Software systems and issues

- **operating systems**
  - controlling the computer
- **file systems and databases**
  - storing information
- **applications**
  - programs that do things
- **middleware, platforms**
  - where programs meet systems

- **interfaces, standards**
  - agreements on how to communicate and inter-operate
- **open source software**
  - freely available software
- **intellectual property**
  - copyrights, patents, licenses

Operating system

- **a program that controls the resources of a computer**
  - interface between hardware and all other software
  - examples: Windows 95/98/NT/ME/2000/XP/Vista/7,
    Unix/Linux, Mac OS X, iOS, Symbian, PalmOS, ...

- **runs other programs ("applications", your programs)**
- **manages information on disk (file system)**
- **controls peripheral devices, communicates with outside**

- **provides a level of abstraction above the raw hardware**
  - makes the hardware appear to provide higher-level services than it really does
  - makes programming much easier

What's an operating system?

"Operating system" means the software code that, inter alia, (i) controls the allocation and usage of hardware resources (such as the microprocessor and various peripheral devices) of a Personal Computer, (ii) provides a platform for developing applications by exposing functionality to ISVs through APIs, and (iii) supplies a user interface that enables users to access functionality of the operating system and in which they can run applications.

US District Court for the District of Columbia
Final Judgment, State of New York, et al v. Microsoft Corporation
November 1, 2002

History of general-purpose operating systems

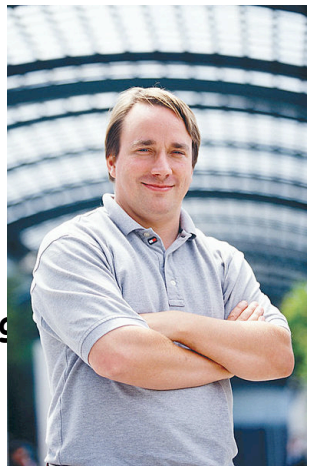
- **1950's: signup sheets**
- **1960's: batch operating systems**
  - operators running batches of jobs
  - OS/360 (IBM)
- **1970's: time-sharing**
  - simultaneous access for multiple users
  - Unix (Bell Labs; Ken Thompson & Dennis Ritchie)
- **1980's: personal computers, single user systems**
  - DOS, Windows, MacOS
  - Unix
- **1990's: personal computers, PDA's, …**
  - PalmOS, Windows CE, …
  - Unix / Linux
- **2000's: Windows vs. Unix/Linux?**
  - Mac OS X is a Unix system
- **2010's: Apple vs. Google?**
  - iOS, Android, Chrome-OS, … (Unix/Linux-based)
- **not all computers have general-purpose operating systems**
  - "embedded systems": small, specialized, but increasingly general

Unix operating system

- **developed ~1971 at Bell Labs**
  - by Ken Thompson and Dennis Ritchie
- **clean, elegant design**
  - at least in the early days
- **efficient, robust, easy to adapt, fun**
  - widely adopted in universities, spread from there
- **written in C, so easily ported to new machines**
  - runs on everything (not just PC's)

- **influence**
  - languages, tools, de facto standard environment
  - enabled workstation hardware business (e.g., Sun Microsystems)
  - supports a lot of Internet services and infrastructure

Linux



- **a version of Unix written from scratch**
  - by Linus Torvalds, Finnish student (started 1991)

- **source code freely available (kernel.org, linux.org**
  - large group of volunteers making contributions
  - anyone can modify it, fix bugs, add features
  - Torvalds approves, sets standard
  - commercial versions make money by packaging and support, not by selling the code itself

- **runs some major operations**
  - Google, Amazon, ABC, CBS, CNN, YouTube, ...

What an operating system does

- **manages CPU, schedules and coordinates running programs**
  - switches CPU among programs that are actually computing
  - suspends programs that are waiting for something (e.g., disk, network)
  - keeps individual programs from hogging resources
- **manages memory (RAM)**
  - loads programs in memory so they can run
  - swaps them to disk and back if there isn't enough RAM (virtual memory)
  - keeps separate programs from interfering with each other
  - and with the operating system itself (protection)
- **manages and coordinates input/output to devices**
  - disks, display, keyboard, mouse, network, ...
  - keeps separate uses of shared devices from interfering with each other
  - provides uniform interface to disparate devices
- **manages files on disk (file system)**
  - provides hierarchy of directories and files for storing information

To run programs, the operating system must

- **fetch program to be run (usually from disk)**
- **load it into RAM**
  - maybe only part, with more loaded as it runs (dynamic libraries)
- **transfer control to it**
- **provide services to it while it runs**
  - reading and writing info on disk
  - communications with other devices
- **regain control and recover resources when program is finished**
- **protect itself from errant program behavior**
- **share memory & other resources among multiple programs running "at the same time"**
  - manage memory , disks, network, ...
  - protect programs from each other
  - manage allocation of CPUs among multiple activities

Memory management

- **what's in memory? over-simplified pictures:**

Unix:

Windows:

- **reality is more complicated**
  - pieces of programs are partly in RAM, partly on disk
    can only execute instructions that are in RAM
- **memory protection:**
  - making sure that one program can't damage another or the OS
- **virtual memory:**
  - making it look like there is more RAM than there really is

Operating system controls devices

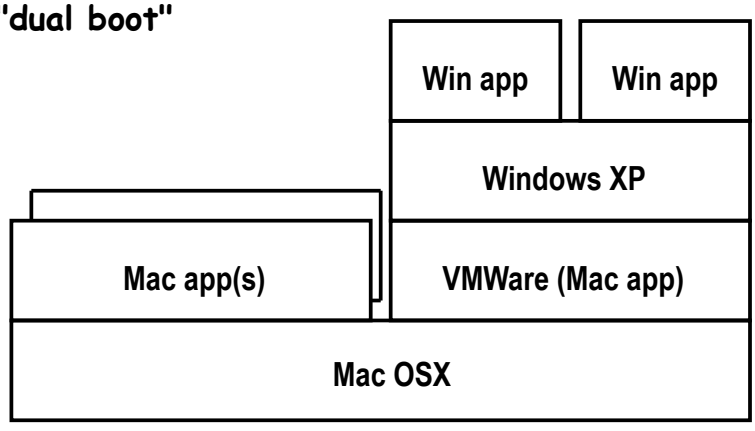
- **operating system hides physical properties of devices**
  - device has specific capabilities, parameters, etc.
  - hardware and software in device and OS present these at higher level
  - e.g., printer
    logical view: put characters out in 66 lines of 80 characters
    physical view: paint individual bits of characters in raster across page
  - e.g., CD-ROM
    logical view: file system just like the one on the hard drive
    physical view: long spiral of individual bits read by a laser
- **OS uses device drivers to control physical devices**
  - driver code has detailed knowledge of how to operate a particular device
  - implemented as functions that provide interface between specific capabilities of a device and what the operating system expects
  - loaded as part of OS as needed, e.g., when a device is plugged in
    ("Windows has found new hardware")
- **drivers insulate OS and application programs from specific properties of devices**

How does an operating system work?

- **loaded into RAM & started when machine is turned on ("boot")**
  - so it starts out being in charge / running on the bare hardware
- **gives control in turn to each program that is ready to run**
- **responds to external events / devices / …**
  - does actions, relays events to programs, …
- **programs (applications) request services by "making a system call"**
  - execute a particular instruction that transfers control to specific part of operating system
  - parameters say what task to do
- **OS does operation, returns control (and result) to application**

Virtual machines

- **running other OS's on top of an OS**
  - e.g., VMWare, Parallels, Xen, ...
- **system calls to "guest" OS are intercepted by "host" OS**
  - e.g., guest == Win XP or Linux, host == MacOSX
- **passed to guest OS, which handles by converting into system calls to host OS**
- **not the same as "dual boot"**



Bootstrapping: how does it all get started

- **CPU begins executing at specific memory location when turned on**
  - location is defined by the hardware: part of the machine's design
  - often in ROM (read-only memory) so not volatile but changeable

- **"bootstrap" instructions placed there read more instructions**
  - CPU tries to read first block from disk as bootstrap to copy more of the operating system
  - if that fails, tries to read bootstrap from somewhere else
    e.g., CD-ROM, USB, network, ...