Slide 1

Administrivia

- Review sheet for midterm on Web.
- Reminder: Homework 2 due Friday.

Slide 2

Recap — Scheduling Algorithms

- Main idea — decide which process to run next (when running process exits, becomes blocked, or is interrupted).
- Goal is to make decisions in a way that meets system objectives (minimize average turnaround time, maximize CPU usage, etc.)
- Some simple algorithms discussed last time. Some (e.g., FCFS) have limited practical value, but notice how many there are — many ways to approach the problem of “who’s next?”
- A few more . . .

Slide 3

Some Other Scheduling Algorithms

- Guaranteed scheduling.

  "Guarantee" each process (of N) 1/N of the CPU cycles; (try to) schedule to make this true.

  Calculate, for each process, fraction of the time it has had the CPU in its lifetime, fraction it "should" have had; choose process for which actual time / entitled time is smallest.

- Lottery scheduling.

  Give each process one or more "lottery tickets" — more or fewer depending on its priority (so to speak); pick one at random to decide who's next.

- Fair-share scheduling.

  Factor in process's owner in deciding which process to pick. I.e., if two "equal" users, schedule processes such that user A's processes get about as much time as those of user B.

Slide 4

Scheduling in Real-Time Systems

- "Real-time system" — system in which events must ("hard real time") or should ("soft real time") be handled by some deadline. Often events to be handled are periodic, and we know how often they arrive and how long they take to process.
- Role of scheduler in such systems could be critical.
- An interesting question — sometimes getting everything scheduled on time is impossible (example?). If we know periodicity and time-to-handle of all types of events, can we decide this?

  Suppose we have m types of events, and event type i has period P_i and time-to-handle C_i.

  General formula on p. 149; can be derived by extending from case where $m = 1 \ldots$
- Complex topic; see chapter 7 for more info.

Slide 5

Scheduling and Threads

- If system uses both processes and threads, we now possibly have an additional level of scheduling.
- Details depend on whether threads are implemented in user space or kernel space:
  - In user space — runtime system that manages them must do scheduling, and without the benefit of timer interrupts.
  - In kernel space — scheduling done at o/s level, so context switches are more expensive, but timer interrupts are possible, etc.

Slide 6

Evaluating Scheduling Algorithms

- How to decide which scheduling algorithm to use?
- One way — evaluate several choices, see which one best meets system goal(s). E.g., if the goal is minimum turnaround time, try to come up with an average turnaround time for each proposed choice.
- Several approaches possible . . . (This discussion is from another operating systems textbook, by Silberschatz and Galvin.)

Slide 7

Deterministic Modeling

- Idea — use a predetermined workload, compute values of interest (e.g., average turnaround time).
- How well does it work?

Slide 8

Deterministic Modeling, Continued

- Simple, fast, gives exact numbers.
- Requires exact numbers as input, and only applies to them.

Slide 9

Queueing Models

- Idea — use “queueing theory” to model system as a network of “servers”, each with a queue of waiting processes. (E.g., CPU is a server, with input queue of ready processes.)
- Input to model — distribution of process arrival times, CPU and I/O bursts for processes, as mathematical formulas. (Base this on measuring, approximating, or estimating.) In queueing-theory terms, “arrival rates” and “service rates”.
- Queueing theory lets you then compute utilization, average queue length, average wait time, etc.
- How well does it work?

Slide 10

Queueing Models, Continued

- Seems more general than deterministic modeling.
- But can be tricky to set up model correctly, and need to approximate / make assumptions may be a problem.

Slide 11

Simulations

- Idea — program a model of the computer system, simulating everything, including hardware.
- Two ways to get input for simulation:
  - Generate processes, burst times, arrivals, departures, etc., using probability distributions and random-number generation.
  - Create “trace tape” from running system.
- How well does it work?

Slide 12

Simulations, Continued

- Potentially very accurate.
- Time-consuming to program and to run!

Slide 13

Implementation

- Idea — code it up and try it!
- How well does it work?

Slide 14

Implementation, Continued

- Seems like potentially the most accurate approach.
- Requires a lot of work, resources.
- Involves implicit assumption that users' behavior is fairly constant.

  (So it's good to build into the algorithm some parameters that can be changed at run time, by users and/or sysadmin. In textbook's phrase, "separate mechanism from policy". Notice, though, users are apt to figure out how to game any system.)

Slide 15

What Do Real Systems Use?

- Traditional Unix: two-level approach (upper level to swap processes in/out of memory, lower level for CPU scheduling), using multiple-queue scheduling for CPU scheduling. See chapter 10 for details.
- Linux: facilities for soft real-time scheduling and "timesharing" scheduling, with the latter a mix of priority and round-robin scheduling. See chapter 10 for details.
- Windows NT/2000: multiple-queue scheduling of threads, with round-robin for each queue. See chapter 11 for details.
- MVS (IBM mainframe): three-level scheme with lots of options for administrator(s) to define complex policies.

Slide 16

Minute Essay

- None — sign in.