Name: _____

UB ID Number: | | | | | | | | |

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	10	5	5	5	5	5	5	20	25	25	100
Score:											

CSE421 Final Exam

07 May 2012

This final exam consists of four types of questions:

1. **Ten multiple choice** questions worth one point each. These are drawn directly from second-half lecture slides and intended to be easy.
2. **Six short answer** questions worth five points each. You can answer as many as you want, but we will give you credit for your best four answers for a total of up to 20 points. You should be able to answer the short answer questions in four or five sentences. These are also drawn from second-half material exclusively.
3. **Two medium answer** questions worth 20 points each, also drawn from second-half material. **Please answer *one*, and only one, medium answer question.** If you answer both, we will only grade one. Your answer to the medium answer should span a page or two.
4. **Three long answer** questions worth 25 points each, integrating material from the entire semester. **Please answer *two*, and only two, long answer questions.** If you answer more, we will only grade two. Your answer to the long answer question should span several pages.

Please answer each question as **clearly** and **succinctly** as possible. Feel free to draw pictures or diagrams if they help you to do so. The point value assigned to each question is intended to suggest how to allocate your time. So you should work on a five point question for roughly five minutes.

No aids of any kind are permitted.

Please fill out your name and UB ID number above. Also write your UB ID number at the bottom of each page of the exam in case the pages become separated.

There are **11** scratch pages at the end of the exam if you need them. If you use them, please clearly indicate which question you are answering.

I have neither given nor received help on this exam.

Sign and Date: _____

Multiple Choice

1. (10 points) Answer all **ten** of the following questions. Each is worth **one** point.

   (a) Which of the following did not happen during lecture this semester?
   ◯ Some students refused to stand-up. ◯ Geoff dropped the f-bomb. ◯ Chuchu wandered out of the lecture hall. ◯ We checked the temperature in Berkeley.

   (b) A head crash occurs when
   ◯ two disk heads collide while scanning across the disk. ◯ a violent movement of a spinning disk causes the heads to make contact with the platter while spinning. ◯ Geoff's head hits his desk after writing this exam. ◯ a disk head crashes into the spindle.

   (c) All of the following are on-disk filesystem data structures *except*
   ◯ inodes. ◯ data blocks. ◯ the superblock. ◯ vnodes.

   (d) Kirk McKusick is
   ◯ the developer of the Berkeley Fast File System (FFS). ◯ Captain Kirk. ◯ the developer of the Log-Structured Filesystem (LFS). ◯ dedicated to using technology to monitor his extensive beer collection.

   (e) Cylinder groups are an idea for improving spinning disk performance introduced by
   ◯ FFS. ◯ LFS. ◯ ZFS. ◯ ReiserFS.

   (f) It is possible to prove the correctness of a microkernel.
   ◯ True. ◯ False.

   (g) All of the following describe ways to structure an operating system kernel *except*
   ◯ multikernel. ◯ zetakernel. ◯ exokernel. ◯ monolithic kernel.

   (h) When your performance data has outliers, you should
   ◯ assume they are experimental error and ignore them. ◯ delete them and never speak of them again. ◯ love and try to understand them. ◯ have never discovered them by exclusively using summary statistics.

   (i) Which of the following is *not* a useful approach to improving system performance?
   ◯ Carefully choosing an appropriate benchmark. ◯ Improving the parts of your code that you just know are slow. ◯ Analyzing data from experiments to identify bottlenecks. ◯ Developing a new simulator to improve reproducibility.

   (j) Which of the following is not one of the requirements of implementing a hardware virtual machine?
   ◯ Fidelity. ◯ Performance. ◯ Atomicity. ◯ Safety.

 UB ID: | | | | | | | | |

Short Answer

Choose **four of the following six** questions to answer. You may choose to answer additional questions, in which case you will receive credit for your best four answers.

2. (5 points) Using appropriate terminology, explain the process of moving from one location to another location on a spinning disk. Identify each major source of latency.

UB ID: | | | | | | | | |

3. (5 points) Describe how filesystem journaling works:

   - What is written to the journal?
   - How is the journal used after a crash to quickly return the filesystem to a consistent state?

UB ID: | | | | | | | | |

4. (5 points) Identify the difference between a *write-through* and *write-back* block-level filesystem buffer cache. Which is better for safety? Which is better for performance?

UB ID: | | | | | | | | |

5. (5 points) List several of the impacts that micro-kernel design and development has had on modern monolithic operating systems.

UB ID: | | | | | | | | |

6. (5 points) Define Amdahl's Law and describe how it guides the process of performance improvement.

UB ID: | | | | | | | | |

7. (5 points) Assuming full hardware virtualization, describe how an application-generated TLB fault is handled by the guest operating system. Assume that this virtual machine architecture supports a software-managed TLB, and that the page causing the fault is already resident in the virtual machine's memory.

UB ID: | | | | | | | | |

Medium Answer

Choose **one of the following two** questions to answer. **Please do not answer both questions.** If you do, we will only read one.

Complete this question on the scratch paper attached to the back of the exam. Clearly label your answer.

8. (20 points) Label your medium answer as **Question 8**.

Virtual Machine `fork()`

You know all there is to know about `fork()`, of course, but your boss at your new software development job just discovered `fork()` and thinks it is just the coolest thing. Unfortunately he also just learned about virtual machines (the second coolest thing), and has hatched a plan to combine the two. Or, to put it accurately, his plan to combine the two is to ask *you* to implement an analog to `fork()` but one that forks *virtual machines* instead of processes.

First, describe virtual machine `fork()` at a high level. Who would implement virtual machine `fork()`? How would they do it? Identify several significant differences between virtual machine `fork()` and process `fork()` arising from the differences between processes and virtual machines. (You may want to think about the rest of the process-related system call interface—`exec()`, `exit()`, and `wait()`—and consider there are virtual machine analogs to these process operations as well.)

Second, explain why this may or may not be a good idea. You may want to consider use cases and overheads.

 UB ID: | | | | | | | | |

Log-structured File Systems on Flash Drives

Flash drives have very different performance characteristics than spinning disks. One important difference is that, because Flash drives contain no moving parts, latencies for access to different locations on disk are roughly constant.

Flash drives do, however, have drawbacks. One concerns how writes are performed. In order to write a byte to a block on a Flash drive you must first erase the entire *erase unit* that contains it, which are usually quite large. Given an erase unit of 16 Kb, the process of modifying a single byte becomes:

- Read the entire 16 kB erase unit into memory in order to preserve the unmodified contents across the erase.
- Change the byte we want to change in our cached copy in memory.
- Erase the erase unit.
- Write the entire modified erase unit back to the Flash drive.

The other important characteristic of Flash drives is that the individual erase units can only be erased a fixed number of times, making it important to try and even out the usage of different portions of the disk.

It seems like it's time for log-structured filesystems to make a comeback, and maybe Flash drives are the place! Let's think this through. Begin by explaining why this may seem like a questionable choice by identifying the key benefit of log-structured filesystems on spinning disks that is lost on Flash. But continue by discussing how log-structured filesystems might help address the two challenges of Flash devices outlined above.

 UB ID: | | | | | | | | |

Long Answer

Choose **two of the following three** questions to answer. **Please do not answer all three questions.** If you do, we will only read the first two.

Complete this question on the scratch paper attached to the back of the exam. Clearly label your answer.

9. (25 points) Label your first long answer as **Question 9**.

10. (25 points) Label your second long answer as **Question 10**.

Application Prefetching

Recall that on systems that use demand paging, page contents are not loaded into memory when the process address space is initialized during `exec()`. Over the lifetime of a process, this procrastination-based technique has significant benefits. In particular, pages containing code that is never executed by the application are never paged in to or swapped out of memory.

One potential downside, however, of demand paging is that when an application first begins executing it generates a large number of page faults to bring the code and libraries it does use in from disk. Application prefetching is a technique supported by several modern operating systems to reduce application startup times on systems with big, slow, spinning disks by *using the past to predict the future*.

First, assuming that the code pages needed by the application as it loads are scattered all over the disk, describe the potentially deleterious consequences of performing these I/O requests in the order the process generates them on a large, spinning disk.

Second, explain why operating systems can accurate predict an application's startup I/O requests. To answer this question, you might consider what is fundamentally different about the first few seconds of an application's execution from other time periods and how that would make its I/O patterns more predictable.

Finally, use these two observations to design application prefetching.

1. Explain what information you need to know about an application to prefetch effectively and how to collect that information.
2. Describe what happens when a prefetched application begins running and how it differs from a non-prefetched application.
3. Present an argument as to why your prefetcher improves application startup performance.

 UB ID: | | | | | | | | |

Asynchronous System Calls

Throughout the semester we have considered system calls as being *synchronous*, or blocking: when a process performs a system call, it is blocked until the call completes. However, modern operating systems also support *asynchronous*, or non-blocking, system calls. These allow a process to request the operating system perform some action on its behalf while not requiring the process wait for the action to take place.

First, considering the system calls we have discussed throughout the semester, describe several cases in which asynchronous system calls would be useful and how. Conversely, describe several synchronous system calls that lack a meaningful asynchronous analog.

Second, describe any changes to the operating system interface that might be necessary to support certain asynchronous system calls. Discuss any additional application programming challenges that non-blocking system calls may introduce.

Third, briefly explain how a process that can fork multiple threads can emulate asynchronous system calls without true non-blocking support from the operating system. Describe the overheads to this approach that might make native asynchronous system calls preferable.

Finally, describe the kernel changes necessary to support asynchronous system calls. Walk through the steps required to complete a non-blocking call, describing what happens at both the process and kernel level.

 UB ID: | | | | | | | | |

Operating System Transactions

Transactions are a database concept and, for the purposes of this problem, can be defined as a set of actions that must be **a**tomic, **c**onsistent, **i**solated and **d**urable. (Together these requirements are known as **ACID** semantics.) We will focus on two of the ACID properties: atomicity and isolation. Atomicity requires that transactions either either *all succeed* or *all fail*. Isolation requires that the changes made by a single transaction not be visible in an intermediate state.

Traditional operating systems interfaces do not provide support for transactions spanning multiple system calls. TxOS, a new research operating system developed at the University of Texas, asserts that adding transaction support will help solve a number of common operating system problems. One common problem is helping keep multiple related files synchronized—adding a user to a UNIX system requires changes to `/etc/passwd`, `/etc/shadow` and `/etc/group`, and certain utilities may fail if they read these files during an update and find inconsistent state: a user without a password during user creation, or a group containing a user that does not exist during user deletion.

First, describe how one of the two cases above can occur. List the system calls in pseudocode for the two processes involved and how they would interleave in time leading to one process viewing inconsistent state. You can assume that both adding and removing a user requires modifications to all three files mentioned.

Second, describe a solution that provides support for transactions consisting of multiple modifications (via `write()`) to one or potentially multiple files. (This is much easier than providing transaction support for the entire filesystem or operating system interface!) Assume that we have added two new system calls `sys_xbegin()` and `sys_xend()` delineating the beginning and end of a single transaction. Your solution should meet the following design requirements ensuring *atomicity* and `isolation`:

1. Any writes to files made by a process after calling `sys_xbegin()` should not be visible until that process calls `sys_xend()`.
2. Any files modified by the transaction can only be modified if they are in the same state when the process calls `sys_xend()` as they were when it called `sys_xbegin()`. You must ensure this by either preventing them from being changed, or by aborting the transaction if you detect that changes have occurred.
3. To achieve good performance you should attempt to keep any locking as fine-grained as possible.
4. It is acceptable, and a property of many good solutions, to fail transactions, but there should be reasonable conditions under which a reasonable transaction can succeed. (Definitions of reasonableness are yours to provide.)

Finally, explain how operating system transactions might be useful when updating system software.

 UB ID: | | | | | | | | |