Slide 1

Administrivia

- Homework 2 to be on Web later today. Due next Thursday.
- Coming attractions (so to speak): Exam 1 10/16, Homework 3 due 10/14.

Slide 2

Minute Essay From Last Lecture

- Suppose you have a batch system with the following jobs. Compute turnaround times for all jobs using first FCFS and then SJF.

	running	arrival	start	stop	turnaround	start	stop	turnaround
job ID	time	time	(FCFS)	(FCFS)	(FCFS)	(SJF)	(SJF)	(SJF)
A	10	0	0	10	10	6	16	16
B	6	0	10	16	16	0	6	6
C	20	10	16	36	26	22	42	32
D	6	10	36	42	32	16	22	12

- Is it a coincidence that the ending time of the last job is the same for both?

Slide 3

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:
  - First Come, First Served (FCFS).
  - Shortest Job First (SJF).
  - Round-Robin (RR).
  - Priority scheduling.
  - Shortest Remaining Time Next.
- A less simple approach — three-level scheduling.

Slide 4

Multiple-Queue Scheduling

- Basic idea — variant on priority scheduling:
  - Divide processes into “priority classes”.
  - When picking a new process, pick one from the highest-priority class with ready processes.
  - Within a class, use some other algorithm to decide (round-robin, e.g.).
  - Optionally, periodically lower processes’ priorities.

Slide 5

Shortest Process Next

- Basic idea — like SJF, but for interactive processes:
  - Consider each interactive process as sequence of “jobs”.
  - In picking next process, pick the one with the shortest time.
  - Estimate time based on past performance:

    One way is “aging” — weighted sum of previous estimate and most recent run. Can be easy to calculate, emphasizes more recent behavior.

Slide 6

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 7

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.

  Derive formula on p. 149 . . .
- Complex topic, see chapter 7 for more info.

Slide 8

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 9

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 . . .

Slide 10

Deterministic Modeling

- Idea — use a predetermined workload, compute values of interest (e.g., average turnaround time).
- How well does it work?
  - Simple, fast, gives exact numbers.
  - Requires exact numbers as input, and only applies to them.

Slide 11

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?
  - 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 12

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?
  - 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?
  - Seems like potentially the most accurate approach.
  - Requires a lot of work, resources.
  - Involves implicit assumption that users' behavior is fairly constant.
- Because of this last point — 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".

Slide 14

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.
- MULTICS: multiple-queue scheduling.
- MVS (IBM mainframe): three-level scheme with lots of options for administrator(s) to define complex policies.

Slide 15

Minute Essay

- What's the most interesting thing you learned from reading chapter 2?