Slide 1

Administrivia

- Reminder: Homework 1 due today (written part 5pm, programming problem 11:59pm).

  If you don't finish the programming problem by the deadline, *turn in what you have*, and optionally turn in an improved version later. As long as you turn in something on time, the late penalty will be reduced or even waived.
- (What to do about office hours? after class there's an ACM meeting.)

Slide 2

Minute Essay From Last Lecture

- Many people seemed to agree with me about the user interface not being the same as the operating system — but also mentioned that non-technical people don't make this distinction. (True!)
- Several people mentioned changes to what we'd call the o/s — switch from 32-bit(?) to 64-bit, increase in how many applications can run concurrently, support for new/different hardware.
- (My thinking: The GUI can be distinct from the O/S, as it is in UNIXworld. So there could be changes in one but not the other. Then again, considering the O/S as a "virtual machine", maybe it *should* include the GUI?)

Slide 3

Operating System Structures — Recap/Review

- General-purpose operating systems are big — tens of millions of lines of code (probably mostly in something C-like). How to organize all of it? several choices, discussed in textbook.
- A possibly-relevant maxim, origin unknown (to me): “Any programming problem can be solved by adding a layer of abstraction. Any performance problem can be solved by removing a layer of abstraction.” Not always true, but true enough?
- One possible structure seems worth a bit more discussion — “virtual machine”.

Slide 4

Virtual Machines

- Idea — o/s provides a simulation of the actual physical machine, this “virtual machine” then runs another o/s – or several of them.
- Examples include VM/370, Windows support for old MS-DOS programs, VMware, Java Virtual Machine, other virtualization schemes.
- (Notice how this is an idea that fell out of favor for a while, then came back.)

Slide 5

Virtual Machines, Continued

- Arguments for — separates multiprogramming from other concerns, emulation aspect can be useful, useful in o/s development.
- Arguments against — another layer, so can be slower. Also, may be difficult for some hardware — e.g., if privileged instructions executed in user mode are simply ignored.

Slide 6

VM/370

- Idea — provide multiple "virtual machines", each running its own o/s, which could be:
  - "Real" o/s such as MVS (another mainframe o/s) — in turn running many processes.
  - Not-quite-real o/s CMS — interactive single-user system rather like MS-DOS, runs under VM/370 only (not on real hardware).
- Allows sharing of physical resources among multiple "client" o/s's:
  - CPU sharing — similar to multitasking.
  - I/O device sharing — share physical devices, or allow exclusive use.
- (Aside: Textbook makes it sound like other IBM mainframe o/s's of that era did not support interactive users. Not so! Some did, though arguably not very efficiently.)

Slide 7

VM/370, Continued

- How does this work? briefly:
  - Applications running on client o/s run native code, request o/s services in the usual way (interrupt or system call).
  - Interrupt handler is part of VM/370 — so it processes I/O requests/interrupts, errors, etc.
  - Client o/s system code runs in simulated supervisor mode (really user mode) — so any use of privileged instructions traps to "real" o/s (VM/370).
- Successors to VM/370 (VM/ESA, z/VM) currently being used to run many copies of Linux on a mainframe (!).

Slide 8

Virtual Machines Revisited

- (More about virtualization in chapter 8 of textbook. Executive-level summary for now.)
- Several issues to address in implementing virtual-machine idea — sharing the CPUs (and what to do about privileged instructions), sharing memory, sharing I/O devices. Focus for now on just the first.
- Several basic approaches. Which to use depends partly on what hardware does if "sensitive" instructions (ideally all kernel-mode-only) in non-kernel mode.

Slide 9

Virtual Machines — Type-1 Hypervisor

- Idea here is like VM/370 — o/s that provides one abstraction, namely multiple virtual machines. Each virtual machine then runs a "client" o/s, *in user mode*.
- So what happens when a client o/s tries to execute a privileged instruction? it's as described for VM/370 — which only works if attempt generates an interrupt. True for hardware used for VM/370. Not true for i386. True (or at least possible) for more-recent x86's.

Slide 10

Virtual Machines — Type-2 Hypervisor

- Idea here is that the "virtual machine" is an application program running under a "real" o/s (the "host o/s") This application program runs a "guest o/s" in user mode.
- So what happens when guest o/s tries to execute a privileged instruction? Assumption is that it might or might not generate an interrupt, so idea is to avoid this happening. Requires examining code as it's loaded in and replacing any privileged instructions with calls to hypervisor. How to do this with reasonable efficiency? Some details in chapter 8. Doable though not trivial!

Slide 11

Virtual Machines — Paravirtualization

- Idea here is similar to type-1 hypervisor, but also (or instead?) providing support for "paravirtualized" client o/s's — i.e., o/s's in which all uses of privileged instructions have been replaced with calls to hypervisor code.
- Paravirtualization means client o/s doesn't do privileged instructions, so the whole problem of what should happen then goes away. It does mean changes to client o/s, though. More details in chapter 8, including some discussion of how to make it possible for a client o/s to run under more than one hypervisor, or without hypervisor.

Slide 12

Minute Essay

- What if anything was (is?) interesting about Homework 1? difficult?