Slide 1

Administrivia

- (None.)

Slide 2

Operating System Functionality — Overview

- 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” (multiple users, multiple applications, background activities such as print spooling, etc.).
- 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.
- Figure on p. 19 shows simplified view of overall organization — components connected to a single bus. (Actual processors may have more than one bus.)

Slide 4

Processors

- "Instruction set" of primitive operations — load/store, arithmetic/logical operations, control flow.
- Basic CPU cycle — fetch instruction, decode, execute. (Again, this is simplified — pipelined or "superscalar" architectures overlap these steps.)
- Registers — "local memory" for processor; general-purpose registers for arithmetic and other operations, special registers (program counter, stack pointer, program status word (PSW)).
- Not shared among processes in the sense of simultaneous use (but context switches "fake it").
- Typically also include features useful in writing an operating system . . .

Slide 5

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/kernel and user), privileged instructions.
  - Privileged instructions — things only o/s should do, e.g., enable/disable interrupts.
  - Bit in PSW indicates kernel 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.
  - How to switch? kernel to user seems straightforward, but how about the other way? Usually handled via TRAP or similar instruction, which generates an interrupt (more about interrupts later).

Slide 6

Multithreaded and Multicore Chips

- For many years (at least 20, to my knowledge) advocates of parallel programming have been saying that eventually hardware designers would run out of ways to make single processors faster — and finally it seems to be happening.
- Basic idea — number of transistors one can put on a chip is still increasing, but how to use them to make single processors faster isn't clear. So, instead, hardware designers have chosen to provide (more) hardware support for parallelism. Two basic approaches:
  - Multithreading — e.g., Intel's "hyperthreading" basically allows fast switching between two threads, but not true parallel execution.
  - Multicore — multiple independent CPUs on a chip, possibly sharing cache.

Slide 7

Memory Hierarchy

- In a perfect world — fast, big, cheap, as permanent as desired.
- In this world — hierarchy of types, from fast but expensive to slow but cheap: registers, cache, RAM, magnetic disk, magnetic tape. (See picture, p. 23.)
- Note also — some types volatile, some non-volatile.

Slide 8

Registers and Caches

- Registers — part of processor, fastest to access but most expensive to build. Managed explicitly in software.
- Caches (possibly multiple levels) — less fast, less expensive, bigger. Mostly managed by hardware.
- Aside: Caching is a widely used strategy in computing! virtual memory, disk blocks in memory, etc., etc.

Slide 9

Main Memory (RAM)

- Still less fast, less expensive, bigger.
- Shared among processes — which presents some interesting challenges . . .

Slide 10

Memory Protection

- 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.

Slide 11

Minute Essay

- 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?

Slide 12

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!

  (Then again, the version of MS-DOS in question was supposedly written to run on hardware that didn't provide memory protection, so maybe it's not DOS's fault.)