Student ID: ____________

CS-736 Midterm: Beyond Compare (Spring 2008)

An Arpaci-Dusseau Exam

Please Read All Questions Carefully!

There are eight (8) total numbered pages

Please put your NAME ONLY on this page, and your STUDENT ID on this and all other pages. Why? So I can grade without knowing who you are. Particularly useful for students who think I don't like them. Note: I like all of you, so this is not really a problem.

Name: ____

Student ID: ____________

This is the grading page.

	Points	Total Possible
1		5
2		5
3		5
4		5
5		5
6		5
7		5
8		5
9		5
10		5
11		5
12		5
13		5
14		5
Total		70

"*Comparisons do ofttime great grievance.*" -John Lydgate

We have read about a lot of individual systems. But is the whole greater than the sum of the parts? In this exam, we tackle this question, by comparing systems. As defined in the dictionary:

```
comparison /n./
1. the act or instance of comparing: the two operating systems invite
comparison with one another.
an analogy: perhaps the best comparison is that of seasickness, or that
feeling you get during a 736 exam.
the quality of being similar or equivalent: if you want a thrill, there
is no comparison to taking a class with Remzi.
```

Thus, on each question, I will not simply ask you a question about a single system. No, that is too easy, a tactic for other classes less talented than this one. Rather, I ask you a question that compares one system with another. More details can be found in each question, of course.

NOTE: Please read each question CAREFULLY. Also, keep your answers short and to the point; there are no style points for long-winded answers (rather, quite the opposite!).

You have two hours to finish the exam. At 9:15pm, the stage coach turns back into a pumpkin, and the grand outfit transforms back into normal graduate-student clothing. All you will be left with is one lousy crystal slipper. If you don't turn in your exam at 9:15, you will also be left with one very poor grade on this exam.

1. **LFS** consults more data structures when it reads a file from disk than a simpler file system such as **FFS**. Describe the new structures in LFS, and describe why they are necessary.

   Inode map: required to locate where inodes are. Checkpoint region: required to find where the different pieces of an inode are. Once you find an inode, same as FFS. Segments have internal structure too, but most of the details there are related to cleaning, etc.

2. **FFS** uses cylinder groups to place files on disk. **LFS** does not. Describe a workload that will result in nearly identical file and directory placement for FFS and LFS.

   Lots of possibilities here. Basically, one needs to write files such that their temporal locality matches the logical locality that FFS has. By creating files that are in the same directory, one file at a time, LFS can approximate the FFS-style layout.

3. **Nooks** is similar to **Nucleus** in providing more protection boundaries than traditional operating systems. Describe how these two approaches to protection are similar, and describe how they are different.

   Nooks provides separate address spaces for drivers. Nucleus provides separate address spaces for OS services. Both thus use address spaces to isolate code and thus potentially improve reliability.

   They are different in many ways. One way is how Nooks is backwards compatible with existing drivers, and spends a lot of effort to avoid driver rewrite. Nucleus, in contrast, is built from the ground up. Nooks is also more specific, being just for drivers, whereas Nucleus is a more general OS structuring approach.

4. The **Flash** web server uses both threads and events to achieve high performance. **Scheduler activations** just uses threads. Describe all the ways in which a threads-based approach is worse than an event-driven approach.

   Lots of answers here. Threads don't allow you control over scheduling. Threads require synchronization code. Depending on the implementation, threads may not properly integrate with kernel activity (e.g., user-level threads). Etc.

5. **Anticipatory scheduling** suggests that waiting may be the best course of action, rather than immediately scheduling one of the current set of disk requests. **LFS** suggests waiting may be the best course of action, rather than immediately writing data to disk. What are the reasons that waiting in LFS can improve write performance? How is this different than anticipatory scheduling?

   LFS waiting improves performance for two reasons. One, it batches a bunch of small write requests into a single large write request, and thus uses the disk more efficiently. Two, by waiting to write, some writes may be avoided (e.g., when files are deleted soon after creation).

   This is different from anticipatory in that anticipatory waits as a guess; it is hoping that a better request will come along.

6. A **RAID** has an internal scheduler that issues disk requests to the underlying disks of the system. Assume we are using a RAID-1 (mirrored) system. What new scheduling issues arise in mirrored disk systems? (i.e., what makes RAID-level scheduling more challenging than a disk scheduler in a **typical operating system?**)

   Lots of new issues. One major one: on reads, a scheduler must choose which of two copies to read from. Lots of different ways this could be handled, including splitting requests such that only half of each mirrored disk is active (short-stroking). Writes must go to both disks, which is a lesser (and obvious) difference, too.

7. **Lottery scheduling** uses tickets to represent the percent of a resource an application should receive. **Scheduler activations** (in the kernel) has to decide which applications get which processors. Describe how scheduler activations could use a lottery-based approach to perform processor allocation. One question to consider: when should lotteries be held?

   This is kind of a painful question. Basically, one has to use a lottery to split up the number of processors in the system; once this is decided, one could then allocate threads to CPUs. Of course, to match the desired allocation, such a competition must be held frequently. Thus, every so often, a new lottery should be held and CPUs reassigned across processes.

   The problem gets more difficult if a process has lots of tickets but only a few threads to run. In such a case, CPUs could go idle. Thus, some machinery is needed to allocate idle CPUs to processes that need them. Perhaps a solution similar to the idle memory tax in the vmware esx server paper could be used.

8. **Disco** intercepts certain actions that a **normal OS** would handle. One example is a TLB miss, which (in IRIX) is handled by a software TLB miss handler. Describe the flow of control when a TLB miss occurs in an application running on IRIX running on Disco. What optimizations does Disco use to speed up this process?

   TLB miss goes to Disco which then calls up into the OS TLB miss handler. The handler will look up the translation in the page table and try to install it, which will fault back to Disco, which will translate physical-to-machine and install that instead. Disco will give back control to the OS, which will eventually return from trap, which will fault to Disco, which will then return to the user application.

   A software TLB is used by Disco to speed this up. Basically, this is a cache of translations that Disco has seen before, and can install immediately upon a TLB miss without consulting the OS.

9. **Pilot** builds an index over files on disk in order to make efficient file lookup possible. In contrast, most traditional file systems (like **FFS**) view an index as something that must be correct at all times for the file system to work. What mechanism in Pilot enables incorrectness in the index? (describe)

   Pilot uses redundant information to check for correctness. Basically, the index points to a block on disk. Each block, though, has a label, which indicates which logical block it should be. If the index and the label disagree, Pilot takes that as a sign that something is broken and rebuilds the index.

10. **Mesa** provides primitives for concurrency, including monitors for locking and condition variables for signaling and waiting. The **THE** system provides semaphores for locking, signaling, and waiting. Describe how semaphores can be used for these different purposes (examples could be helpful).

    Basically, the answer is in the appendix to the THE paper. Locks are easy to achieve via semaphores; simply set the value of the semaphore to 1 and do a P and V around any critical section. Signalling is easy too, also as described in the appendix.

11. The **IRON file system** uses parity within a disk to protect against partial disk failures. **RAID** systems, in contrast, use parity across disks to protect against disk failures. Describe the *performance* differences of parity protection within a disk versus across disks.

    Parity within a disk raises some new performance issues. If writes are large and sequential, parity is easy to calculate and write out with only the additional I/O of the parity block. However, if a small write occurs, one must perform the read old data, read old parity, compute new parity, write new block, write new parity cycle. This is slow on a RAID, but even slower on a single disk, because each I/O is done in sequence. Thus, both single-disk and multiple-disk systems perform large writes well, whereas small writes slow write latency by a factor of two (RAID) and four (single-disk).

12. In **Nucleus**, processes request services from the OS or other user-level services by sending a message. In a **typical OS** (such as IRIX), service is requested via a system call. What is better about the Nucleus approach? What is worse?

    Message sending tends to be slower than a simple trap. But, message sending is generic and uniform and allows some interesting possibilities. For example, it is easy to extend to make it work across machines that communicate via network. Lots of other possibilities here too.

13. **Scheduler Activations** uses what the **exokernel** would call “visible revocation” when taking away a processor from a running program. Why is this important in scheduler activations?

    It is critical because without this knowledge, SA would not know that it lost a CPU, and thus may not be running the threads it wishes to run given its current set of resources. Thus, control is lost. Could lead to performance problems, etc.

14. **Exokernel** wishes to export a hardware-like interface to operating systems running on top of it. **Disco** (the virtual machine monitor) wishes to export a hardware-like interface to operating systems running on top of it. Describe the crucial difference(s) between these two approaches.

    Quite different. Disco wishes to transparently look like the real hardware, and thus does lots of tricks to fool the OS. But there is a cost: what would be traps to the OS now trap to Disco first and lead to extra overhead.

    Exokernel, in contrast, wants to provide low-level control over hardware, and have applications/OSes fully aware of what they are doing. Thus, exokernel is not virtualizing the hardware as Disco does; rather, it simply makes the hardware primitives and resources available.