Slide 1

Administrivia

- Classes next week: I plan to be at a conference but may be able to find a guest lecturer. I'll notify you via the course Web page and e-mail.
- Homework 1 to be on Web by next week. Due the following week.
- (Minute essays from first class — people did some interesting things over the summer!)

Slide 2

Operating System Functionality

- Provide a "virtual machine":
  - Filesystem abstraction — files, directories, ownership, access rights, etc.
  - Process abstraction — "process" is a name for one of a collection of "things happening at the same time" (in effect if not in fact), including:
    - In batch systems, user "jobs", plus input/output spooling.
    - In timesharing system, interactive users.
    - In PC o/s, concurrently-executing tasks.

    Here too, idea of ownership / access rights.
- Manage resources (probably on behalf of multiple users/applications):
  - Memory.
  - CPU cycles (one or more CPUs).
  - I/O devices.

Slide 3

Overview of Hardware

- Simplified view of hardware (as it appears to programmers) — processor(s), memory, I/O devices, bus.
- (See figure, p. 21.)
- Next few sections talk about each component — what it does (from user's point of view) and low-level interface to software. Today, look at processor only; other components later.

Slide 4

Processors

- "Instruction set" of primitive operations — load/store, arithmetic/logical operations, control flow.
- Basic CPU cycle — fetch instruction, decode, execute.
- Registers — "local memory" for processor; general-purpose registers for arithmetic and other operations, special registers (program counter, stack pointer, program status word (PSW)).
- Now consider what additional features would make it easier to write an operating system . . .

Slide 5

Interrupt Mechanism

- Very useful to have a way to interrupt current processing when an unexpected or don't-know-when event happens — error occurs (e.g., invalid operation), I/O operation completes.
- On interrupt, goal is to save enough of current state to allow us to restart current activity later:
  - Save old value of program counter.
  - Disable interrupts.
  - Transfer control to fixed location ("interrupt handler" or "interrupt vector") — normally o/s code that saves other registers, re-enables interrupts, decides what to do next, etc.
- Usually have a `TRAP` instruction for generating interrupt.
- Could you write an o/s without this?

Slide 6

Dual-Mode Operation, Privileged Instructions

- Useful to have mechanism to keep application programs from doing things that should be reserved for o/s.
- Usual approach — in hardware, define two modes for processor (supervisor and user), privileged instructions.
  - Privileged instructions — things only o/s should do, e.g., enable/disable interrupts.
  - Bit in PSW indicates supervisor mode (o/s only, privileged instructions okay) or user mode (application programs, privileged instructions not allowed).
  - When to switch modes? when o/s starts application program, when application program requests o/s services, on error.
- Could you write an o/s without this?

Memory Protection

Slide 7

- Very useful to have a way to give each process (including o/s) its own variables that other processes can't alter.
- Usual approach — provide a hardware mechanism such that attempting to access memory out of ranges generates exception/interrupt; several ways, including:
  - Limit each process to a range of memory locations; hold starting and ending addresses in special registers.
  - Partition memory into blocks, give each block a numeric key, give each process a key, and only allow processes to access blocks if keys match.
- Could you write an o/s without this?

Timer

Slide 8

- Useful to have a way to set a timer / "alarm clock" — e.g., to get control back if application program enters infinite loop.
- Usual approach — hardware features that tracks real time and can be set to interrupt CPU.

Slide 9

Operating System Services, Again

- Process management.
- Memory management.
- I/O subsystem.
- File systems.
- Security.
- Shell.

Slide 10

Shell

- History — early batch systems had to interpret “control cards”; modern equivalent is to interpret “commands” (usually interactive).
- Not technically part of o/s, but important and related.
- Typical shell functionality:
  - Invocation of programs (optionally in background).
  - Input/output redirection.
  - Program-to-program connections (pipes).
  - “Wildcard” capability.
  - Scripting capability.
- Examples — MS-DOS `command.com`; Unix `sh`, `bash`, `csh`, `tcsh`, `ksh`, `zsh`, ...

System Calls

Slide 11

- Recall — some things can/should only be done by o/s (e.g., I/O), but application programs need to be able to request them.
- How to make this work — “system call” (good discussion on pp. 45–46):
  - Library routine (running in user mode) sets up parameters and issues `TRAP` instruction or similar — causing an interrupt.
  - Interrupt handler (running in supervisor mode) processes system call using parameters set up by library routine.
  - Control returns to library routine in user mode.
- Typical services provided — creating processes, creating files and directories, etc., etc. — see tables in textbook (Unix on p. 47, Windows on p. 55).

Minute Essay

Slide 12

- I once had a learning experience about “how DOS is different from a real o/s”. Summary version: A program using pointers (possibly uninitialized) caused the whole machine to lock up, so thoroughly that the only recovery was to power-cycle.

  What do you think went wrong?

Minute Essay Answer

- The program changed memory at the addresses pointed to by the uninitialized pointers — and this memory was being used by the o/s, possibly to store something related to interrupt handling. A “real” o/s wouldn’t allow this!