Slide 1

Administrivia

- Lecture notes online; some have extra material (e.g., message-passing examples).

Slide 2

Recap — Synchronization Mechanisms

- What's the point? need some way to make one process (or thread) wait/block until another does something.

  Relevant to systems-level programming, also for "parallel" applications.
- One view — mechanism as ADT (or similar), how to use.

  Some require compiler support; others provided as library functions. E.g., `man pthread_mutex_init` ("lock" ADT), `man sem_init`.
- Another view — implementing the ADT.

  "Wait/block" can mean busy-waiting or changing process state to "blocked".

  At lowest level, typically make use of hardware feature such as `TSL`.

Slide 3

Review — Processes and Context Switches

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

  (Try `ps -A f` on a Linux system.)
- Each process has a "virtual CPU" (registers, program counter, etc.) and is running some program.

  ("Heavyweight processes" have other resources too — address space, files, etc. "Lightweight processes" (threads) share.)

  Sometimes program must wait — for I/O, because of synchronization mechanism, etc.
- Apparent concurrency provided by interleaving.

Slide 4

Review — Processes and Context Switches

- 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 5

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 6

Scheduler Goals

- Importance of scheduler can vary; extremes are
  - Single-user system — often only one runnable process, complicated decision-making may not be necessary.
  - 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).

CSCI 4320 September 30, 2003

Slide 7

Aside — Terminology

- 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”.

Slide 8

Scheduler Goals By System Type

- 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.

Slide 9

First Come, First Served (FCFS)

- 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?

Slide 10

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?

CSCI 4320 September 30, 2003

Slide 11

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 12

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 13

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 14

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?

CSCI 4320 September 30, 2003

Slide 15

Minute Essay

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

job ID	running time	arrival time
A	10	0
B	6	0
C	20	10
D	6	10

Compute turnaround times for all jobs using first FCFS and then SJF.