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

06 May 2013

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?

   ○ We were temporarily locked out of Davis 101. ○ All three Tims attended. ○ Geoff advertised a woodshop tool. ○ We investigated a wine collection in Berkeley.

   (b) What could cause a head crash?

   ○ Throwing a Flash drive against the wall. ○ Tapping your fingers on your laptop keyboard. ○ Your desktop tipping over while running. ○ Dropping your laptop while it is off.

   (c) Many operating system crashes are caused by

   ○ device drivers. ○ page translation. ○ filesystems. ○ applications.

   (d) Which of the following is a useful approach to improving system performance?

   ○ Choosing a benchmark randomly. ○ Improving the parts of your code that you just know are slow. ○ Analyzing data from experiments to identify bottlenecks. ○ Working on the slowest part first.

   (e) When your performance data has outliers, you should

   ○ eliminate them. ○ assume they are experimental error. ○ understand them. ○ use a summary statistic.

   (f) Modern filesystems are nothing like the Berkeley Fast File System (FFS).

   ○ True. ○ False.

   (g) A virtual address might point to all of the following *except*

   ○ physical memory. ○ a disk block. ○ a port on a hardware device. ○ a register on the CPU.

   (h) Address translation allows the kernel to provide all of the following *except*

   ○ the address space abstraction. ○ process isolation. ○ an inter-process communication mechanism. ○ direct access to hardware.

   (i) Significant differences between filesystems include everything *except*

   ○ on-disk layout. ○ reliable data storage. ○ data structures. ○ crash recovery mechanisms.

   (j) Which of the following is *not* a reason that virtualization became popular?

   ○ Difficulty migrating software setups from one machine to another. ○ Useful hardware virtualization features. ○ Ability to reprovision hardware resources as needed. ○ Lack of true application isolation provided by traditional operating systems.

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) First, using appropriate terminology, describe the process of swapping out a page. For each step, explain *why* it must be performed. Second, point out how the operating system can reduce the overhead of this operation by planning ahead.

3. (5 points) List all the steps that are taken by a traditional hierarchical filesystem when creating a new file. Assume that the file is not empty.

4. (5 points) Identify two tradeoffs inherent in operating system memory management. A tradeoff should involve a balance between providing two desirable properties. For each,

   - briefly describe cases in which the operating system would want to resolve the tradeoff in favor of one of the desirable properties, and
   - explain how balance is achieved during normal operation.

UB ID: □ □ □ □ □ □ □ □

5. (5 points) Describe the difference between software- and hardware-managed TLBs and how they affect kernel TLB and page fault handling. Briefly describe the pros and cons of each approach.

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

7. (5 points) First, identify one way an instruction might fail to be “classically” virtualizable. Second, explain how this prevents the Virtual Machine Monitor from performing trap-and-emulate virtualization. Finally, briefly describe how modern virtual machine monitors work around this problem to perform full hardware virtualization.

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**.

Identifying Identical Pages

We discussed in class how operating systems use a technique called copy-on-write to allow identical virtual pages to be shared between the parent and child process after `fork()`. Copy-on-write relies on the observation that `fork()` makes an *identical* copy of the parent's address space. So afterward, we know that for two virtual addresses in the parent and child, VA_{parent} and VA_{child}, if $VA_{parent} = VA_{child}$ at the time `fork()` is called then the pages have identical contents. Marking the shared page ready-only ensures that any modifications by either process cause a page fault allowing the kernel to create private copies of the now not-identical page.

While copy-on-write is a clever mechanism to use after `fork()`, it will not identify all possible page sharing opportunities. First, explain why not. Second, describe a system for identifying page-sharing opportunities missed by copy-on-write. Please be specific about how your system works: how it identifies identical pages, how they are merged, and how it ensures that pages that should be private stay private.

Finally, this approach emerged as a response to the increase use of virtualization technologies. Briefly explain why this is. Can you think of any other devices that would benefit from your approach?

Controlling Virtual Machine Memory Usage

Virtual machine monitors typically require a substantial amount of memory in order to create a virtual machine for the guest operating system. However, when running one or multiple VMMs the host operating system may want or need to reclaim memory from them. The process of reducing guest operating system memory usage is complicated by the fact that the guest OS *thinks* that it has access to a entire large chunk of memory, the amount presented to it by the virtual machine. But in reality the host OS will want to reclaim memory from the guest OS.

In most cases the host operating system *can* directly swap out pages used by the virtual machine monitor—an approach called hypervisor (another name for VMM) swapping. However, hypervisor swapping can cause several potential problems. First, explain the two following problems in more detail, including a description of why the would occur:

- The host OS may choose the wrong page.
- Under certain conditions pages may actually be swapped in and out *twice*. (As a hint, consider what happens if the host OS has swapped out a page that is then swapped out by the guest OS.)

Second, describe a solution that allows the guest and host OS to collaborate to reduce guest OS memory usage. Keep in mind that the guest OS does not and *should not* know that it is running inside a virtual machine. Instead, you should design a way for the host OS to create memory pressure inside the guest OS and cause it to begin to evict pages itself. Describe why this is a superior approach by discussing how it addresses the weaknesses you identified 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**.

Incorporating FPGAs into Kernel Designs

Field-programmable gate arrays (FPGAs) are a form of reprogrammable hardware that provide some of the flexibility of software (reprogrammability) and some of the speed of hardware, with performance much better than general-purpose processors but worse than specialized application-specific integrated circuits (ASICs). For the purposes of this question we won't get into the details of how to program FPGAs, but you should keep in mind that the speedup achieved by moving code from a general-purpose processor to an FPGA may vary based on what computations are being done.

Describe how to modify an existing operating system to effectively utilize an FPGA incorporated into a system design. Your goal should be to harness the computational horsepower of the FPGA to significantly improve application performance, while effectively multiplexing this new hardware resource. Your system should be able to decide *who* can use the FPGA but also help determine *what* they should be doing with it. There are some design tradeoffs here that you should point out and decide how to balance. You may also want to consider modifications to the executable-and-linking format (ELF) binaries we have discussed earlier this semester to support FPGA incorporation.

You can make the following assumptions:

- The FPGA cannot effectively be space-multiplexed, i.e. it can be programmed to do only one thing at a time.
- Reprogramming the FPGA takes time.
- Once the FPGA is reprogrammed, essentially a new instruction exists that can be used by any thread running on the CPU to activate the FPGA to perform a custom calculation (whatever it is programmed to do) based on inputs that could be drawn from memory or processor registers. This also implies that applications *do not* have to make a system call or involve the kernel to use the FPGA.
- The kernel has access to a program that can transform any binary code written for the native instruction-set architecture (ISA) into code to run on the FPGA, but this process has some overhead to it.

Deterministic Shared Memory Parallelism

(Credit for this question goes to Bryan Ford at Yale and his Determinator system.)

Earlier this semester we presented synchronization primitives as a solution to *race conditions*, which we defined as cases where the ordering and timing of thread execution would cause a result to be incorrect or unpredictable. (Remember the bank account example?) Reasoning about how threads might interleave at runtime and the pattern of their accesses to shared memory has made multi-threaded programming difficult for years.

Part of the problem is that, when accessing shared memory, reads and writes from each thread happen synchronously, meaning that if synchronization primitives are not used correctly threads can observe variables in an inconsistent state. Consider the following game-based scenario. During each time step, each actor in the game performs some action and updates its position based on the position of all other actors. The game uses a separate thread to perform the computation for each actor, but all threads read from and write to a state table stored in shared memory. However, the position of an actor is defined as a tuple in three-dimensional space, (x, y, z), and so requires three memory writes to change and three memory reads to access, and so absent proper synchronization is *not* atomic.

The next page contains pseudo-code for the game described above. `thread_fork` you are familiar with: it creates a new thread, in this case a new *user thread*. The `thread_join` command will wait for the thread to complete before returning. It's analogous to `waitpid` but for threads, rather than processes.

First, describe how a race condition can occur if the table is not properly synchronized. You should identify a condition where one thread will observe a shared variable in an inconsistent state. Briefly, outline a solution to this problem that uses one of the synchronization primitives we discussed in class. (Clearly, this is not the *real* question.)

Second, obviously the world would be a better place if every programmer used synchronization primitives correctly. But the world is not that place, sadly, and so operating systems may want to aid programmers by providing primitives that improve safety during multithreading. Propose a solution to the race condition you identified above that *does not* modify the application code above. Instead, you should confine your modifications to `thread_fork` and `thread_join`. As a hint, you might want to consider the design and operation of the Git version control system that you have been using throughout the semester.

```

void update_actor(int i) {
  // ... examine state of other actors ...
  state[i].x = new_x
  state[i].y = new_y
  state[i].z = new_z
}

int main() {
  // ... initialize state of all actors ...
  for (int time = 0; ; time++) {
    thread t[NACTORS];
    for (i = 0; i < NACTORS; i++) {
      t[i] = thread_fork(update_actor, i);
    }
    for (i = 0; i < NACTORS; i++) {
      thread_join(t[i]);
    }
  }
}
```

Characteristic	SSD	HDD
Price-per-gigabyte	$0.59	$0.05 (See note 1.)
Random access time	100 μs	10 ms
Data transfer rate	100 MB/s	100 MB/s (See note 2.)
Durability	No moving parts, resistant to shock and vibration	Sensitive to shock and vibration

[1] For the 3.5 in form factor.

[2] Once the head is positioned.

Table 1: Comparison of HDD and SSD disk drives.

Hybrid Disk Drives

Today disks are at a crossroads. Magnetic spinning drives still offer a order-of-magnitude improvement in price-per-byte, but Flash technologies are improving latencies and bandwidth. Specifically, you can make the following assumptions about magnetic hard-disk drives (HDDs) and Flash-based solid-state drives (SSDs) listed in Table 1.

Your new company, HyDrive ®, thinks it can provide the best of both worlds by shipping a hybrid hard drive *and* new filesystem. Your hybrid drive contains both a large(r) HDD and a small(er) SSD, each of which is block-addressed and independently controlled; consider them two independent disks that are sold inside the same package. Unfortunately, *everyone*—HyDisk ®, DriveBrid ®, DiskBrid ®, Grapes ®—is cramming two disks into one box and pretending that it's innovative. What sets HyDrive apart is the innovative new filesystem you are selling that takes full advantage of the capabilities of both the HDD and SSD.

Describe the design of the breakthrough HyDrive ®filesystem. You can use any of the ideas presented in class, but should aim to achieve the best of both worlds and have a single filesystem spanning both disks, with seek times as fast (or close to) as the SSD, capacity as large as (or close to) the HDD and SSD combined, and better durability and failure recovery than an HDD alone. Be very specific about the design and implementation of your new filesystem. In particular, it is necessary (but not sufficient) to address the following questions:

- What file metadata is required to name files and locate their contents?
- Where is file metadata stored?
- Where is file data stored?
- How does your filesystem determine where to put file metadata and file data?
- How does your filesystem recover after an unplanned power outage or reboot? How do you recover from failures on the HDD? What about failures on the SSD?
- How does your design harness the best features of the SSD? What about the benefits of the HDD?

Keep in mind that because you are implementing a *filesystem*, rather than a disk controller, you have visibility into file operations such as reads and writes and don't have to confine yourself to observing block-level operations.

UB ID: □□□□□□□□

Scratch. **Please indicate what question you are answering.**

Scratch. **Please indicate what question you are answering.**

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**

Scratch. **Please indicate what question you are answering.**

UB ID: □ □ □ □ □ □ □ □

Scratch. **Please indicate what question you are answering.**