Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on the Web; due in a week.
- No Quiz 6.

Slide 2

Minute Essay From Last Lecture

- Votes for “Windows way makes more sense” — familiarity, better for end users.
- Votes for “UNIX way makes more sense” — familiarity, simplicity(?).
- Votes for “it depends” —- end users versus programmers.

Slide 3

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 4

Resources

- “Resource” is anything that should be used by only one process at a time — hardware device, piece of information (e.g., database record), etc.

  Can be unique (e.g, particular database record) or non-unique (e.g., one block of a fixed-size disk area such as swap space).
- Preemptible versus non-preemptible — preemptible resources can be taken away from current owner without causing something to fail (e.g., memory); non-preemptible resources can’t (e.g., hardware device).
- Normal sequence for using a resource — request it, use it, release it. If not available when requested, block or busy-wait.

  Can easily implement this using semaphores, but then deadlock is possible if processes aren’t disciplined.

Slide 5

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 6

What To Do About Deadlocks — Nothing

- One strategy for dealing with deadlocks — “ostrich algorithm” (ignore potential for deadlocks, hope they don’t happen).
- Does this work?

Slide 7

Do Nothing, Continued

- Doesn't always work, of course.
- But simple to implement, and in practice works most of the time.

Slide 8

What To Do About Deadlocks — Detection and Recovery

- How to detect deadlocks — DFS on resource graph, (or if more than one resource of each type, algorithm from text).
- When to check for deadlocks:
  - Every time a resource is requested.
  - At regular intervals.
  - When CPU utilization falls below threshold.
- What to do if deadlock is found?
  - Preemption.
  - Rollback.
  - Process termination.
- Does this work?

Slide 9

Detection and Recovery, Continued

- Does work.
- But potentially time-consuming, and “what to do” choices aren’t very attractive!

Slide 10

What To Do About Deadlocks — Avoidance

- Can base on idea of “safe” states (in which it’s possible to schedule to avoid deadlock) versus “unsafe” states (in which it’s not). Idea is to avoid unsafe states. (Details in textbook.)
- “Banker’s algorithm” (Dijkstra, 1965) — idea is to never satisfy request for resource if it leads to unsafe state. (Details in textbook.)
- Does this work?

Slide 11

Avoidance, Continued

- Does work.
- But not much used because it assumes a fixed number of processes, resource requirements known in advance.

Slide 12

What To Do About Deadlocks — Prevention

- Idea here is to make it impossible to satisfy one of the four conditions for deadlock:
  - Mutual exclusion — don't allow more than one process to use a resource. E.g., define a printer-spool process to manage printer.
  - Hold and wait — require processes to request all resources at the same time and either get them all or wait.
  - No preemption — allow preemption.
  - Circular wait — impose strictly increasing ordering on resources, and insist that all processes request resources "in order".
- Do these work?

Slide 13

Prevention, Continued

- Don’t allow more than one process to use a resource:

  Solves immediate problem but may produce others.
- Require processes to request all resources at the same time and either get them all or wait:

  Works but may not be possible or efficient.
- Allow preemption.

  Not usually possible/desirable.
- Impose strictly increasing ordering on resources, and insist that all processes request resources “in order”.

  Works, but finding an ordering may be difficult.

Slide 14

Deadlocks — Related Issues

- Classical description is in terms of “resources”, but other kinds of deadlock are possible (e.g., involving communication).
- Other situations that aren’t classical deadlock but are also not good include “livelock” and “starvation” (see textbook).

Slide 15

Deadlocks — Summary

- Take-home message — there's some interesting theory related to this topic, but not a lot of practical advice, except for deadlock prevention.

Slide 16

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 17

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 18

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 19

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 20

“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 21

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 22

"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.

Minute Essay

- None — quiz.