Administrivia

Slide 1

- Reminder: Homework 1 due today at 5pm. Submit code by e-mail. For everything else I prefer hard copy.

  (Office/lab hours this afternoon if last-minute questions.)

Minute Essay From Last Lecture

Slide 2

- (See notes from last time.)
- Most people more or less got the answers I had in mind. (Apparently no budding lawyers here quibbling about details.) Minimum number running — is there one? Answer might be “it depends”.

Minute Essay

- FIX THIS

Slide 3

Implementing Processes

- Think about how you would implement this abstraction . . .
- First, you'd want a data structure to represent each process, to include — what?

Slide 4

Slide 5

Implementing Processes, Continued

- Data structure to represent each process would include some way to represent such things as:
  - Process ID.
  - Process state (running / ready / blocked).
  - Information needed for context switch — a place to save program counter, registers, etc.
  - Other stuff as needed — a list of open files, e.g.
- Then you'd collect these into a table (or some similar structure) — "process control table", with individual data structures being "entries in the process control table" or "process control blocks".

Slide 6

Implementing Processes, Example — Linux

- Each process ("task") is represented by a C `struct` containing information similar to what we described.
- These `struct`s are chained as a doubly-linked list; there is also a hash table keyed by PID.
- (This is according to online information about the 2.4 kernel.)

Slide 7

Processes Versus Threads

- So far I've used "process" in an abstract/general way.
- In typical implementations, though, "process" is more specific — something that has its own address space, list of open files, etc. Often these are called "heavyweight processes".
  - Advantages — such processes don't interfere with each other.
  - Disadvantages — they can't share data, switching between them is expensive ("a lot of state" to save/restore).
- For some applications, might be nice to have something that implements the abstract process idea but allows sharing data and faster context switching — "threads".

Slide 8

Threads

- So, threads are another way to implement the process abstraction.
- Typically, a thread is "owned" by a (heavyweight) process, and all threads owned by a process share some of its state — address space, list of open files.
- However, each thread has a "virtual CPU" (a distinct copy of registers, including program counter).
- Implementation involves data structures similar to process table.
- Advantages / disadvantages (compared to processes)?

Slide 9

Threads, Continued

- Advantages: threads can share data (same address space), switching from thread to thread is fairly fast.
- Disadvantages: sharing data has its hazards (more about this later).

Slide 10

Implementing Threads

- Two basic approaches — “in user space” and “in kernel space” (next two slides).
- Various hybrid schemes also possible.

Slide 11

Implementing Threads “In User Space”

- Basic idea — operating system thinks it’s managing single-threaded processes, all the work of managing multiple threads happens via library calls within each process.
- Advantages / disadvantages?

Slide 12

Implementing Threads “In User Space”, Continued

- Advantages: fewer system calls, hence probably more efficient.
- Disadvantages:
  - If a thread blocks, it may do so in a way that blocks the whole process.
  - Preemptive multitasking is difficult/impossible.
  - Using multiple CPUs is difficult/impossible.

Slide 13

Implementing Threads “In Kernel Space”

- Basic idea — operating system is involved in managing threads, the work of managing multiple threads happens via system calls (rather than user-level library calls).
- Advantages / disadvantages?

Slide 14

Implementing Threads “In Kernel Space”, Continued

- Advantages: avoids the difficulties of implementing in user space.
- Disadvantages: probably less efficient.

Slide 15

Threads — Example Implementations

- Unix systems vary as to which they use (see chapter 10 for more info). Early versions of Linux provided no support for kernel-space threading, but there were libraries for the user-space version. Kernel now provides support, but threads apparently basically processes with some different flags allowing them to share memory, etc.
- Windows NT/2000 apparently is such that *all* processes have at least one thread, and the basic scheme is either kernel-space or a hybrid (see chapter 11 for more info).

Slide 16

Minute Essay

- What did you learn from doing Homework 1 (questions about what should be allowed in user/supervisor mode, tracing system calls, writing a simple shell)? Also tell me about anything you found particularly easy / difficult / interesting / annoying.
- I'm reviewing options for out-of-class assignments for the rest of the course. Would you welcome more emphasis on programming? Are you comfortable in Java? C++? C?