):

```
create(/d/f1)
append(/d/f1, aa)
create(/d/f2)
append(/d/f2, bb)
rename(/d/f2, /d/f3)
append(/d/f3, cc)
create(/d/f4)
append(/d/f4, dd)
```

You then read file f4 on W1 and observe that it contains "dd". You stand up and leave, but as you walk away from the workstation you accidentally trip over W1's power cord and unplug it, so it instantly stops. You walk over to workstation W2 (connected to the same Echo server) and look at the files in /d. During this whole time, no other user or process is modifying the file system, and there are no failures other than the crash of W1. The software has no bugs in it. Keeping in mind the guarantees stated in Section 3.1 of the Echo paper, answer the following questions about the possible states of the files in /d that you might see on W2.

3. [2 points]: Is f4 guaranteed to exist (that is, to appear in the directory /d)? **No, since W1 might not have written it from its write-back cache to the server before the crash.**

4. [2 points]: If f4 exists, is f1 guaranteed to be non-empty? **No. Echo only guarantees ordering for writes to the same object, and f1 and f4 are different objects.**

5. [2 points]: If f4 exists, is f3 guaranteed to be non-empty? **Yes. The following individual orders imply that Echo must send the contents of f2 to the server before sending the create of f4: the rename "writes" f2 and must come after the append, and the create(/d/f4) "writes" /d and must come after the rename.**

6. [2 points]: Could f2 exist and contain exactly "bb"? **Yes. The workstation might have crashed after it sent the append to the server, but before sending the rename.**

7. [2 points]: Could f2 exist and contain exactly "bbcc"? **No. Echo guarantees to send the rename to the server before the append(/d/f3, cc): they both write f3, so Echo must order them.**

Now suppose that you had been using an NFS server, as described in *Design and Implementation of the Sun Network Filesystem*, by Sandberg *et al*. You do all the same things, using a UNIX program that looks like this:

```
fd1 = creat("/d/f1", ...);
write(fd1, "aa", 2);
close(fd1);
fd2 = creat("/d/f2", ...);
write(fd2, "bb", 2);
close(fd2);
rename("/d/f2", "/d/f3");
fd3 = open("/d/f3", ..., O_APPEND);
write(fd3, "cc");
close(fd3);
fd4 = creat("/d/f4", ...);
write(fd4, "dd", 2);
close(fd4);
```

Again, you read f4 and observe that it contains "dd", accidentally unplug W1's power, and look at /d on W2. Based on the design and implementation described in the NFS paper, answer these questions about the state of the files in /d as seen on workstation W2.

8. [2 points]: Is f4 guaranteed to exist? **Yes. The NFS paper does not mention any write-behind or caching for directory modifications, so the client must send the CREATE RPC to the server. Furthermore, the paper says that close() does not return until the server acknowledges all WRITEs, and the WRITE RPCs cannot be issued until the CREATE has returned because the WRITE needs to know the file handle.**

9. [2 points]: If f4 exists, is f1 guaranteed to be non-empty? **Yes. The program doesn't create f4 until close(fd1) returns, which guarantees that the server knows about the write to f1.**

10. [2 points]: If f4 exists, is f3 guaranteed to be non-empty? **Yes, for the same reason as the previous question.**

11. [2 points]: Could f2 exist and contain exactly "bb"? **No. The fact that the program finished means that the rename and write of "cc" completed.**

12. [2 points]: Could f2 exist and contain exactly "bbcc"? **No, for similar reasons as the previous question.**

13. [8 points]: Outline a simple scenario in which a client using Echo would likely experience much better performance than a client using NFS.

A single client that repeatedly opens, overwrites, and closes the same file. Creation and writing of a series of distinct files probably doesn't run much faster in Echo than in NFS because in the end the Echo client does have to send the data to the server, though it is true that Echo avoids stalling the application in each close().

Now suppose that you had been using a workstation with a local disk using the FSD file system, as described in *Reimplementing the Cedar File System Using Logging and Group Commit*, by Hagmann. Note that FSD is a local disk file system; there is no network and no server. You perform the same operations as in the Echo scenario, including reading f4 and seeing that it contains "dd", and then you accidentally unplug the workstation's power cord. You're very agile and the whole sequence takes only about half a second. You plug the workstation back in, it reboots, it runs the FSD recovery program, and you then look at the files in /d.

14. [2 points]: Is f4 guaranteed to exist? **No. FSD might not have written that part of the the in-memory log to disk yet.**

15. [2 points]: If f4 exists, is f1 guaranteed to be non-empty? **Yes. FSD file data writes are synchronous, so f1's contents are guaranteed to be on the disk before the log records for the creation of f1 or f4.**

16. [2 points]: If f4 exists, is f3 guaranteed to be non-empty? **Yes, for much the same reason as the previous question.**

17. [2 points]: Could f2 exist and contain exactly "bb"? **No. f2 could still exist, because the rename log entry might not have been written to disk before the crash. But since file data writes are synchronous, the "cc" must have been written to disk, so if f2 (or f3) exists it must contain "bbcc".**

18. [2 points]: Could f2 exist and contain exactly "bbcc"? **Yes.**

19. [2 points]: Could both f2 and f3 exist? **No. The paper explains that much of the point of using the log was to ensure atomic operations on the file name table.**

III Tamper-Evident Boot

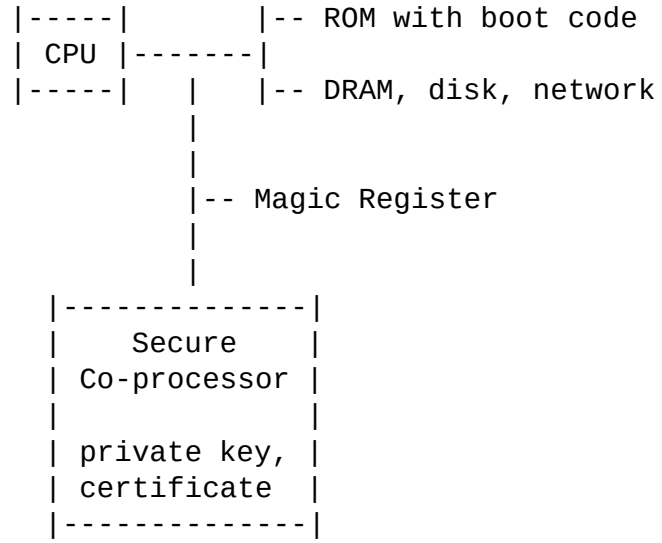
Alyssa P. Hacker operates the YETI@home project, a giant distributed Internet computation whose goal is to simulate the behavior of the hypothetical Yeti (a giant ape-like creature) in order to guess where on Earth it might be found. Alyssa's basic approach is to divide her simulations into small pieces and farm them out to legions of Internet enthusiasts who are running her application on their home PCs. When a participating machine finishes its piece of the simulation, it reports its result back to Alyssa's central server and fetches a new unit of work. Alyssa's server keeps track of which participants perform the most work, and posts the names of winners on her web site.

Sadly, Alyssa finds that some participants are cheating, apparently by modifying their copy of her software to return a random result to her server, without actually doing the computation. Alyssa has no way to check the correctness of the reported results, other than doing the computation herself, which would be too time-consuming.

Alyssa searches the Computer Science research literature looking for a potential solution. She reads about XOM (in *Architectural Support for Copy and Tamper Resistant Software*), which seems like it might be able to solve her problem. However, no actual XOM hardware exists, the design seems complex enough that such hardware might be a long time coming, and besides Alyssa only needs to be able to detect whether her application has been tampered with; she doesn't need copy protection. Furthermore, Alyssa doesn't need a tremendously high level of security: she just wants to raise the level of effort required for a participant to send in a fake result.

Alyssa invents what she calls "Tamper-Evident Boot." She believes that it will help her solve her problem, and she thinks that her design will be easy to deploy because it involves no changes to existing PC CPUs and just a few simple additions to PC motherboard designs.

Here's Alyssa's hardware architecture:



This is a component view of a motherboard. The CPU is an ordinary CPU chip, such as an Intel Pentium. The connection to the DRAM and boot ROM is an ordinary memory/device bus. Alyssa's design adds two new devices to the motherboard, connected to the bus.

The Magic Register holds 160 bits of data (the size of the output of a typical cryptographic hash), but the CPU can write it only once after each power cycle or reset. That is, the Magic Register contains a flag; the flag is cleared to zero after a reset or power cycle, the Magic Register sets the flag to one after a write, and the Magic Register ignores writes if the flag is already equal to one. The CPU can read the Magic Register at any time.

The Secure Co-processor is a simple microprocessor chip that contains public-key algorithms, a private key, and a certificate. All of the Secure Co-processor's code is in a read-only memory inside the co-processor chip. The private key is stored inside the co-processor in a way that's difficult for anything but the co-processor's software to read. The manufacturer gives every Secure Co-processor a unique private/public key pair and a certificate. The certificate is signed by the manufacturer's private key, and includes the co-processor's public key; we'll assume that the manufacturer's public key is well known.

The Secure Co-processor supports just one operation: the CPU can send it some data over the device bus, and the Co-processor will return a copy of the data and and its certificate and the contents of the Magic Register, signed by the Secure Co-processor's private key. That is, the Co-processor implements this function for the CPU:

```
attest(data) {
  struct scp_msg scpm;
  scpm.cpu_data = data;
  scpm.register = Magic Register contents;
  scpm.cert = co-processor's certificate;
  scpm.signature = Sign(scpm, co-processor's private key);
  return scpm;
}
```

Alyssa's intent is to persuade computer manufacturers like Dell to include her new hardware in every PC. She believes this hardware can solve a wide range of security problems at minimal expense.

Alyssa is not trying to defend against attackers who modify the hardware or extract the private key from the co-processor. We'll assume for all questions below that no-one has modified any hardware or stolen any private keys.

20. [10 points]: Suppose that software running on a PC with Tamper-Evident Boot calls attest("I've correctly computed YETI@home result XYZ") and sends the resulting signed message to Alyssa's server. The server observes that the signature on the message matches the public key in the certificate, and that the manufacturer's signature on the certificate is correct. On the basis of the contents of the message and these signature checks, is it reasonable for Alyssa's server to believe that XYZ is a correct YETI@home result? Briefly outline why or why not.

No. Any program running on Alyssa's hardware might have produced this message and included an incorrect result; there's no reason to believe the creator of the message was her application.

21. [10 points]: Is it reasonable for Alyssa's server to believe that the message was generated by a PC with Alyssa's hardware? Briefly outline why or why not.

Yes. Only a Secure Co-processor could produce a signature with a key certified by the manufacturer. (You have to believe that no-one has compromised any Secure Co-processors in order to believe this, but we're assuming no hardware attacks.)

Here's how Alyssa intends the hardware to be used to implement Tamper-Evident boot. Every motherboard comes with a read-only memory (ROM) that holds some boot code provided by the manufacturer. The CPU starts executing the instructions in this ROM after a power-cycle or reset. The job of the boot code is to read the operating system from the hard disk into main memory, and then jump into the operating system to start executing it. Alyssa's boot ROM does one extra step before entering the operating system: it computes a cryptographic hash of the entire operating system it just loaded from the disk, and stores that hash into the Magic Register.

Alyssa intends to modify the operating system software to provide a new os_attest() system call to applications. os_attest() takes an argument with application-supplied data, and causes the Secure Co-processor to sign that data along with a cryptographic hash that the kernel computes over the application's executable file:

```
os_attest(data) {
  struct os_msg osm;
  osm.app_data = data;
  osm.app_hash = cryptographic_hash(application's original executable file);
  return attest(osm);
}
```

The operating system would instruct the CPU hardware to prevent applications from using the attest() functionality of the Secure Co-processor device.

When Alyssa's server receives a message, it checks that scpm.cert is correctly signed by the manufacturer, that scpm.signature is correctly signed by the public key in the certificate, that scpm.register is the hash of an "approved operating system," and that osm.app_hash is the hash of Alyssa's application.

22. [10 points]: What are the key properties an operating system would have to have in order that it be reasonable for Alyssa's server to consider an "approved operating system?"

If a given executable file's hash appears in osm.app_hash, then the operating system should ensure that osm.app_data is from an undisturbed execution of that executable file. This is a pretty broad requirement, but example properties might include preventing applications from calling attest() directly and giving each application its own protected address space using virtual memory.

23. [10 points]: Suppose that the Magic Register did not reject all but the first write after each reboot or reset; that is, suppose that it allowed any number of writes at any time. What new attacks would be easy to perform as a result of this change?

A non-approved operating system could write the hash of an approved operating system to the Magic Register, and then ask the Secure Co-processor to sign messages that claimed to be from "correct" executions of Alyssa's application.

End of Exam

6.824 Questionnaire

We'd like to ask you for feedback to help us make the course better, both for the rest of this semester and in future years. Feel free to tear off this page and hand it in separately from the exam for anonymity.

If you could change one thing about 6.824 to make it better, what would it be?

Get the paper questions out on time.

What's your favorite aspect of 6.824? That is, what *shouldn't* we change?

The labs.

What could we do to make the paper discussions more informative and useful?

Introduce each paper, and cut off lengthy argument over uninteresting details.

Any other suggestions about the labs, lectures, TA, choice of papers, or projects?

Fewer disk file-system papers. Fewer Network Objects papers.