Slide 1

Administrivia

- I've listed reading for today, but really it's optional (though interesting!).
- If you're behind on homework: I'll still accept all homeworks (for reduced credit unless you have an extracurricular reason for being late — in which case remind me when you turn something in), as long as you haven't looked at a solution. However, there will be a "not accepted past" deadline during finals week. Specifics TBA by e-mail soon.

Slide 2

Minute Essay From Last Lecture

- Pretty much everyone agreed that people in general seem to prefer features to security (sometimes — though not always — including themselves).
- Several people thought most computer users aren't even aware of the possible risks.
- "I think people tend to think that they are smart enough that they don't need to worry about security." (!)
- Heard on the phone with a support person a while back: With regard to identity theft, there are two categories of people: those who have been victims, and those who haven't — yet. Sadly probably true!

Slide 3

Homework 6 Essays

- Most people found the problems relatively easy, though not all. A few thought they were hard, and several found the last problem especially hard. (I thought it was a not-too-tough extension of the previous problem, but apparently several of you didn't agree!)
- Several people mentioned finding the calculations and/or results interesting.
- A few people mentioned finding the question about permissions interesting.

Slide 4

Virtualization — Overview

- Process abstraction allows multiple applications to run concurrently on the same hardware, isolating them from each other to some extent.
- Virtualization takes that a step(?) further and allows multiple operating systems to run concurrently on the same hardware, isolating them from each other.
- Attractive for many reasons, most of them related to usefulness of seeming to have more computers than you actually have (e.g., run many different operating systems without having many computers).
- Idea is far from new — goes back to the mainframe days (VM/370 mentioned early in the semester) — but has recently become very popular, especially for providing computing services "in the cloud". Trinity's ITS also seems to be moving toward use of virtual machines for servers.

Slide 5

Virtualization — Overview, Continued

- To make this work, "real" operating system is a "hypervisor" that essentially simulates multiple "virtual machines" on a single physical machine.
- Goals:
  - Safety: Hypervisor controls all resources.
  - Fidelity: Programs run the same on virtual machine as on physical machine.
  - Efficiency: Most code should run without interference/help from hypervisor.
- Could meet first two of these with "interpreter" that basically emulates instruction of actual hardware. But performance would be dismal.

Slide 6

Approaches

- Type 1 hypervisor runs on actual hardware and provides virtual machines, each running a guest O/S.
- Type 2 hypervisor runs as an application in host O/S, provides virtual machines to guest O/S's.
- (Figure 7-1 in textbook.)

Slide 7

Processor Virtualization

- Key problem is that real O/S's have full access to all hardware resources. How then can a hypervisor be in control?
- Obvious(?) solution is to have each guest O/S run as application program. But then what happens when it executes a privileged instruction? interrupt is generated, control goes to interrupt handler in hypervisor, it decides what to do.

  (Figure 7.3 in textbook.)
- This works well for some architectures (ones that are "virtualizable"), but not all are — in particular, x86 and many successors.
- Non-virtualizable architectures have instructions that are not privileged (i.e., can be executed in user mode without causing interrupt) but behave differently in user and kernel modes (e.g., they do nothing in user mode). What to do?

Slide 8

"Virtualizing the Unvirtualizable"

- One approach: "Binary translation", in which guest O/S is translated during execution in a way that replaces so-called "sensitive" instructions (ones that behave differently in user and kernel modes) with calls to hypervisor. Sounds very inefficient, but in practice apparently works well. More details in textbook.
- Another approach: "Paravirtualization", in which guest O/S is revised so that any operations normally requiring kernel mode are replaced with "hypervisor calls" (analogous to "supervisor calls" in application programs). Of course(?), then you have an O/S that no longer runs on actual hardware.

Slide 9

"But Wait, There's More"

- Key problem in virtualization is that O/S running on actual hardware is in charge of all physical resources, but if it's to run as a guest O/S under hypervisor, it can't be.
- Discussion so far has focused on processor instructions. But there are similar problems related to both memory and I/O — guest O/S thinks it's "king of the mountain", but it can't be.

Slide 10

Memory Virtualization

- In O/S supporting virtual memory, recall distinction between program addresses and physical addresses. Translation is done by hardware (MMU) with help from O/S (e.g., maintaining page tables).

  Clearly(?) a problem if each guest O/S thinks it has access to full memory.
- Software-only approach to virtualizing this (assuming paging) involves "shadow page tables". Keeping these in synch with guest-O/S page tables is a challenge. More than one approach possible, all workable but messy and/or inefficient.
- Hardware-based approach based on defining "guest virtual address", "guest physical address", and actual physical address, with hardware support for two translations.

  (Figure 7-7 in textbook.)
- More details in textbook.

Slide 11

I/O Virtualization

- At first thought, this seems more manageable: I/O-related operations in guest O/S involves access to privileged instructions or (real) addresses, so will generate exceptions, which the hypervisor will handle.
- How? usual approach is for hypervisor to define "virtual disk" that to it looks like a regular file. Note that this allows emulating I/O hardware that doesn't exist on physical system.
- But what about DMA? an approach that works well for that and provides additional functionality is "I/O MMU" that translates addresses, including ones used for memory-mapped I/O (I think!).
- More details in textbook.

Slide 12

Minute Essay

- Have you used virtualization? (VMware, VirtualBox, cloud-based virtual machines, . . .)