Slide 1

Administrivia

- Reminder: Homework 5 due — nominally today, but really Thursday. ("Not accepted past" deadline for all homeworks?)
- Sample solutions in hardcopy only. All but Homework 5 available now (extras in envelope outside my office door). Homework 5 Thursday?
- Reminder: Final Friday. Review sheet on the Web. No formal review session but I am planning office hours. (Details by e-mail.)
- Some work graded but not all. I can send grade summaries — when?
- Should there be an extra-credit assignment?

Slide 2

Deadlocks — Introduction

- Some resources should not be shared — among processes, computers, etc.
- To enforce this, o/s (or whatever) provides mechanism to give one process at a time exclusive use, make others wait.
- Possibility exists that others will wait forever — deadlock.

Slide 3

Deadlocks — Definitions and Conditions

- Definition — set of processes is "deadlocked" if each process in set is waiting for an event that only another process in set can cause.
- Necessary conditions:
  - Mutual exclusion — resources can be used by at most one process at a time.
  - Hold and wait — process holding one resource can request another.
  - No preemption — resources cannot be taken away but must be released.
  - Circular wait — circular chain of processes exists in which each process is waiting for resource held by next.
- Modeling deadlock — "resource graphs" (examples in textbook).
- What do about them? Various approaches.

Slide 4

What To Do About Deadlocks

- Nothing — "ostrich algorithm" (ignore potential for deadlocks, hope they don't happen — sounds bad but often works okay in practice).
- Try to detect and recover (detect by building/maintaining graph and checking periodically for cycles, if found try to recover — but choices mostly not attractive).
- Try to "avoid" (various algorithms, mostly useful only if resource usage known in advance).
- Try to "prevent" (by making one of the four conditions impossible to satisfy — again, mostly not attractive, with one exception . . .)

Slide 5

Deadlocks Prevention — Order Resources

- Only strategy that (my opinion!) is promising.
- Idea is to impose total ordering on resources and require that processes obtain resources "in order".

Slide 6

Security — Overview

- Goals:
  - Data confidentiality — prevent exposure of data.
  - Data integrity — prevent tampering.
  - System availability — prevent DOS (denial of service).
- What can go wrong:
  - Deliberate intrusion — from casual snooping to "serious" intrusion.
  - Accidental data loss — "acts of God", hardware or software error, human error.

Slide 7

User Authentication

- Based on “something the user knows” — e.g., passwords. Problems include where to store them, whether they can be guessed, whether they can be intercepted.
- Based on “something the user has” — e.g., key or smart card. Problems include loss/theft, forgery.
- Based on “something the user is” – biometrics. Problems include inaccuracy/spoofing.

Slide 8

Attacks From Within

- Trojan horses (and how this relates to `$PATH`).
- Login spoofing (and how this related to the Windows control-alt-delete login prompt).
- Logic bombs and trap doors.
- Buffer overflows (and how this relates to, e.g, `gets`).
- Code injection attacks.
- And many more . . .

Slide 9

Buffer Overflows

- How many times, when you read the technical description of a security flaw, do you notice the phrase “buffer overflow”? (For me — often.)
- You already know what a buffer overflow is, from writing programs in C, and how it can lead to interesting(?) bugs.
- How can this be turned to advantage by crackers? Textbook provides a brief description. A frequently-mentioned paper is called “Smashing the Stack for Fun and Profit”. Interesting reading, but the methods apparently don’t work on systems that disallow executing code from “the stack”. Textbook mentions alternatives that do still work.

Slide 10

“Attacks From Within” — Summary?

- Textbook discusses several ways programs can be made to do things their authors would not want and probably did not intend — buffer overflows, code injection attacks, etc.
- Common factor (my opinion!) is what one might call insufficient paranoia on the part of the programmers.

Slide 11

Attacks From Outside

- Can categorize as viruses (programs that reproduce themselves when run), worms (self-replicating), spyware, etc. — similar ideas, though.
- Many, many ways such code can get invoked — when legit programs are run, at boot time, when file is opened by some applications ("macro viruses"), etc.
- Also many ways it can spread — once upon a time floppies were vector of choice, now networks or e-mail. Common factors:
  - Executable content from untrustworthy source.
  - Human factors.

  "Monoculture" makes it easier!
- Virus scanners can check all executables for known viruses (exact or fuzzy matches), but hard/impossible to do this perfectly.
- Better to try to avoid viruses — some nice advice in textbook.

Slide 12

"Attacks From Outside" — Summary?

- Textbook discusses several ways "malware" (viruses, worms, etc.) can infect a system.
- Common factor (my opinion!) is allowing execution of code that does something unwanted. (Either users don't realize this is happening, or they don't realize the implications?) Social engineering is often involved. Monoculture makes the malware writer's job easier.

Safe Execution of “Mobile” Code

Slide 13

- Is there a way to safely execute code from possibly untrustworthy source? Maybe — approaches include sandboxing, interpretation, code signing.
- Example — Java’s designed-in security:
  - At source level, very type-safe — no way to use `void*` pointers to access random memory. (Contrast with C and C++!)
  - When classes are loaded, “verifier” checks for potential security problems (not generated by normal compilers, but could be done by hand).
  - At runtime, security manager controls what library routines are called — e.g., applets by default can’t do file operations, many kinds of network access.

Trusted Systems

Slide 14

- Is it possible to write a secure O/S? Yes (says Tanenbaum).
- Why isn’t that done?
  - People want to run existing code.
  - People prefer (or are presumed to prefer) more features to more security.

Designing a Secure System

- “Security through obscurity” isn’t very.
- Better to give too little access than too much — give programs/people as little as will work.
- Security can’t be an add-on.
- “Keep it simple, stupid.”

Slide 15

Security — Summary

- Huge topic. Important and (I think!) interesting, though somewhat beyond the scope of this course.
- Shameless not-self-promotion: Strongly consider taking Dr. Myers’s course “Information Assurance and Security” (CSCI 3311).

Slide 16

Slide 17

Course Recap

- Four key areas (the gospel according to former chair Pitts):
  - Process management.
  - Memory management.
  - I/O management.
  - Filesystem management.
- Two views of operating systems:
  - “Virtual machine” that provides useful abstractions for applications programs, end users.
  - Resource manager.
- Also a little about history, a little about security.

Slide 18

Process Management

- O/S as virtual machine — process abstraction, “concurrent” execution, IPC, concurrent algorithms.
- O/S as resource manager — implementation of above, including interrupts and context switches, CPU scheduling.

Memory Management

- O/S as virtual machine — memory protection, virtual memory, “multiprogramming”.
- O/S as resource manager — implementation of above, including page replacement algorithms.

Slide 19

Filesystem Management

- O/S as virtual machine — filesystem abstractions (files, file attributes, directory structures).
- O/S as resource manager — implementation of above, disk-space management, reliability and consistency.

Slide 20

Slide 21

I/O Management

- O/S as virtual machine — layered abstractions for working with I/O devices (user-level s/w, device-independent s/w).
- O/S as resource manager — implementation of above, plus a little about lower-level interaction with devices (programmed versus interrupt-driven I/O versus DMA).

Slide 22

Recap, Continued

- Some recurring themes:
  - Interaction between h/w and s/w — some h/w features are there to support o/s features; o/s influenced by what's available in h/w.
  - Trade-offs — often the answer to "which is best?" is "it depends".
- We didn't cover the whole book, but if you look at the ACM's guidelines for an undergrad o/s course — we pretty much did what they said.

Slide 23

Recap, Continued

- A very smart person I know once said the only interesting part of an o/s course was concurrent algorithms, and the rest is "just details".

  A student a few years ago said "a lot of this just seems like common sense" (once you understand the basic ideas).

  Both sort of right . . .
- Goal of this course is to learn/retain basic ideas. Details may help with that — and can be interesting in themselves — but should not be the focus.

Slide 24

Minute Essay

- None really — sign in, but also tell me:
- Are you possibly interested in extra-credit problems?