Name	UBID	Seat	Side

Question:	1	2	3	4	5	6	7	8	9	10	11	12	Total
Points:	10	5	5	5	5	5	5	20	20	25	25	25	100
Score:													

CSE 421/521 Final Exam

12 May 2014

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 (very) 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. The point value assigned to each question is intended to suggest how to allocate your time. **No aids of any kind are permitted.**

Please check your name and UB ID number above. There are 12 scratch pages at the end of the exam. When using 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) The funniest thing that happened *during* class this semester was when
   ◯ Guru fell asleep. ◯ someone thought that one class was actually only five minutes long. ◯ Geoff discussed what to do if you are flying a helicopter and someone points a laser pointer at you[1]. ◯ a PDF rotated unexpectedly.

   (b) Which of the following is *not* a difference between the Xen and VMware approaches to virtualization?
   ◯ Approach to instructions that are not classically virtualizable ◯ Support for the x86 architecture ◯ Ability to run unmodified guest OSes ◯ The way that guest OSes modify their page tables

   (c) It is the *most* difficult to get repeatable performance results when
   ◯ consulting Jinghao's blog. ◯ running a system simulator. ◯ measuring a real system. ◯ using a system model.

   (d) Which of the following is *not* a page replacement algorithm we discussed?
   ◯ FIFO ◯ Least-Recently Used (LRU) ◯ Clock ◯ The Oracle

   (e) All of the following are RAID levels presented by Patterson et. al's paper *except*
   ◯ RAID 0. ◯ RAID 1. ◯ RAID 4. ◯ RAID 5.

   (f) Which of the following is a requirement of implementing a virtual machine?
   ◯ Synchronicity ◯ Performance ◯ Atomicity ◯ Carl Nuessle

   (g) A correctly-implemented journaling filesystem will never lose data.
   ◯ True ◯ Only if Zihe implemented it ◯ False

   (h) Which was one motivation for log-structured filesystems?
   ◯ Disks were getting faster. ◯ Filesystem buffer caches were getting larger. ◯ John Gerber's 80s hairstyle ◯ Filesystem workloads were increasingly dominated by writes.

   (i) Which is *not* one of Butler Lampson's hints for improving system performance?
   ◯ Use hints. ◯ Cache answers. ◯ Compute in the background. ◯ Aggregate load.

   (j) Which is most likely to cause a disk head crash?
   ◯ Scott Haseley's music collection ◯ ASST4 ◯ Dropping a running laptop off the roof of Davis ◯ Your smartphone falling out of your pocket

[1] Just continue flying the helicopter! Simple.

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) Now that you've implemented the OS/161 system calls, let's improve their performance. Might as well start with `getpid()`, since it's simple. **First**, briefly explain the overhead of performing a call to `getpid()` (1 point). **Second**, propose a way to reduce this overhead by exploiting properties of the `getpid()` system call (2 points). **Finally**, will this improve performance? If so, why? If not, why not? Justify your answer (2 points).

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

3. (5 points) Google's File System (GFS) and RAID share a common approach to improving performance. **First**, describe the approach (3 points). **Second**, identify one pro (1 point) and one con (1 point) with this technique.

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

4. (5 points) Describe two ways that filesystems compensate for or are designed around specific properties of spinning disks. For each, **first** identify what the property is (1 point) and **second** explain how the filesystem is designed around it (2 points).

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

5. (5 points) Describe each step in the process of translating the path `/home/trinity/os161` to an inode number in an FFS-like filesystem (1 point each step).

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

6. (5 points) Consider the lock interface used by OS/161. **First**, give an example of improper use of locks that violates the lock interface (2 points). **Second**, explain how good interfaces minimize the disruption required to improve existing code (3 points).

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

7. (5 points) The `POPF` x86 instruction is intended to alter the interrupt flag (`IF`) system flag bit, which controls which interrupts the processor will handle[2]. However, if `POPF` is executed by an unprivileged process it simply fails silently, i.e. it is executed but does not modify the interrupt flag.

   First, describe why `POPF` makes x86 difficult to virtualize (1 point). What property is it missing that would make virtualization easier? **Second**, briefly describe how use of `POPF` by the guest OS would be handled by full hardware virtualization (such as VMware) and by paravirtualization (such as Xen) (2 points each).

[2]So it acts somewhat similarly to the `splx()` command in OS/161, although at the hardware level.

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

Medium Answer

Choose 1 of the following 2 questions to answer. **Do not answer both questions.** If you do, we will only read the shorter one. Complete this question on attached scratch paper. Clearly label your answer.

8. (20 points) Filesystem Versioning

Versioning is an attractive feature for filesystems to provide. It means that not only does the filesystem provide access to the most recent file content through the familiar API, but also uses available disk space to store older version of files and directories as well as deleted files and directories.

Let's consider how to add this feature to the traditional hierarchical filesystem designs we've discussed. Here are design requirements for Our Versioned Filesystem (OVFS):

1. OVFS does not track every change to every file. Instead, it creates periodic *snapshots* that record the state of the filesystem at a particular point in time.
2. OVFS provides access to old file versions without changing the familiar UNIX file API—through `open()`, `close()`, `read()`—with the exception that old file versions cannot be written to, only read from. (Attempts to open file versions with write permissions or perform writes can fail with an error.)
3. OVFS utilizes all available space to store old file versions. Once the disk capacity is reached, it should apply a reasonable policy to remove old versions and create space for new file content.

Present a design for OVFS. **First**, describe changes to filesystem naming required to support #2 without causing undue interference to users' abilities to name files normally (5 points). **Second**, describe how to perform a snapshot to store old file versions (5 points). Keep in mind that other file activity may be taking place, and consider whether you can provide a coherent snapshot of the entire filesystem at an exact moment in time or something more approximate. **Third**, present a default policy for removing old file versions and argue why it is an appropriate default (5 points). **Finally**, describe one performance or storage optimization that you can implement in OVFS to either reduce the disk activity or storage overhead of file versioning compared with the simplest approach (5 points).

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

9. (20 points) Improving `exec()` ELF Loading

Recall that the ELF binary file contains an address space blueprint used by `exec()`. Depending on the process, the executable file can be quite large.

The simplest approach to implementing `exec()` is to have the ELF loader issue writes to load all of the code pages from the ELF executable file. Each page load will generate a VM fault which the OS will handle by adding pages to the code region allowing the content to be copied from the file into the processes address space. The result is that when the process begins running, all of its code pages are already present. OS/161's ELF loader does this by default.

Unfortunately, this violates one of our core system design principles! Let's address the problem[3]. **First**, identify the system design principle that is being violated and describe how (5 points). **Second**, describe a way to modify the ELF load during `exec()` to avoid this problem (10 points). Your solution should *completely* eliminate disk activity during `exec()`, at the price of increased activity as the process begins to execute. Carefully describe what happens during `exec()`, what happens afterward, and any changes to core paging data structures that are required by your approach. **Third**, identify one complication with this approach that could cause processes to crash and describe a way to address it (5 points).

[3] You do not need to consider shared libraries—assume that all code needed by the process is in its own ELF executable, either because it does not use shared libraries or was statically linked.

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 shortest two. Complete these questions on the attached scratch paper. Clearly label your answers.

10. (25 points) OS Implications of Fast, Cheap, Non-Volatile Memory

Since the dawn of computing OS designs have been forced to make price, performance, and capacity tradeoffs when managing memory and secondary storage. Memory is fast but expensive (per byte) and volatile. Spinning disks are cheap (per byte) and non-volatile but slow. Flash is also non-volatile but much slower than memory and more expensive than spinning disks. To a large extent, these tradeoffs have driven the design of modern operating systems.

Now, imagine that you can cheaply provide a device with a terabyte of fast and byte-addressable (like memory) but non-volatile (like disk) storage. This isn't science fiction—the architecture community is exploring the potential of next-generation NVRAM chips that overcome the limitations of Flash. So let's do some dreaming...[4]

Present and motivate *five* different significant aspects of OS design that you would reconsider if you were designing an OS for a device with a single large and fast byte-addressable NVRAM chip replacing both memory and the disk (5 points each)[5].

Five may seem like a lot, but there are dozens of ways that this could revolutionize OS design. Think through the various subsystems that currently manage or use memory and the disk. Think about various OS operations that move state back and forth between memory and the disk. Think about how memory and disk are managed differently and how you could unify management of a single NVRAM chip. Think about process startup and shutdown, installation and update, state maintenance, and the effect of software bugs. Think about reboot. Consider big parts of the OS that may no longer need to exist, but also about side effects of the volatile nature of memory that you may want to preserve on NVRAM systems. Most of all: have fun!

[4] Thanks to Katelin Bailey, Luis Ceze, Steven D. Gribble and Henry M. Levy for inspiring this question.
[5] You can assume that we continue to use processor caches.

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

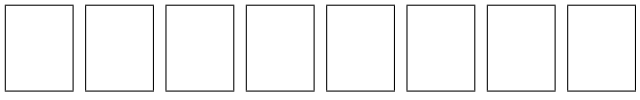
11. (25 points) OS Energy Management

At the beginning of the semester we identified two main OS responsibilities: (1) providing abstractions and (2) multiplexing system resources. We then looked at three case studies of how this is accomplished for three core system resources: the CPU, memory, and the disk. In each case we discussed both how the OS performs and enforces resource allocations, and abstractions it provides to simplify resource usage.

A newer but equally-important resource for operating systems to manage is *energy*. Energy consumption matters at both ends of the computing spectrum, including for data centers for cost and power capping reasons, but particularly on mobile devices. For this question let's focus on smartphones—after all, consumers have been complaining for years about their short battery lifetimes.

First, consider how energy as a resource differs from other system resources we have discussed (10 points). These differences are significant from the perspective of designing effective approaches to managing energy. You might want to use memory management as a point of comparison, since several of the OS requirements for multiplexing memory (grant, enforce, . . .) don't really have appropriate analogs with energy, and several new capabilities are needed.

Second, present a detailed design allowing operating systems to manage process energy consumption on battery-powered smartphones, describing any changes needed at the hardware, OS, and system call interface layers (15 points). Be careful not to make assumptions about OS capabilities regarding energy. Compared with the resources we have discussed, managing energy has some unique prerequisites that complicate the problem. It might be helpful to consider, as a starting point, an OS that knows *nothing* about the energy consumed by processes and go from there. You don't need to provide abstractions unless they are required, but your solution should enable multiplexing. You may want to consider typical usage of smartphones as well as charging patterns as part of your design.

 UB ID: 

12. (25 points) Smartphones, Meet Cloud

Today's computing ecosystem is increasingly dominated by the interaction between two "devices"—smartphones and the cloud—with complementary characteristics. The cloud is always-on, wall-powered, has large amounts of computational power as well as memory and storage capacity, but is located at a datacenter far, far away[6]. In contrast, smartphones are also always-on and always nearby, but have capabilities constrained by multiple factors: energy, form, price, and heat dissipation. It seems likely that smartphones will never be as powerful as the cloud, but neither will the cloud ever be as nearby as your smartphone. It is also much easier to make the cloud faster—just add more machines—whereas making the smartphone faster can be challenging—how do I get eight cores into the same spot where four used to fit, and without lighting a fire in someone's pants?

But what we want to achieve is for our smartphones to serve as a front-end for the cloud, making our smartphone *seem* as powerful as the cloud by intelligently arranging interactions between the two devices. First, make an argument using Amdahl's Law about how future improvements to cloud performance might be lost if cloud resources aren't used intelligently (5 points).

Second, apply two of the systems design principles that we discussed in class to improving the interaction between energy-constrained smartphones and the cloud (10 points each). You can make the following simplifying assumptions:

- The cloud and the smartphone are always connected.
- However, access to local smartphone resources—CPU, memory, and storage—is always much faster than accessing cloud resources due to network delays and limitations[7].
- The cloud can compute much faster and has much more storage available than the smartphone.
- Smartphones are energy-constrained but also regularly recharged.

For each, state the design principle (2 points) and why it is applicable (2 points), and then present a reasonable design sketch of how you propose to apply the principle to smartphone-cloud interactions. Each of your improvements should probably have the effect of either (a) improving user-visible performance, (b) reducing smartphone energy consumption or (c) providing smartphones access to some new capability that requires the cloud's resources.

[6] Not necessarily that far, but farther than your pocket.

[7] At some point the speed of light also starts to play a role.

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