OS Concepts and structure

Today

- OS services
- OS interface to programmers/users
- OS components & interconnects
- Structuring OSs

Next time

- Processes

OS Views

- Vantage points
  - OS as the services it provides
    - To users and applications
  - OS as its components and interactions
- OS provides a number of services
  - To users via a command interpreter/shell or GUI
  - To application programs via system calls
  - Some services are for convenience
    - Program execution, I/O operation, file system management, communication
  - Some to ensure efficient operation
    - Resource allocation, accounting, protection and security

Command interpreter (shell) & GUI

- Command interpreter
  - Handle (interpret and execute) user commands
  - Could be part of the OS: MS DOS, Apple II
  - Could be just a special program: UNIX, Win XP
    - In this way, multiple shells are possible
  - The command interpreter could
    - Implement all commands
    - Simply understand what program to invoke and how (UNIX)
- GUI
  - Friendlier (desktop), if sometimes limiting
  - Xerox PARK Alto >> Apple >> Windows >> Linux

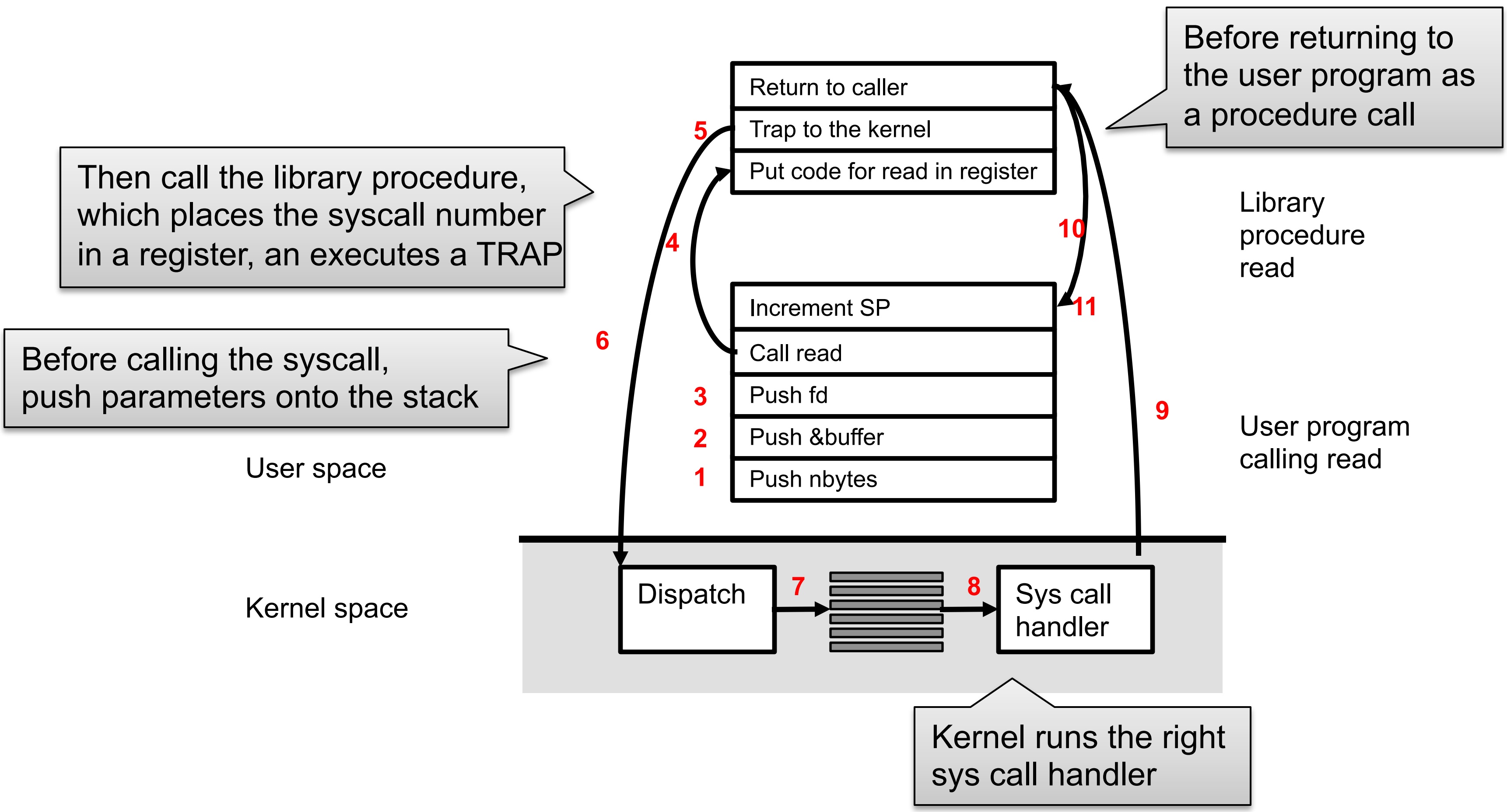
System calls

- Low-level interface to services for applications
- Higher-level requests get translated into sequence of system calls
- Writing `cp` – copy source to destination
  - Get file names
  - Open source
  - Create destination
  - Loop
    - Read from source
    - Copy to destination
  - Close destination
  - Report completion
  - Terminate

System calls

- The steps in making a read system call

```
count = read(fd, buffer, nbytes);
```



Major OS components & abstractions

- Processes
- Memory
- I/O
- Secondary storage
- File systems
- Protection
- Accounting
- Shells & GUI
- Networking

Processes

- A program in execution
  - Address space
  - Set of registers
  - Threads and processes – for now consider each process to have a single thread (we'll change that later)
  - Linux: `ps -auwwx` to list all processes

- To get a better sense of it
  - What data do you need to (re-) start a suspended process?
  - Where do you keep this data?
  - What is the process abstraction interface offered by the OS?
    - Create, delete, suspend, resume & clone a process
    - Inter-process communication & synchronization

Memory management

- Main memory – the directly accessed storage for CPU
  - Programs must be stored in memory to execute
  - Memory access is fast (e.g., 60 ns to load/store)
    - but memory doesn't survive power failures
- OS must
  - Allocate memory space to processes (explicitly and implicitly)
  - Decide how to allocate to each process
  - Deallocate space when needed by rest of system
  - Maintain mappings from physical to virtual memory
  - Decide when to remove a process from memory

I/O

- A big chunk of the OS kernel deals with I/O
  – Hundreds of thousands of lines in NT
- The OS provides a standard interface between programs & devices
  – file system (disk), sockets (network), frame buffer (video)
- Device drivers are the routines that interact with specific device types
  – Encapsulates device-specific knowledge
    - e.g., how to initialize a device, request I/O, handle errors
  – Examples: SCSI device drivers, Ethernet card drivers, video card drivers, sound card drivers, …

Secondary storage

- Secondary storage (disk, tape) is persistent memory
  - Often magnetic media, survives power failures (hopefully)
- Routines that interact with disks are typically at a very low level in the OS
  - Used by many components (file system, VM, …)
  - Handle scheduling of disk operations, head movement, error handling, and often management of space on disks
- Usually independent of file system
  - Although there may be cooperation
  - File system knowledge of device details can help optimize performance
    - e.g., place related files close together on disk

File systems

- Storage devices are hard to work with
  - File system offers a convenient abstraction
  - Defines logical abstractions/objects like files & directories
  - As well as operations on these objects
- A file is the basic unit of long-term storage
- A directory is just a special kind of file
  - … containing names of other files & metadata
- Interface
  - File/directory creation/deletion, manipulation, copy, lock
- Other higher level services: accounting & quotas, backup, indexing or search, versioning

Protection

- Protection is a general mechanism used throughout the OS
  - All resources must be protected
    - memory
    - processes
    - files
    - devices
    - …
- Protection mechanisms help to detect and contain errors, as well as preventing malicious destruction

And now a short break …

OS structure

- OS made of number of components
  - Process & memory management, file system, …
  - and system programs
    - e.g., bootstrap code, the init program, …
- Major design issue
  - How do we organize all this?
  - What are the modules, and where do they exist?
  - How do they interact?
- Massive software engineering
  - Design a large, complex program that:
    - performs well, is reliable, extensible, backwards compatible, …

OS design & implementation

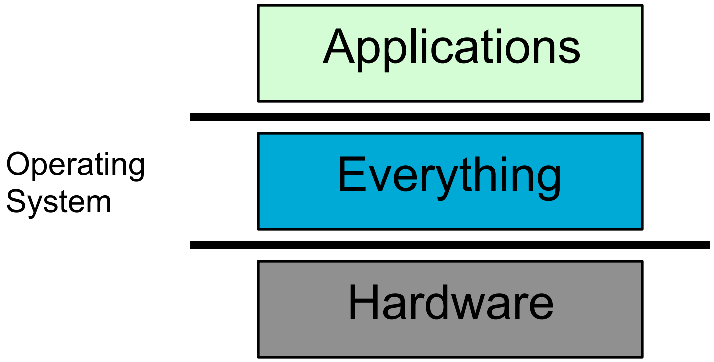
- *User* goals and *System* goals
  - User – convenient to use, easy to learn, reliable, safe, fast
  - System – easy to design, implement, & maintain, also flexible, reliable, error-free & efficient
- Affected by choice of hardware, type of system
- Clearly conflicting goals, no unique solution
- Some other issues complicating this
  - Size: Windows XP ~40G SLOC, RH 7.1 17G SLOC
  - Concurrency – multiple users and multiple devices
  - Potentially hostile users, but some users want to collaborate
  - Long expected lives & no clear ideas on future needs
  - Portability and support to thousands of device drivers
  - Backward compatibility

OS design & implementation

- A software engineering principle – separate policy & mechanism
  - Policy: What will be done?
  - Mechanism: How to do it?
  - Why do you care? Max flexibility, easier to change
- Implementation on high-level language
  - Early on – assembly (e.g. MS-DOS – 8088), later Algol (MCP), PL/1 (MULTICS), C (Unix, …)
  - Advantages – faster to write, more compact, easier to maintain & debug, easier to port
  - Cost – Size, speed?, but who cares?!

> Early versions … were written in assembly language, but during the summer of 1973, it was rewritten in C. The size of the new system is about one third greater than the old. … much easier to understand and to modify but also includes many functional improvements … we considered this increase in size quite acceptable.
>
> *D. Ritchie and K. Thompson,* The UNIX time-sharing system*, CACM 17(7),1974*

Monolithic design



- Major advantage
  - Cost of module interactions is low (procedure call)
- Disadvantages
  - Hard to understand
  - Hard to modify & maintain
  - Unreliable (no isolation between system modules)
- Alternative?
  - How to organize the OS in order to simplify design, implementation and maintenance?

Layering

- The traditional approach
  - Implement OS as a set of layers
  - Each layer shows an enhanced ‘virtual mach’ to layer above

Layer	Function
5	The operator
4	User programs
3	I/O management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Dijkstra’s THE system

- Each layer can be tested and verified independently

Problems with layering

- Imposes hierarchical structure
  - but real systems have complex interactions
  - Strict layering isn’t flexible enough
- Poor performance
  - Each layer crossing has an associated overhead
- Disjunction between model and reality
  - Systems modelled as layers, but not built that way

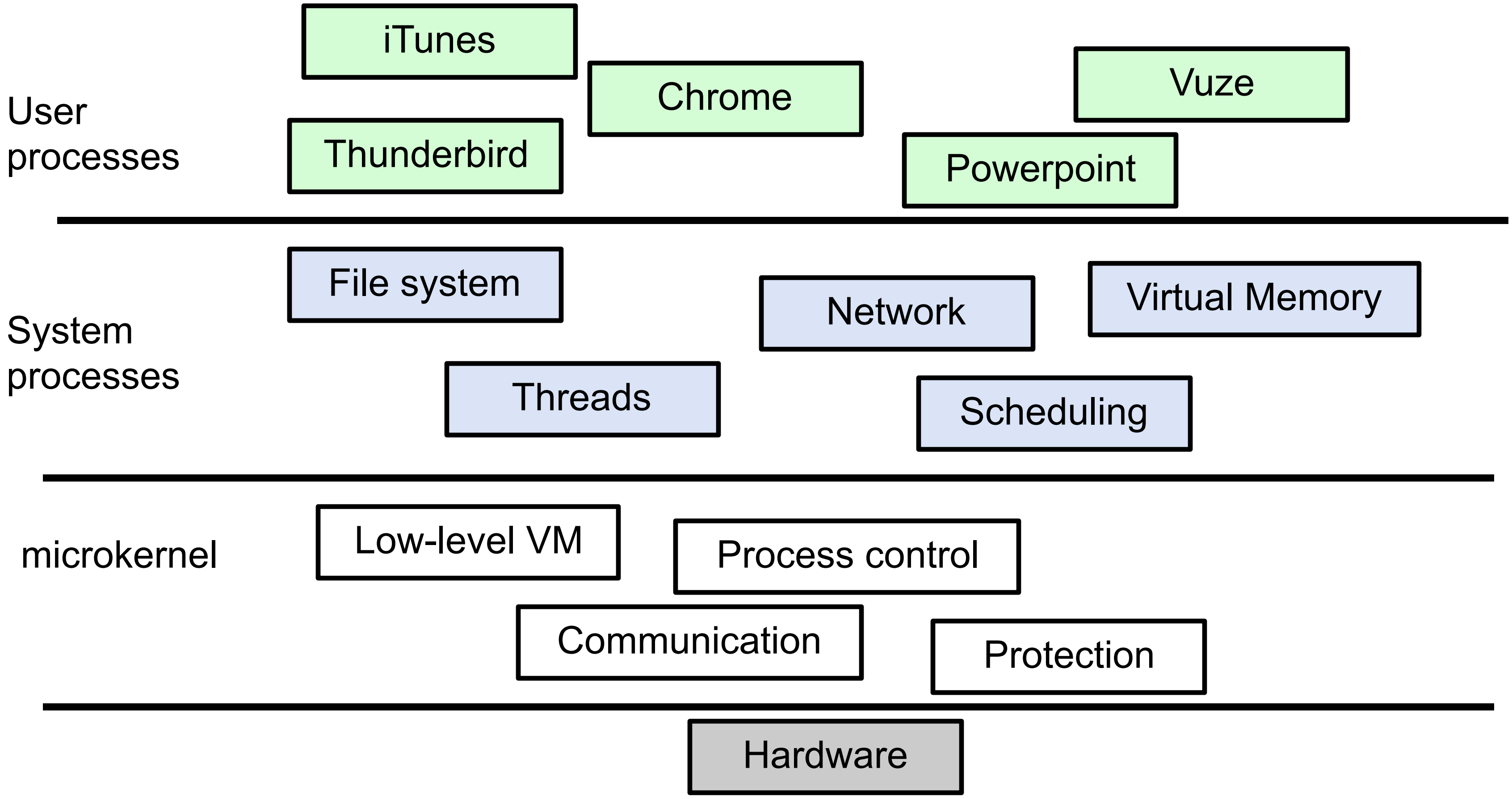
HAL – Hardware Abstraction Layer

- An example of layering in modern OSs
- Goal – to hide differences in hardware from most of the OS kernel
  - On a PC, you can consider it as the driver of the motherboard
  - BSD, Mac OS X, Windows NT, Linux, NetBSD all use a HAL either explicitly identified or not

Microkernels

- Popular in the late 80’s, early 90’s
  - Recent resurgence
- Goal
  - Minimize what goes in kernel
  - Organize rest of OS as user-level processes
- This results in
  - Better reliability (isolation between components)
  - Ease of extension and customization
  - Poor performance (user/kernel boundary crossings)
- First microkernel Hydra (CMU, 1970)
  - … Mach (CMU), Chorus (UNIX-like), OS X (Apple), in some ways NT (Microsoft), L4 (Karlsruhe), MINIX 3, …

Microkernel

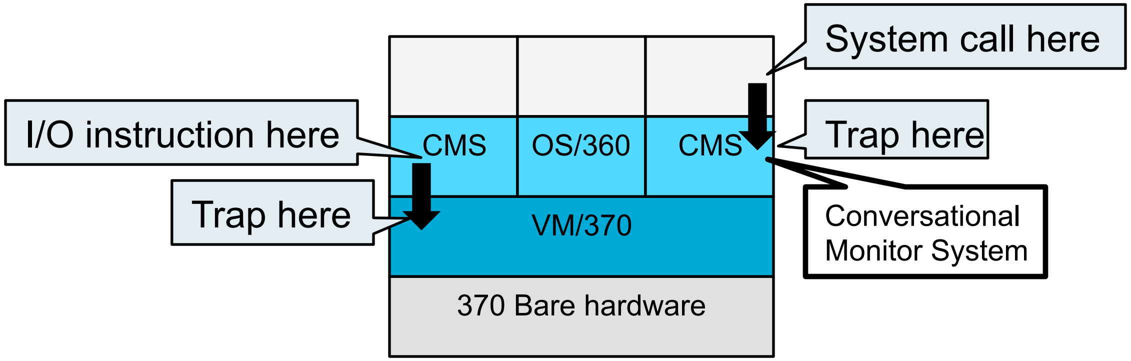


Virtual machines

- Initial release of OS/360 were strictly batch but users wanted timesharing
  – IBM CP/CMS, later renamed VM/370 (‘79)
- Note that timesharing systems provides
  1. Multiprogramming
  2. Extended (virtual) machine
- Essence of VM/370 – separate the two
  – Heart of the system (VMM) does multiprogramming & provides multiple exact copies of bare HW to next layer up
  – Each VM can run any OS

Virtual machines

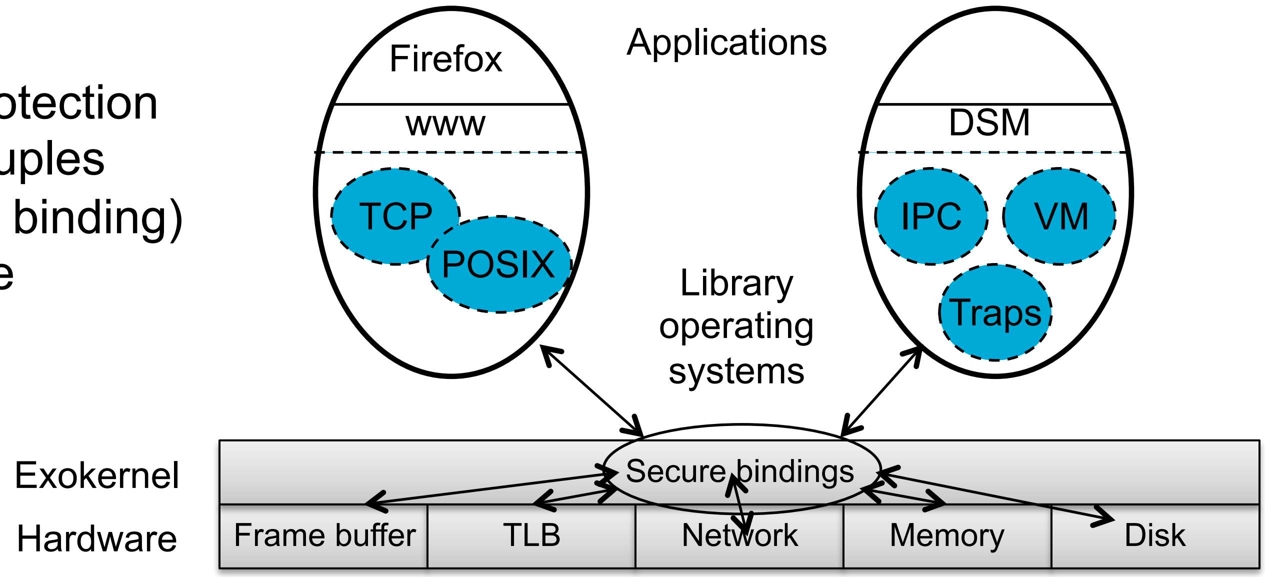
- A resurgence in mid-90s, started with Rosenblum’s work on Disco and VMWare
  - Nowadays … Java VM, Xen, VritulaBox, Virtual Iron, VMLite, Simics, Parallels, Palacios, QEMU, …
- What for?
  - Server consolidation – from different services in different lightly used machine to consolidation (administration cost)
  - Different applications for other OS in your desktop
  - Testing and debugging



Exokernels

- OS, typically securely multiplexes & *abstract* physical resources
- But no OS abstractions fits all!
- Exokernel
  - A minimal OS securely multiplexes resources
  - Library OSes implement higher-level abstractions

Secure binding – a protection mechanism that decouples authorization (done at binding) from use of a resource



Summary & preview

- Today
  – The mess under the carpet
  – Basic concepts in OS
  – OS design has been an evolutionary process
  – Structuring OS - a few alternatives, not a clear winner
- Next …
  – Process – the central concept in OS
    • Process model and implementation
    • What it is, what it does and how it does it