

### Administrivia

- Reminder: Homework 2 due today by 5pm (hardcopy for non-programming problems, e-mail for code — really, any time before midnight on code is okay).

Slide 1

### Minute Essay From Last Lecture

- How to decide which scheduler to use? many suggestions:
  - Do experiments (possibly focusing on particular kinds of work) or simulation.
  - Research existing algorithms.
  - Ask about goals / consider type of work / analyze system needs.
- Also there were mentions of specific algorithms, and a couple of entertaining speculations on writing one's own.

Slide 2

### Recap — Scheduling Algorithms

Slide 3

- 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?”

### Scheduling in Real-Time Systems

Slide 4

- “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 on p. 149; interesting to work through details.)
- Complex topic; see chapter 7 for more info.

### Scheduling and Threads

Slide 5

- 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.

### Evaluating Scheduling Algorithms

Slide 6

- 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.)

### Deterministic Modeling

- Idea — use a predetermined workload, compute values of interest (e.g., average turnaround time, as in Homework 3 problem).
- How well does it work?

Slide 7

### Deterministic Modeling, Continued

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

Slide 8

### Queueing Models

Slide 9

- 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?

### Queueing Models, Continued

Slide 10

- 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.

## Simulations

Slide 11

- 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?

## Simulations, Continued

Slide 12

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

## Implementation

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

Slide 13

## 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 14

### What Do Real Systems Use?

Slide 15

- 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. Newest kernel (not yet released?) will use the “Completely Fair Scheduler”, which sounds like what Tanenbaum calls “guaranteed scheduling”.
- 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.

### Minute Essay

Slide 16

- What did you find interesting / annoying / difficult about Homework 2?