Slide 1

Administrivia

- No class Thursday — I plan to be at a conference.
- Homework 1 (covering chapter 1) on Web, linked from the "Lecture topics and assignments" page. Due date September 16.

Slide 2

Minute Essay From Last Lecture

- Question: 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 and need to be power-cycled. What do you think went wrong?
- 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!

Slide 3

Hardware Overview, Recap

- Simplified view of basic hardware components (processor, memory, I/O devices) — their purpose from a user's perspective, how they work from an assembly-language programmer's perspective.
- Interaction between hardware design and o/s design — what the hardware can do influences o/s design, but what o/s designers want also influence hardware design.

Slide 4

Operating System Services, Again

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

Slide 5

Process Management

- "Process" abstraction to represent one of a collection of "things happening at the same time".

  A working definition — "program in execution" (program code plus associated variables, sequence of states tracking progress through code and changes in variables).

- "Concurrent" execution via interleaving of actions.

  In effect, each process has a "virtual CPU", with the actual CPU repeatedly suspending one process to work on another ("context switch").

- O/s must provide a way to manage this, including ways to create processes, allow/force them to terminate, communicate among them (e.g., to coordinate/synchronize).

Slide 6

Memory Management

- Managing physical memory:
  - How to divide it up among processes/programs/users — each has an "address space" of memory it can access.
  - How to protect each process's memory from other processes (requires h/w support, but managed by o/s).
- Managing address spaces (virtual memory):
  - Originally, address space limited by size of physical memory.
  - "Virtual memory" allows bigger address spaces, by shuffling data between disk and physical memory.

Slide 7

I/O Subsystem

- Encapsulates messy low-level details.
- Allows sharing of I/O devices among programs/users.

Slide 8

File Systems

- "File system" abstraction, including:
  - "File" abstraction — collection of related information, possibly with associated ownership, permissions, timestamps, etc.
  - "Directory" abstraction.
  - "Path names" — absolute and relative.
  - "Opening a file" — connecting program to file (check permissions, etc., return "file descriptor").
- Additional Unix ideas/terms:
  - Mounting filesystems.
  - Special files — idea is to treat other devices (e.g., printers) like files.
  - Pipes — connections between processes that can be treated like file.

Slide 9

Security

- Protect users/applications from each other.
- Protect users/applications from the outside world.

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`, ...

Slide 11

System Calls

- 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).

Slide 12

Operating System Structures

- Clearly o/s could involve a whole lot of code — how to structure?
- Some choices:
  - Monolithic systems.
  - Layered systems.
  - Virtual machines.
  - Exokernels.
  - Client-server model.

Slide 13

Monolithic Systems

- Tanenbaum's description — "The Big Mess".
- Examples include MS-DOS, early Unix.
- Advantages? "works, sort of" — often justification is historical.
- Disadvantages? "big mess". (Not everyone agrees, though.)

Slide 14

Layered Systems

- Idea — use layers of abstraction, just as one structures application programs.
- Examples include THE, MULTICS, OS/2, Windows NT (more so in early releases).
- Advantages? — nice separation of concerns, modularity.
- Disadvantages? — tricky to plan layers, performance can be slow.

Slide 15

Virtual Machines

- Idea — o/s provides a simulation of the actual physical machine, this "virtual machine" then runs another o/s – or several of them.
- Examples include VM/370, Windows support for old MS-DOS programs, VMware, Mac-on-Linux, Java Virtual Machine.
- Advantages? — separates multiprogramming from other concerns, emulation aspect can be useful, useful in o/s development.
- Disadvantages? — another layer, so can be slower.

Slide 16

VM/370

- Idea — provide multiple "virtual machines", each running its own o/s, which could be:
  - "Real" o/s such as MVS (another mainframe o/s) — in turn running many processes.
  - Not-quite-real o/s CMS — interactive single-user system rather like MS-DOS, runs under VM/370 only (not on real hardware).
- Allows sharing of physical resources among multiple "client" o/s's:
  - CPU sharing — similar to multitasking.
  - I/O device sharing — share physical devices, or allow exclusive use.

Slide 17

VM/370, Continued

- How does this work? briefly:
  - Client o/s's run native code, request o/s services in the usual way (interrupt or system call).
  - Interrupt handler is part of VM/370 — so it processes I/O requests/interrupts, errors, etc.
  - Client o/s system code runs in simulated supervisor mode (really user mode).
- Successors to VM/370 (VM/ESA, z/VM) currently being used to run many copies of Linux on a mainframe (!).

Slide 18

Minute Essay

- This wraps up lectures on chapter 1; is there anything that was particularly unclear or you want to know more about?