Name	UBID	Seat

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:											

CSE 421/521 Final Exam

09 May 2016

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.

This final exam consists of four types of questions:

1. **Ten multiple choice** questions worth one point each. These are drawn directly from the 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 mostly (but not entirely) drawn from second-half material.
3. **One medium answer** question worth 20 points drawn from second-half material. Your answer to the medium answer should span a page or two.
4. **Two long answer** questions worth 25 points each, integrating material from the entire semester. 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.**

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) One day in class GWA ate

   ◯ something orange. ◯ oranges. ◯ something that Ali videotaped him eating. ◯ a fruit of the citrus variety.

   (b) Significant differences between file systems include everything *except*

   ◯ on-disk layout. ◯ reliably storing Jinghao's data. ◯ data structures. ◯ crash recovery mechanisms.

   (c) What was one approach that Wickizer et. al used to improve Linux scalability to many cores?

   ◯ Running `gmake` ◯ Reducing false sharing in the cache by rearranging in-memory data structures ◯ Rewriting applications ◯ Asking Carl

   (d) Flash drives present none of the layout difficulties typical to spinning disks.

   ◯ True ◯ False

   (e) On `ext4` inodes are *not* stored

   ◯ in groups. ◯ right next to the file contents. ◯ in fixed locations.

   (f) What is a hint that a page might be good to swap out?

   ◯ It hasn't been used for a while ◯ It's currently loaded into a core's TLB ◯ Gela doesn't like it ◯ It's shared by multiple processes

   (g) When your performance data has outliers, you should *not*

   ◯ hide them by exclusively using summary statistics. ◯ inspect them carefully with Xu. ◯ rexamine your mental model of the system. ◯ ensure that they are not due to bugs in your simulator.

   (h) Log-structured file systems provide better performance.

   ◯ True ◯ For some workloads ◯ False

   (i) Which of the following is *not* a useful approach to improving performance?

   ◯ Talking to Yousuf about 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) What is a company known for container virtualization?

   ◯ Virtualbox ◯ Vagrant ◯ Docker ◯ Haseley Inc.

Short Answer

Choose **4 of the following 6** 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) Recall that in RAID level 1 (RAID 1) array, both drives store identical contents. (Assume the drives are spinning disks.) First, explain why you would expect to see a significant performance difference between reads and writes to and from a RAID 1 array (3 points). Second, describe how to coordinate RAID 1 reads to further improve performance (2 points).

3. (5 points) List and explain three of Butler Lampson's hints for improving computer system performance (1 point per hint, 1 point per explanation up to 5 points total).

4. (5 points) We discussed two different formulations of Amdahl's Law:

   > The impact of any effort to improve system performance is constrained by the performance of the parts of the system **not targeted** by the improvement.

   —or—

   > Ignore the thing that **looks** the worst and fix the thing that is **doing the most damage.**

   However, Amdahl's Law also has an important corollary. First, state it (3 points), and then explain how it also guides the process of performance improvement (2 points).

5. (5 points) Explain the difference between placing the buffer cache above the file system interface or below it. What interface must the cache support at each level (2 points)? What is cached (and not cached) in each case (2 points)? Describe one operation with a significant performance difference in each case and explain why this occurs (1 point).

6. (5 points) Describe how file system journaling works:

   - What is written to the journal (2 points)?
   - How is the journal used after a crash to quickly return the file system to a consistent state (3 points)?

7. (5 points) Explain the core cost-benefit tradeoff faced when swapping pages to disk. What is the cost (2 points)? What is the benefit, and how can it differ (2 points)? What is a clever way to reduce the swap-time cost to zero (1 point)?

Medium Answer

8. (20 points) Virtualization Comparison

We discussed three types of virtualization in class: full hardware virtualization, paravirtualization, and OS or container virtualization. First, clearly describe each type of virtualization and give a brief overview of how it works (5 points each). You should explain what is virtualized, define the pieces of software that are involved, explain any constraints that this virtualization places on the virtualized environment, and identify any key challenges to this virtualization approach.

Second, list three virtualization use cases that motivate each of the three approaches (2 points each, up to 5 points total). For each of your examples you should make a convincing case that the other virtualization approaches are impossible, ineffective, or perform poorly.

Long Answer

9. (25 points) Supporting Heterogeneous Cores

An increasing number of computer systems—from smartphones to servers—now feature multiple *heterogeneous cores* with different power-performance tradeoffs. For example, ARM now has many systems that use their `big.LITTLE` architecture which combines one or more fast and high-power cores with one or more slower but lower-power cores. (While this is not important to answering the question, the reason for this approach is that how the core is made has an impact on how much power it consumes—even in cases where the energy-performance tradeoff can be adjusted at runtime by changing the operating frequency.)

Operating systems with heterogeneous cores creates new challenges and opportunities for OS design. In this question we ask you to consider a two-core heterogeneous system with the following properties:

1. **Performance and energy consumption.** Both cores can scale their speed and power consumption up and down at runtime, but the big core is always faster and always consumes more energy than the small core. Put another way, even when the big core is running at its *slowest* frequency it is almost equivalent to the small core running at its *fastest* frequency.
2. **Instruction Set Compatibility.** All of the user-mode instructions that can be run on the small core can be run on the big core (more quickly), but all of the user-mode instructions that can be run on the big core *cannot* be run on the small core. However, those instructions will generate an exception if they are run on the small core. (You can assume all kernel-mode instructions work identically on both cores.)
3. **Memory Access.** The big core has access to all of the hardware memory, but the small core can only see the first 512 MB of available hardware memory. Unlike the instruction set case above, attempts to access invalid hardware memory address on the small core will result in fatal exceptions that cause both processors to reset, so they must be avoided by the OS.

Describe how to design an OS that allows multithreaded application processes to effectively and efficiently use both big and small cores. First, explain two of the OS design challenges that this architecture creates compared with a typical homogeneous multicore share memory system (5 points each). Second, describe how to address each challenge through changes to the OS but *without* changing compiled applications (5 points each). Finally, describe changes to applications themselves or the OS API that might help improve performance or energy efficiency on this heterogeneous system (5 points).

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. . .[1]

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)[2].

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!

[1] Thanks to Katelin Bailey, Luis Ceze, Steven D. Gribble and Henry M. Levy for inspiring this question.

[2] You can assume that we continue to use processor caches.