Administrivia

- Homework 1 due date moved to Monday at 5pm.

Slide 1

Process Abstraction

- We want o/s to manage "things happening at the same time" — applications, hidden tasks such as managing a device, etc.

- Key abstraction for this — "process" — program plus associated data, including program counter.

- Can also associate with process an "address space" — range of addresses the program can use. Simplifying a little, this is "virtual memory" (like the virtual CPU) that only this process can use.

- True concurrency ("at the same time") requires more than one CPU (more properly now, "more than one CPU/core"?). Can get apparent concurrency via interleaving — model one virtual CPU per process and have the real processor switch back and forth among them ("context switch").

  (Aside: In almost all respects, this turns out to be indistinguishable from true concurrency. "Hm!"?)

Slide 2

Context Switches

- What is it? switch from one process to another.

- When should this happen?

Slide 3

Context Switches, Continued

- Should happen
  - when a process's "time slice" is up.
  - when there's an unrecoverable error.
  - when there's something that needs to be done right away (e.g., deal with input/output).
  - maybe other times? (when a process has to wait for something, e.g.).

  All signalled by some kind of interrupt.

- Goal is to suspend work on a process such that we can later pick up exactly where we left off. How do we make that happen?

  (Think about what the hardware does when an interrupt happens, what's included in that "virtual CPU".)

Slide 4

Slide 5

Context Switches, Continued

- On interrupt, hardware saves program counter (at least — why?), transfers control to fixed location — which contains o/s code.

- That O/S code has to
  - Save CPU state (program counter, registers, etc.) for the current process.
  - Deal with interrupt (details depend on type — I/O versus timer versus . . .).
  - Restore CPU state for "next" process (previously saved), thereby restarting it.
    ("Next" process? yes, o/s might have to choose — more about that later.)

Slide 6

Process Creation and Termination

- When are processes created?
  - At system startup.
  - When another process makes a "create process" system call — e.g., to start a new application.

- When are processes destroyed?
  - At program exit.
  - After some kinds of errors.
  - When another process makes a "kill process" system call.

Process States

Slide 7

- Can think of processes as being in one of three states:

  - "Running" — being executed by a CPU.

  - "Blocked" — waiting for something to happen (I/O to complete, another process to do something, etc.) and unable to do anything useful until it does.

  - "Ready" — not blocked, but waiting because all CPUs are currently executing other processes.

- Possible transitions? Which ones require decision-making?

Process States, Continued

Slide 8

- Possible transitions:

  - Running to blocked — happens when, e.g., a process makes an I/O request and can't continue until it's complete.

  - Blocked to ready — happens when the event the blocked process is waiting for occurs.

  - Running to ready, ready to running — needed if we want some sort of time-sharing (give all non-blocked processes "a turn" frequently).

- Notice that moving to and from "blocked" state doesn't involve decision-making, but ready/running transitions do.

- The decision-maker — "scheduler" (to be discussed later). Often "running to ready" is triggered by an interrupt (I/O, timer, etc.), and "ready to running" involves this scheduler.

Minute Essay

Slide 9

- In a system with 8 CPUs and 100 processes, what are the maximum and minimum number of processes that can be running? ready? blocked?

Minute Essay Answer

Slide 10

- Blocked: Maximum of 100 (unless you assume that there's an "idle" operating system process that runs when nothing else does and never blocks, and maybe one of these is needed for every CPU). Minimum of 0.

- Running: Maximum of 8, because there are 8 CPUs. Minimum of 0 (again unless you assume that there's an o/s process that runs when nothing else does).

- Ready: Maximum of 92, since all CPUs will be running processes if there are any that can be run. (Depending on details, you might have to add "except during context switches, when the scheduler is choosing the next process to run on a CPU".) Minimum of 0, since they could all be blocked or running.