Administrivia

- Reminder: Homework 2 due today.

- Notice that Homework 3 written problems are due at class time next Wednesday.

- (Review minute essay from last time.)

Slide 1

Review — Processes and Context Switches

- Recall idea behind process abstraction — make every activity we want to manage a "process", and run them "concurrently".

- Apparent concurrency provided by interleaving. (Some) true concurrency provided by multiple cores/processors.

- To make this work — process table, ready/running/blocked states, context switches.

- Context switches triggered by interrupts — I/O, timer, system call, etc.

- On interrupts, interrupt handler processes interrupt, and then goes back to some process — but which one?

Slide 2

Which Process To Run Next?

- Deciding what process to run next — scheduler/dispatcher, using "scheduling algorithm".

- When to make scheduling decisions?

  - When a new process is created.

  - When a running process exits.

  - When a process becomes blocked (I/O, semaphore, etc.).

  - After an interrupt.

- One possible decision — "go back to interrupted process" (e.g., after I/O interrupt).

Slide 3

Scheduler Goals

- Importance of scheduler can vary; extremes are

  - Single-user system — often only one runnable process, complicated decision-making may not be necessary (though still might sometimes be a good idea).

  - Mainframe system — many runnable processes, queue of "batch" jobs waiting, "who's next?" an important question.

  - Servers / workstations somewhere in the middle.

- First step is to be clear on goals — want to make "good decisions", but what does that mean? Typical goals for any system:

  - Fairness — similar processes get similar service.

  - Policy enforcement — "important" processes get better service.

  - Balance — all parts of system (CPU, I/O devices) kept busy (assuming there is work for them).

Slide 4

Aside — Terminology

Slide 5

- Discussion often in term of "jobs" — holdover from mainframe days, means "schedulable piece of work".

- Processes usually alternate between "CPU bursts" and I/O, can be categorized as "compute-bound" ("CPU-bound") or "I/O bound".

- Scheduling can be "preemptive" or "non-preemptive".

Scheduler Goals By System Type

Slide 6

- For batch (non-interactive) systems, possible goals (might conflict):
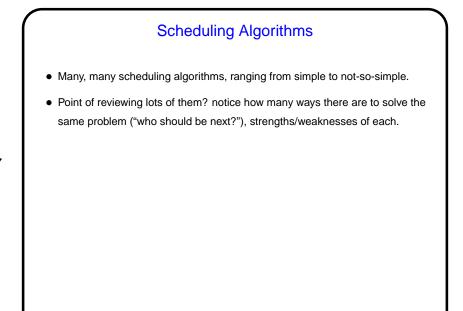 - Maximize throughput — jobs per hour.
  - Minimize turnaround time.
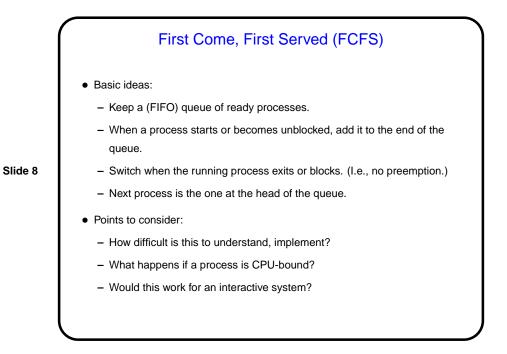 - Maximize CPU utilization.

  Preemptive scheduling may not be needed.

- For interactive systems, possible goals:
  - Minimize response time.
  - Make response time proportional (to user's perception of task difficulty).

  Preemptive scheduling probably needed.

- For real-time systems, possible goals:
  - Meet time constraints/deadlines.
  - Behave predictably.

Scheduling Algorithms

Slide 7

- Many, many scheduling algorithms, ranging from simple to not-so-simple.

- Point of reviewing lots of them? notice how many ways there are to solve the same problem ("who should be next?"), strengths/weaknesses of each.

First Come, First Served (FCFS)

Slide 8

- Basic ideas:
  - Keep a (FIFO) queue of ready processes.
  - When a process starts or becomes unblocked, add it to the end of the queue.
  - Switch when the running process exits or blocks. (I.e., no preemption.)
  - Next process is the one at the head of the queue.

- Points to consider:
  - How difficult is this to understand, implement?
  - What happens if a process is CPU-bound?
  - Would this work for an interactive system?

Shortest Job First (SJF)

- Basic ideas:
    - Assume work is in the form of "jobs" with known running time, no blocking.
    - Keep a queue of these jobs.
    - When a process (job) starts, add it to the queue.
    - Switch when the running process exits (i.e., no preemption).
    - Next process is the one with the shortest running time.
- Points to consider:
    - How difficult is this to understand, implement?
    - What if we don't know running time in advance?
    - What if all jobs are not known at the start?
    - Would this work for an interactive system?
    - What's the key advantage of this algorithm?

Slide 9

Round-Robin Scheduling

- Basic ideas:
    - Keep a queue of ready processes, as before.
    - Define a "time slice" — maximum time a process can run at a time.
    - When a process starts or becomes unblocked, add it to the end of the queue.
    - Switch when the running process uses up its time slice, or it exits or blocks. (I.e., preemption allowed!)
    - Next process is the one at the head of the queue.
- Points to consider:
    - How difficult is this to understand, implement?
    - Would this work for an interactive system?
    - How do you choose the time slice?

Slide 10

Slide 11

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 12

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 13

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 14

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 15

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 16

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; interesting to work through details.)

- Complex topic; see chapter 7 for more info.

Slide 17

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 18

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.

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 19

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

Slide 20