The University of Texas at Austin
CS 395T Operating Systems Implementation: Fall 2010
Midterm Exam

- This exam is **75 minutes**. Stop writing when "time" is called. *You must turn in your exam; we will not collect them.* Do not get up or pack up between 70 and 75 minutes. The instructor will leave the room 78 minutes after the exam begins and will not accept exams outside the room.
- There are **14** questions in this booklet. Many can be answered quickly. Some may be harder than others, and some earn more points than others. You may want to skim all questions before starting.
- **This exam is closed notes. You may not use electronics: phones, PDAs, calculators, etc.** You may use your copies of the papers—but only if they do not have class notes on them.
- If you find a question unclear or ambiguous, be sure to write any assumptions you make.
- Follow the instructions: if they ask you to justify something, explain your reasoning and any important assumptions. **Write brief, precise answers. Rambling brain dumps will not work and will waste time.** Think before you start writing so you can answer crisply. Be neat. If we can't understand your answer, we can't give you credit!
- To discourage guessing and brain dumps, we will give 25%-33% of the credit for any problem left *completely blank* (for example, 1 point for a 3 point question). If you attempt a problem, you start at zero points for the problem. Note that by *problem* we mean numbered questions for which a point total is listed. *Sub-problems* with no points listed are not eligible for this treatment. Thus, if you attempt any sub-problem, you may as well attempt the other sub-problems in the problem. The exception is the True/False problems. There, we grade by individual True/False item: correct items earn positive points, blank items earn 0 points, and incorrect items earn negative points. However, the minimum score on any problem—that is, any group of True/False items—is 0, and leaving an entire T/F problem (that is, all items) blank gets you 25%-33% of the credit for the entire problem.
- Don't overthink all of this. What you should do is: if you *think* you *might* know the answer, then answer the problem. If you *know* you *don't* know the answer, then leave it blank.
- Don't linger. If you know the answer, give it, and move on.
- **Write your name and UT EID on this cover sheet and on the bottom of every page of the exam.**

Do not write in the boxes below.

Name (4)	I (xx/10)	II (xx/36)	III (xx/16)	IV (xx/24)	V (xx/10)	Total (xx/100)

Name (4 points): **UT EID:**

I Unix (10 points total)

1. [4 points] In the original Unix, as described in the papers that we read, hard links to directories were not allowed (besides the initial hard link that created the directory). In other words, each directory was allowed to have only one parent. This question asks about the motivation for this design decision.

What could go wrong if directories could have multiple parents?

2. [6 points] Recall that in Unix, the `fork()` system call creates a nearly-exact duplicate of the process that called `fork()`. Further recall that the `exec(filename, arg1, ...)` system call (more accurately, the variants like `execl()`, `execve()`, etc.) replaces the currently-running image with a new process image (specifically, `filename` is executed). The question we are asking here is: what is the power of having `fork()` and `exec()` be separate calls? The comparison is to a call like `spawn(filename, arg1, ...)` that simply creates a new running process by executing `filename`.

Explain *briefly* the advantage of separating `fork()` and `exec()` versus having a single call like `spawn()`.

II Virtual memory and virtualization (36 points total)

3. [8 points] In the paper "Virtual Memory Primitives for User Programs", Appel and Li quote another paper (Mike O'Dell, "Putting UNIX on very fast computers", Proc. Summer 1990 USENIX Conference):

> Modern Unix systems . . . let user programs actively participate in memory management functions by allowing them to explicitly manipulate their memory mappings. This . . . serves as the courier of an engraved invitation to Hell

Explain *briefly* what the authors of this paper take this quotation to mean.

Explain *briefly* why the authors say that the invitation to hell can be "returned to sender". (Hint: a proper answer to this sub-question is in two parts.)

4. [6 points] State two reasons why paravirtualization (as practiced by Xen) performs so much better in execution time than full virtualization (as practiced by some of Xen's competitors).

Be concise.

5. [6 points] Consider the x86 virtualization extensions that are measured in "A comparison of software and hardware techniques for x86 virtualization". Assume that the load on a given VMM is such that there is no need to page to the disk and likewise that the load on the guest VMs is such that there is no need to page to the virtual disk.

Circle True or False for each item below, and note that the definitions below the items may be helpful:

True / False These extensions provide absolutely no help to the authors in reducing *tracing faults*.

True / False These extensions provide absolutely no help to the authors in reducing *hidden page faults*.

True / False These extensions provide absolutely no help to the authors in reducing *true page faults*.

Relevant definitions:

Tracing fault: A tracing fault occurs when (1) the guest OS attempts to modify the page tables that it sees while (2) the virtual address through which it access those page tables is, unbeknownst to that guest OS, not mapped in the shadow page tables that are currently pointed to from the real processor's `%cr3`.

Hidden page fault: A hidden page fault occurs when (1) the guest virtual machine attempts to gain access to a given virtual memory address while (2) that virtual address is, unbeknownst to that guest OS, not mapped in the shadow page tables that are currently pointed to from the real processor's `%cr3`.

True page fault: A true page fault occurs when a guest process attempts to gain access to a virtual memory address that is not mapped in the page tables pointed to from the *virtual* `%cr3`.

6. [8 points] This question also concerns "A comparison of software and hardware techniques for x86 virtualization".

(a) Describe qualitatively (i.e., do not state the name of a benchmark) a type of workload for which the authors show that *hardware*-based x86 virtualization will perform better. (b) Explain *briefly* why the hardware-based techniques perform better on this workload.

(a) Describe qualitatively (i.e., do not state the name of a benchmark) a type of workload for which the authors show that *software*-based x86 virtualization will perform better. (b) Explain *briefly* why the software-based techniques perform better on this workload.

7. [8 points] This question concerns "Memory Resource Management in VMware ESX Server". Recall that, in this paper, the VMM estimates the fraction of physical memory that each virtual machine (VM) is actually using, and recall that the VMM does so by statistical sampling, as explained in the third paragraph of section 5.3. The VMM uses this statistical sample as an input into which of the following decisions and calculations?

Circle all that apply:

A The choice of VM to reclaim a page from when using the ballooning technique.

B The choice of VM to reclaim a page from when using random paging.

C The choice of VM to inspect for possible content-based page matches.

D The choice of VMs that are subject to the "innocent until proven guilty" binary translation policy.

E How much to dock a virtual machine's shares, S, if the virtual machine hoards idle memory.

F The calculation of a virtual machine's adjusted shares-per-page ratio.

III Kernel structure (16 points total)

8. [8 points] This question concerns "Improving IPC by Kernel Design" and concentrates on Liedtke's Lazy Scheduling technique.

Circle True or False for the two items below:

True / False The gain from Lazy Scheduling, as a percentage of IPC time, depends heavily on the size of IPC messages.

True / False Given Lazy Scheduling, the scheduler, when making a scheduling decision, is forced to scan all queues because threads could be on inappropriate queues.

Beginning at time t_1, the following sequence happens: a thread A unblocks because a message is available, then processes the message, then replies to it, thereby blocking again for some time, then unblocks because another message is available, then processes that message, then replies to it, thereby blocking again. This last blocking begins at time t_2.

Circle True or False for the two items below:

True / False It is possible that, over the interval $[t_1, t_2]$, A is never on the ready queue.

True / False It is possible that, over the interval $[t_1, t_2]$, A is always on the ready queue.

9. [8 points] Recall that in JOS, user-level environments have access to their own page tables, through the virtual addresses $[\texttt{UVPT}, \texttt{UVPT} + 4\text{MB})$. However, this access is read-only.

If an environment were permitted to write to its page tables, how could that environment harm the system or other environments?

In Aegis, as described in "Exokernel: An Operating System Architecture for Application-Level Resource Management", an application (which is analogous to a JOS environment) has both read *and* write access to its page tables.

Why is it safe for applications to write to their page tables in Aegis whereas it is not safe in JOS?

IV Concurrency and scheduling (24 points total)

10. [8 points] This question concerns monitors and condition variables, as described in "Experiences with Processes and Monitors in Mesa" and refined by "Programming with threads", by Mike Dahlin. If you believe that the Mesa paper and Mike Dahlin's document conflict, follow Mike Dahlin's document; deviating from the standards there is considered incorrect. Recall that programmers who use condition variables are given a set of primitives:

```
Cond::wait();
Cond::signal(); // called "notify()" in Mesa
Cond::broadcast();
```

Assume that `Cond::wait()` does not take a timeout argument and that there is no facility for aborting a process or thread that is `wait()`ing. Define the *programmer* to be the user of condition variables, threads, processes, locks, etc. Define the *implementation* to be the code that implements condition variables, threads, processes, locks, etc.

Circle True or False for each item below:

True / False The Mesa scheduler is preemptive.

True / False There are cases when it is correct for the programmer to `signal()` a condition variable without holding the monitor lock.

True / False Assume that several threads are inside `wait()`. If no `signal()` or `broadcast()` has taken place, but the implementation returns one of the threads from the `wait()` call, that is a bug.

True / False Assume that several threads are inside `wait()`. If a `signal()` takes place, but the implementation ignores it (by not placing any thread in the runnable state), that is a bug.

11. [8 points] This question concerns "Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler". Assume that the operating system implements a single-level BVT scheduler and that each thread has a single warp value set at thread creation time. Further assume that the operating system provides monitors. *There are four True/False items in this problem.*

Circle True or False for the three items below, and note that the definitions at the end of the question may be helpful:

True / False Under BVT, starvation is possible.

True / False Under BVT, priority inversion is possible.

True / False Under BVT, deadlock is possible.

Now consider the following scenario:

At time t_a microseconds (μs), a thread T_1 is given the processor. At time $t_b = t_a + 1\mu$s, a thread T_2 becomes runnable. The context switch allowance, C, is 20 milliseconds, and *mcu*, the minimum charging unit, is 5 milliseconds. Both threads have the same weight.

Circle True or False for the item below:

True / False If, at time t_b, the effective virtual time of T_2 is smaller than the effective virtual time of T_1, then the scheduler stops running thread T_1 at t_b.

Definitions

Starvation: One thread permanently preventing another from executing.

Priority inversion: A medium priority thread (where priority is defined below) prevents a high priority thread from executing, when neither is waiting for I/O. This can happen if the high priority thread is waiting for a resource held by a low priority thread. The Mesa paper gives an example of priority inversion.

Priority: In BVT, what it means for a thread T_i to have higher priority than a thread T_j is that if both T_i and T_j are runnable, and if neither has run in some time, then T_i has lower dispatch latency, that is, the scheduler will choose T_i in preference to T_j. This preference is expressed by giving T_i a higher warp value than T_j.

12. [8 points] This problem concerns Determinator, as described in "Efficient System-Enforced Deterministic Parallelism". Assume that Determinator is running on a single processor machine.

Below, note that instructions and questions are interleaved; please read carefully.

If two or more Determinator *spaces* (as defined in section 3) are runnable (that is, not blocked inside `Put`, `Get`, or `Ret`), some module in Determinator must assign a space to the processor for some length of time. Call this module the *space scheduler*.

Circle True or False for both of the items below:

True / False When choosing a space to run next, Determinator's space scheduler is permitted to choose randomly. (By *random*, we mean that the choice is based on a source of external randomness, such as nuclear decay as detected by a Geiger counter.)

True / False When choosing the maximum length of time that the chosen space will be allowed to run, Determinator's space scheduler is permitted to choose randomly. (By *maximum length of time*, we mean the length of time after which the space will be preempted, if it has not yielded or blocked.)

Now consider *processes* (section 4.1), which are built on top of Determinator's spaces. Each process runs on a single space. Assume that, for synchronization, the processes use only deterministic synchronization primitives (for instance, `fork()`, a deterministic version of `wait()`, and barriers—but not mutexes).

Circle True or False below:

True / False The interleaving of processes on the processor is not permitted to depend on an external source of randomness. (By *interleaving*, we mean a conceptual schedule: one can imagine listing which process gets the processor and for how many instructions, then which process next gets the processor and for how many instructions, etc. This conceptual list is the interleaving, or the schedule.)

Now imagine that a single process *P* uses Determinator's runtime (which itself uses spaces) to instantiate several *threads* (section 4.5). Assume that the threads use nondeterministic synchronization primitives (mutexes, condition variables, etc.).

Circle True or False below:

True / False The interleaving of *P*'s threads is nondeterministic. (By *nondeterministic*, we mean that if *P* is run multiple times with the same input, the interleavings might be different.)

V JOS and feedback (10 points total)

13. [8 points] Recall that the JOS kernel runs non-preemptively. That is, when kernel-level code is executing, interrupts are *never* enabled. This guarantee ensures that the kernel will never itself be interrupted by, say, a device receiving data. On the other hand, when user-level code is executing, interrupts are *always* enabled. This guarantee ensures that the kernel can regain control from user-level processes (as is necessary on timer interrupts, device I/O, illegal instructions from a user-level process, page faults in a user-level process, etc.)

What code did you write and/or what data structures did you set up to ensure that interrupts are *never* enabled in kernel mode? Be specific and concise.

What code did you write and/or what data structures did you set up to ensure that interrupts are *always* enabled in user mode? Be specific and concise.

14. [2 points] This is to gather feedback. For both of the questions below, any answer will receive full credit. *A blank answer will earn zero points.*

Please list the two papers or classes in this course that you have most enjoyed:

Please list the two papers or classes in this course that you have least enjoyed:

End of Midterm

Name: **UT EID:**