Slide 1

Administrivia

- Reminder: Homework 3 due Monday. Programming problem starter code and example updated this morning.
- Review sheet for midterm on the Web. Describes exam format and topics. Sample solutions for assignments will be available Monday.

Slide 2

Scheduling Algorithms — Review/Recap

- Purpose of a scheduling algorithm is to decide which process to run next.
- Many of them, ranging from simple to not-so-simple.
- Last time we talked about some simple ones. Continuing . . .

Slide 3

Sidebar — Simulating Scheduling Algorithms

- Can be helpful in understanding how these algorithms work to simulate what they do given a particular sequence of inputs.
- Example — batch system with the following jobs.

job ID	running time	arrival time
A	6	0
B	4	0
C	10	0
D	2	2

Asked to compute turnaround times for all jobs using FCFS, what would you do . . .

Slide 4

Priority Scheduling

- Basic ideas:
  - Keep a queue of ready processes, as before.
  - Assign a priority to each process.
  - When a process starts or becomes unblocked, add it to the end of the queue.
  - Switch when the running process exits or blocks, or possibly when a process starts. (I.e., preemption may be allowed.)
  - Next process is the one with the highest priority.
- Points to consider:
  - What happens to low-priority processes? (So, maybe we should change priorities sometimes?)
  - How do we decide priorities? (external considerations versus internal characteristics)

Slide 5

Shortest Remaining Time Next

- Basic idea — variant on SJF:
  - Assume that for each process (job), we know how much longer it will take.
  - Keep a queue of ready processes, as before; add to it as before.
  - Switch when the running process exits *or* a new process starts. (I.e., preemption allowed — requires recomputing time left for preempted process.)
  - Next process is the one with the shortest time left.
- Points to consider:
  - How does this compare with SJF?

Slide 6

Three-Level Scheduling

- Basic idea — break up problem of scheduling (batch) work into three parts:
  - Admissions scheduling — choose from input queue which jobs to "let into the system" (create processes for).
  - Memory scheduling — choose from among processes in system which to keep in memory, which to "swap out" to disk.
  - CPU scheduling — choose from among processes in memory which to actually run.
- Points to consider:
  - Are there advantages to limiting how many processes, how many in memory? What criteria could we use?
  - Are there advantages to the explicit three-level scheme?
  - Would this (or a variant) work for interactive systems?
  - Do all three schedulers have to be efficient?

Slide 7

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 8

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 9

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? (Yes — general formula in textbook; can be interesting to work through details.)
- Complex topic; see chapter 7 for more info.

Slide 10

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 11

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. As of kernel version 2.6.23, replaced with "Completely Fair Scheduler", which sounds like what Tanenbaum calls "guaranteed scheduling".
- Windows NT/2000/Vista: 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 12

Minute Essay

- Suppose you have a batch system with the following jobs.

job ID	running time	arrival time
A	6	0
B	4	0
C	10	0
D	2	2

Compute turnaround times for all jobs using SJF.

Slide 13

Minute Essay Answer

- Solution:

job ID	start time	stop time	turnaround time (SJF)
A	6	12	12
B	0	4	4
C	12	22	22
D	4	6	4