CMPE 220

Class 20 – Case Studies

Case Study: The Linux Operating System

- Linux as a case study of operating systems should be like a semester review.
  - We've pretty much been looking at Linux all along.

- Designed by programmers for programmers, to be:
  - Simple
  - Elegant
  - Consistent
  - Powerful
  - Flexible
  - Open Source!

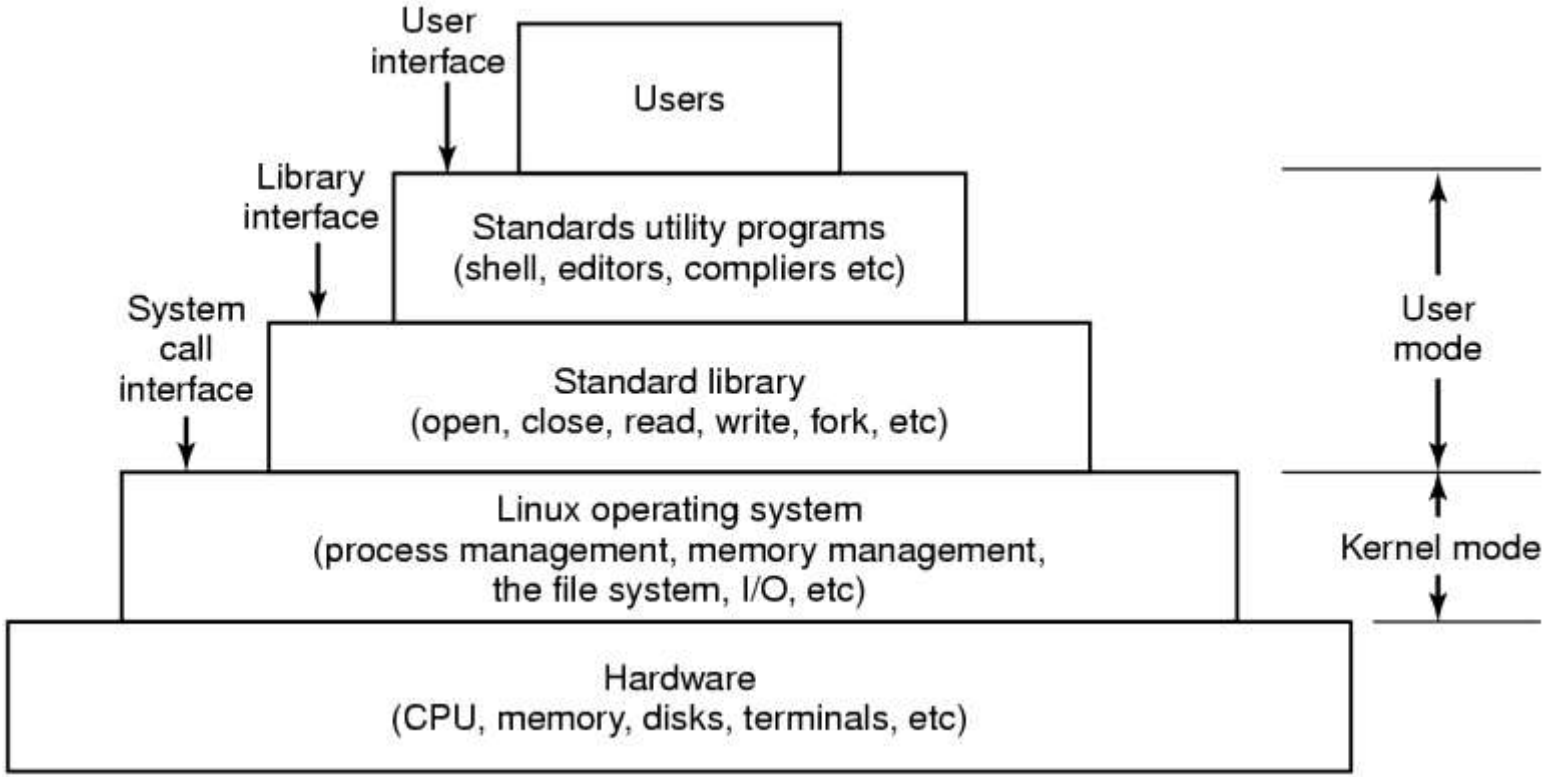
Linux versus POSIX

- Linux, computer operating system created in the early 1990s by Finnish software engineer Linus Torvalds and the Free Software Foundation (FSF).
- Linux is a “reverse engineered” version of Unix.

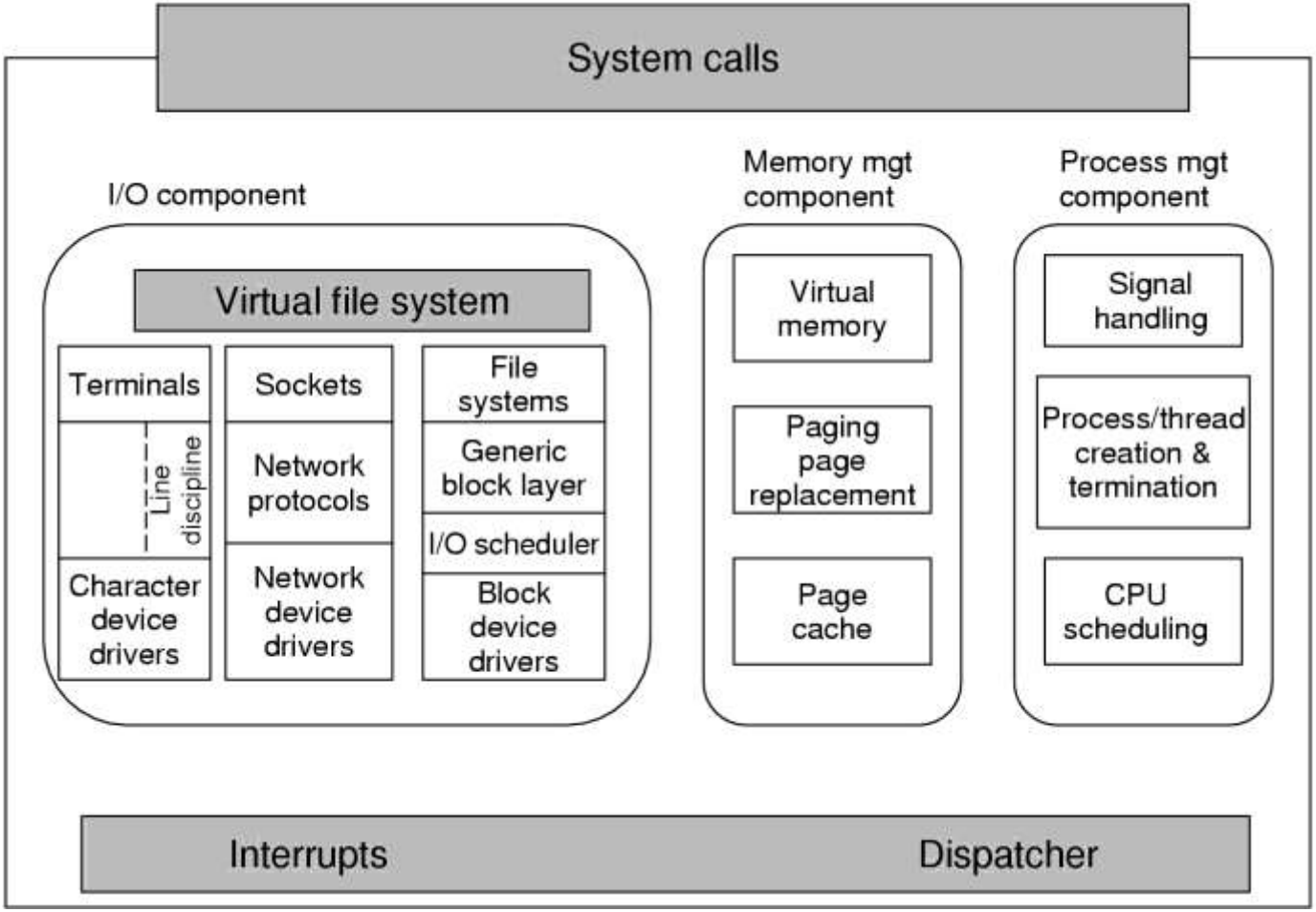
Linux versus POSIX

- POSIX stands for Portable Operating System Interface. It's a family of standards specified by IEEE for maintaining compatibility among operating systems.
- Developed in the mid-to-late 1990s.
- Any software that conforms to POSIX standards should be compatible with other operating systems that adhere to the POSIX standards.
- Linux distros: Centos, Debian, Ubuntu, Fedora…
- MacOS is POSIX compliant

The Layers in a Linux System



Structure of the Linux Kernel



Linux Processes and Threads

- Linux tasks represent processes and threads.

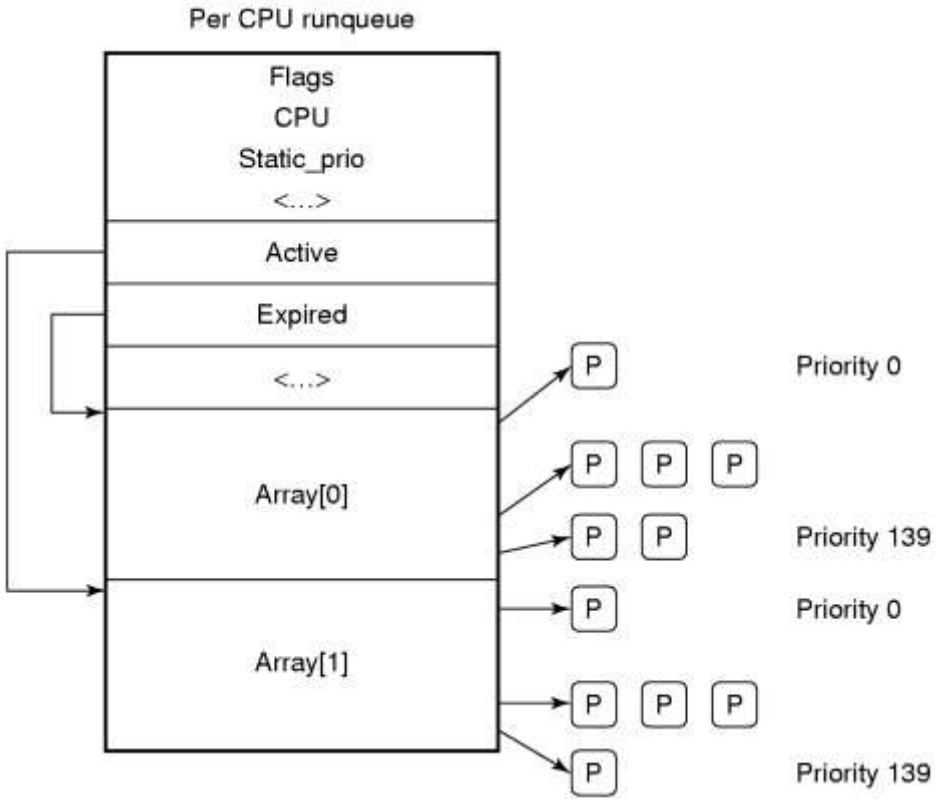
- Same internal process descriptor (PID):
  - Scheduling parameters (e.g. I/O wait, block)
  - Memory image
  - Signals & semaphors
  - Machine registers
  - System call state
  - File descriptor table
  - Accounting
  - Kernel stack
  - Miscellaneous

Linux Scheduler

- Scheduling
  - FCFS and round-robin, as required by POSIX.
  - Priority-based
    - Priority ranges 0-99.
  - Pre-emptive Completely Fair Scheduler (CFS).
    - CFS always schedules the task which has the least amount of time left on the CPU.
  - **`nice`** command and system call can adjust a process's priority up or down.

Linux Scheduler, *cont'd*

- The runqueue is associated with *each CPU*.
  - Two arrays, active and expired.
  - Each contains 100 list heads, one per priority level.
  - Each list head points to a list of processes at that priority level.



Linux ext3 File System

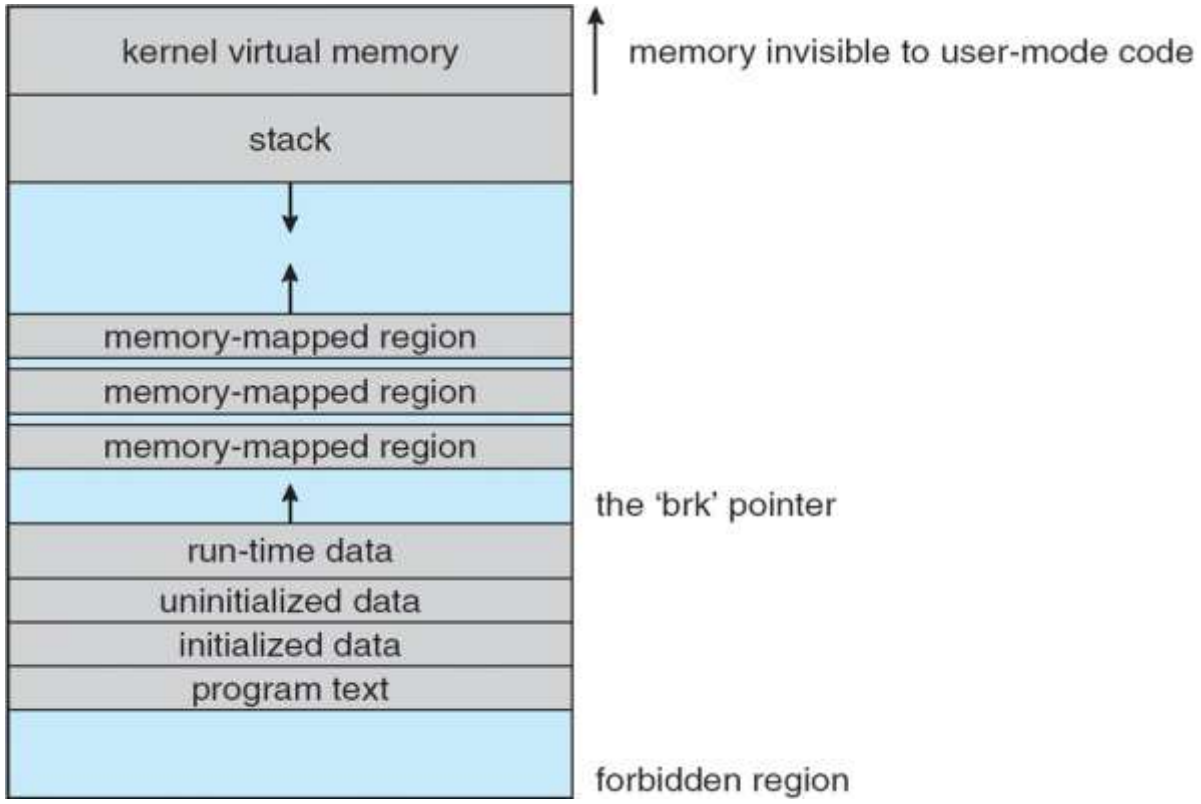
- Default block size is a function of the total size of the file system.
  - Block sizes of 1, 2, 4, and 8 KB supported.

- To maintain high performance, *cluster* physically adjacent I/O requests.

- Journaling file system
  - Transactions are logged to a journal.
  - Journal entries are replayed to the actual file system.
  - Journaling may improve file system performance.

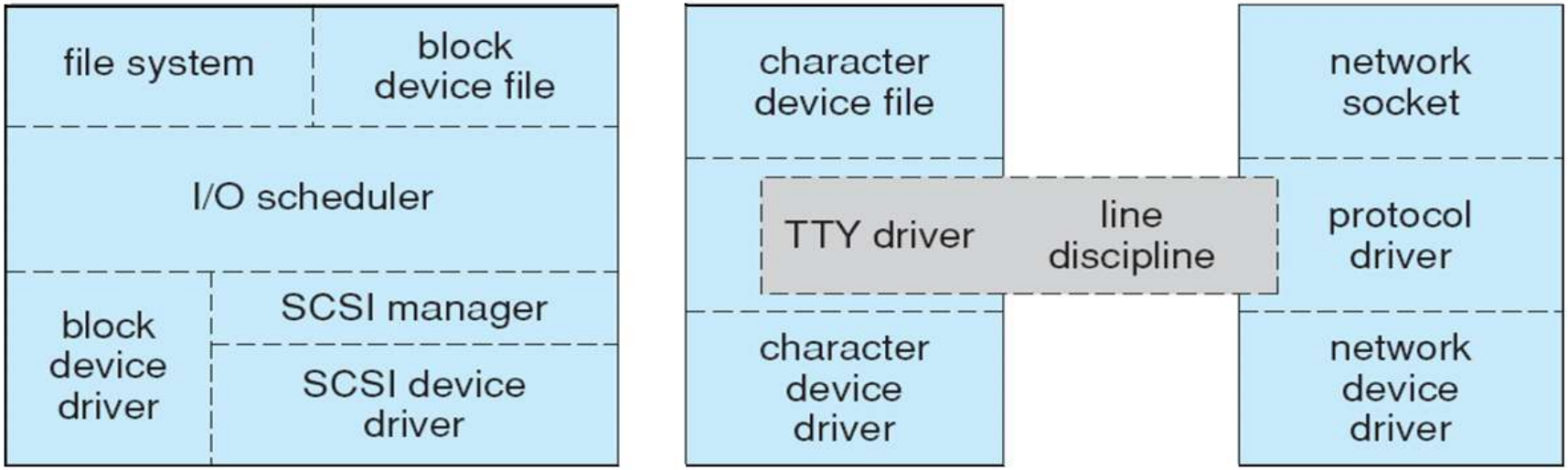
Linux Executable Files

- Old format: **`a.out`**
- New format: ELF (Executable and Linkable Format)
  - Memory layout:



Linux I/O

- Device-driver overall structure.



Block devices | Character devices | Network devices

Operating System Concepts, 10th edition
by Abraham Silberschatz, Greg Gagne, and Peter B. Galvin
Wiley, 2018, ISBN 978-1119456339

Linux I/O, *cont'd*

- Block devices
  - Traditionally used C-SCAN (Circular Elevator) disk scheduling
    across all processes.
  - New algorithm: Completely Fair Queuing (CFQ)
    - One request list per process.
    - Each list maintained according to C-SCAN.
    - Lists are serviced round-robin.
- Character devices
  - Line discipline interprets terminal device data.
- Network devices

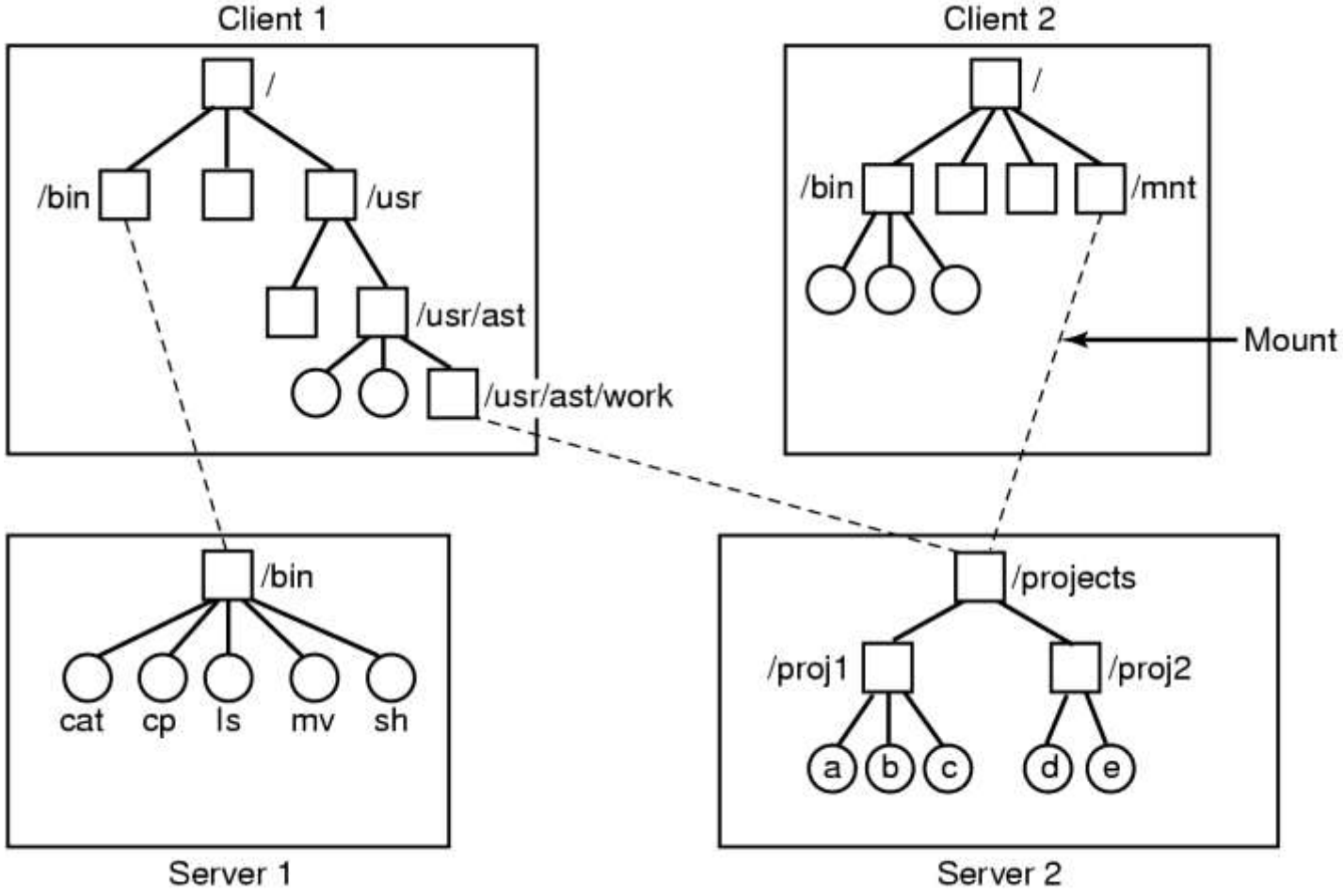
Linux Networking

- Supports standard Internet protocols
  - UNIX-to-UNIX
  - Non-UNIX operating systems (standards!)
- Three networking layers:
  - Socket interface (a socket is a combination of IP address and port)
  - Protocol drivers
  - Network-device drivers
- User applications perform all networking requests through the socket interface

TCP/IP Protocol Suite

- IP protocol implements routing between hosts.
  - UDP, TCP, and other protocols implemented on top of the routing protocol
- UDP
  - Arbitrary individual datagrams between hosts (stateless)
- TCP
  - Reliable connections between hosts
  - Guaranteed delivery of packets in order
  - Automatic retransmissions of lost packets

NFS (Network File System) Mounting



Sun Microsystems – mid 1980s

Microsoft Windows 10

- Extensible and portable operating system.
- Object Oriented: Kernel objects provide basic services.
  - Virtual memory, caching, preemptive scheduling
- Elaborate security mechanisms to protect user data and to guarantee program integrity
- Internationalization features
- Sophisticated scheduling and memory management features
- Feature-rich and expansive API programming environment for developers
- *Why is windows disparaged?*

Why is Windows Disparaged?

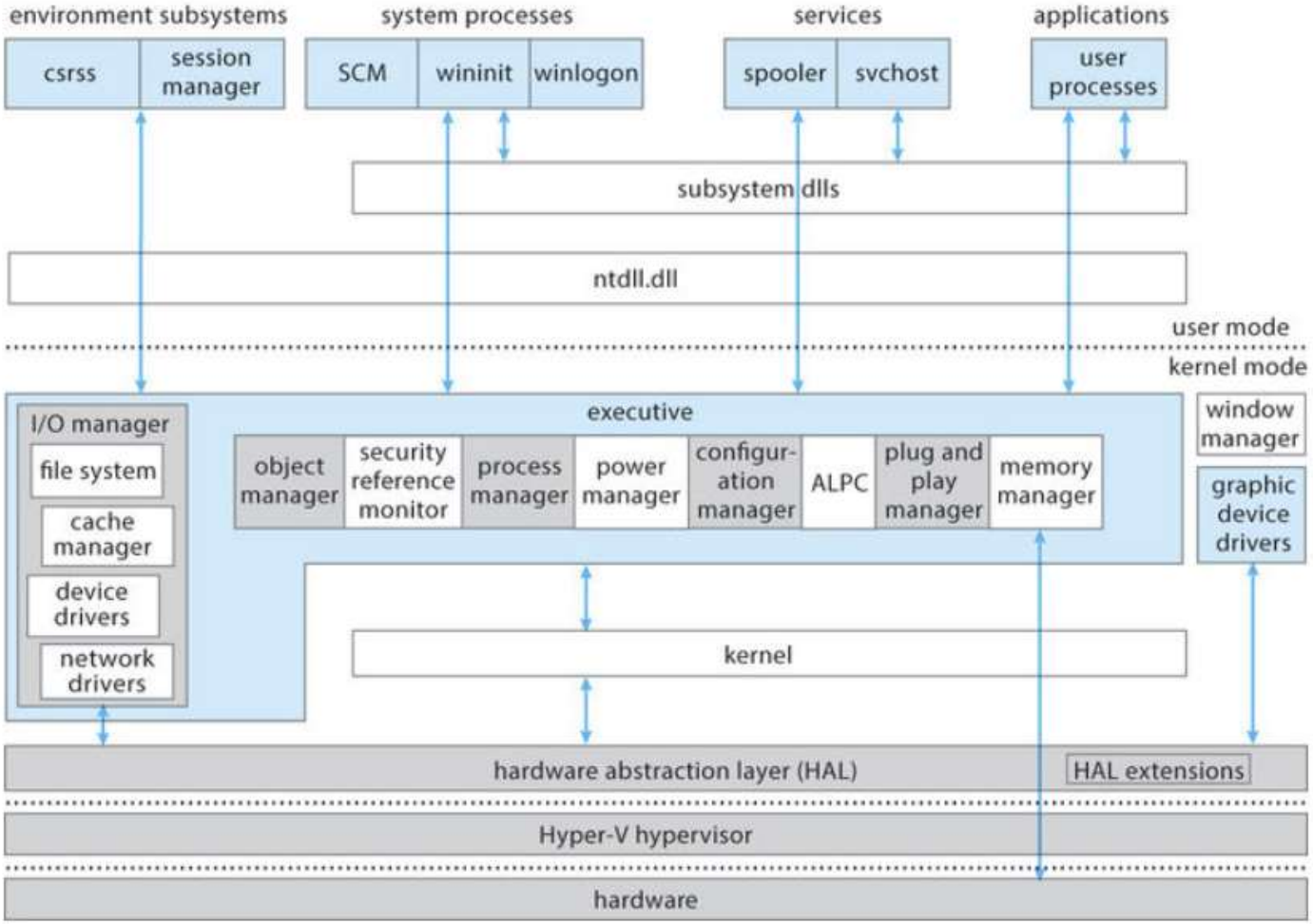
- History:
  - Microsoft MS-DOS and “The IBM PC”
- Microsoft did not play well with others
  - Not standards compliant
- Bill Gates was a jerk
- Continuing issues with “being different”

Windows vs. Linux

- Linux
  - Simple OS functions
  - Few parameters
  - Few examples of multiple ways to do something.
  - Kernel panic for a fatal error.

- Windows
  - Comprehensive APIs with many parameters.
  - Several ways to do the same thing.
  - Mixed low-level and high-level functions.
  - Blue Screen of Death for a fatal error.

Windows 10 Architecture



Windows 10 Dispatcher Objects

- Dispatcher objects control dispatching and synchronization in the system:
  - **event**: records an event occurrence and synchronizes the event with some action
  - **mutex**: enforces mutual exclusion
  - **semaphore**: synchronizes threads
  - **thread**: scheduled by the kernel dispatcher
  - **timer**: keeps track of time and signals timeouts

Processes and Threads, *cont'd*

Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SwitchToFiber	Run a different fiber on the current thread
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled, then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section
WaitOnAddress	Block until the memory is changed at the specified address
WakeByAddressSingle	Wake the first thread that is waiting on this address
WakeByAddressAll	Wake all threads that are waiting on this address
InitOnceExecuteOnce	Ensure that an initialize routine executes only once

Modern Operating Systems, 4th edition
by Andrew Tanenbaum & Herbert Bos
Pearson, 2014, ISBN 978-0133591620

Interrupt Request Levels (IRQLs)

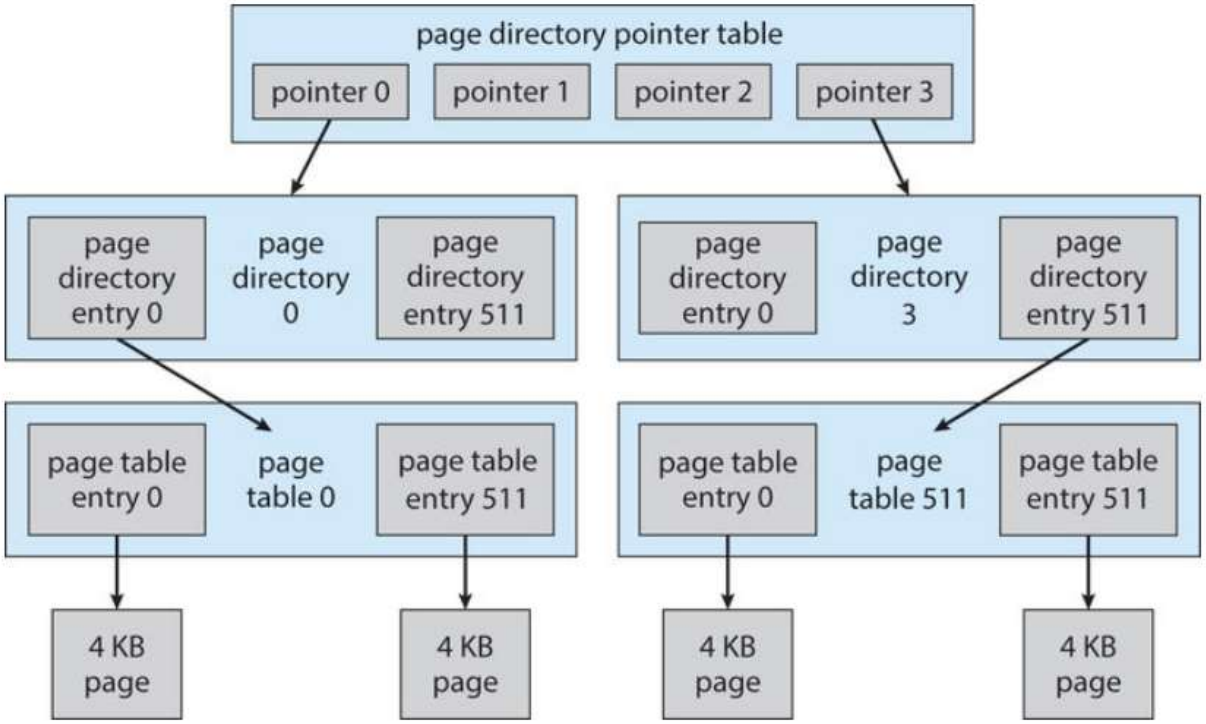
interrupt levels	types of interrupts
31	machine check or bus error
30	power fail
29	interprocessor notification (request another processor to act; e.g., dispatch a process or update the TLB)
28	clock (used to keep track of time)
27	profile
3–26	traditional PC IRQ hardware interrupts
2	dispatch and deferred procedure call (DPC) (kernel)
1	asynchronous procedure call (APC)
0	passive

Exceptions and Interrupts

- The kernel dispatcher provides trap handling for exceptions and interrupts generated by hardware or software.
  - Integer or floating-point overflow
  - Integer or floating-point divide by zero
  - Illegal instruction
  - Data misalignment
  - Privileged instruction
  - Access violation
  - Paging file quota exceeded
  - Debugger breakpoint

Virtual Memory

- The page table layout can be expanded to manage ever larger physical memory sizes



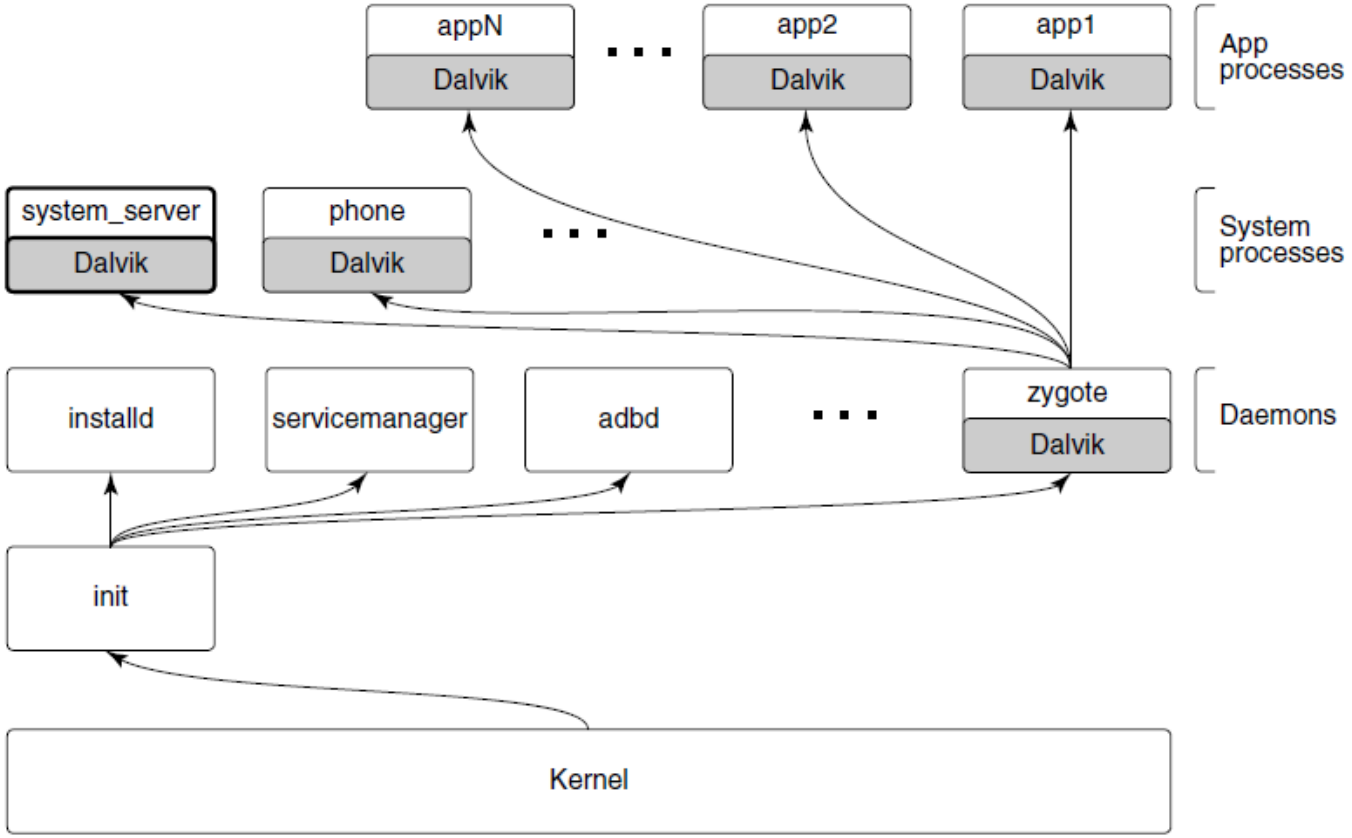
Android

- Designed by Google to run on mobile devices.
  - NASA uses Android for its “cubesats”.
- Based on the Linux kernel and facilities.
  - Much written in Java.
- Combines open-source code with closed-source third-party applications.
  - Supports a wide variety of proprietary cloud services.
  - Google Play online store for Android apps.

Android Design Goals

- Open-source platform for mobile devices
- Support 3rd party apps with robust, stable API
- 3rd party apps compete on level playing field
- Users need not deeply trust 3rd party apps
- Support mobile user interaction
- Manage app processes for users
- Encourage apps to interoperate, collaborate
- Full general-purpose OS

Android Architecture



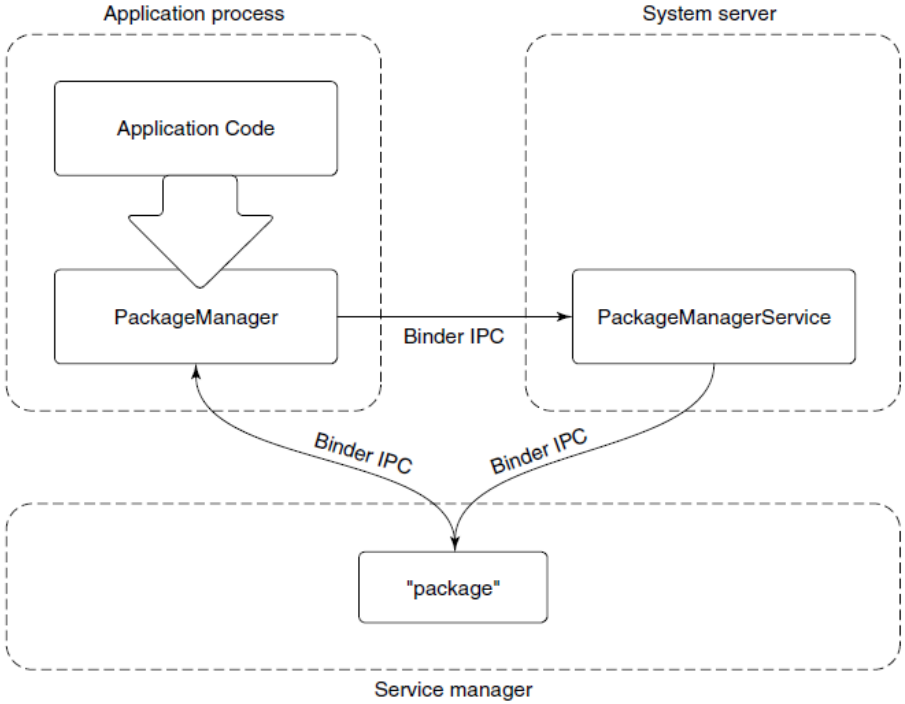
Dalvik is a JVM

Android Architecture, *cont'd*

- The init process starts Android daemons.
  - Focused on low-level functions such as managing file systems and hardware access.

- Additional layer of processes that run Dalvik's Java language environment.
  - Dalvik is a managed runtime environment.
  - Brought up by the zygote native daemon.
  - JVM bytecode is translated to Dalvik bytecode.

Android Application Framework



- Typical design for Android framework APIs that interact with system services, such as the **`PackageManager`** class.

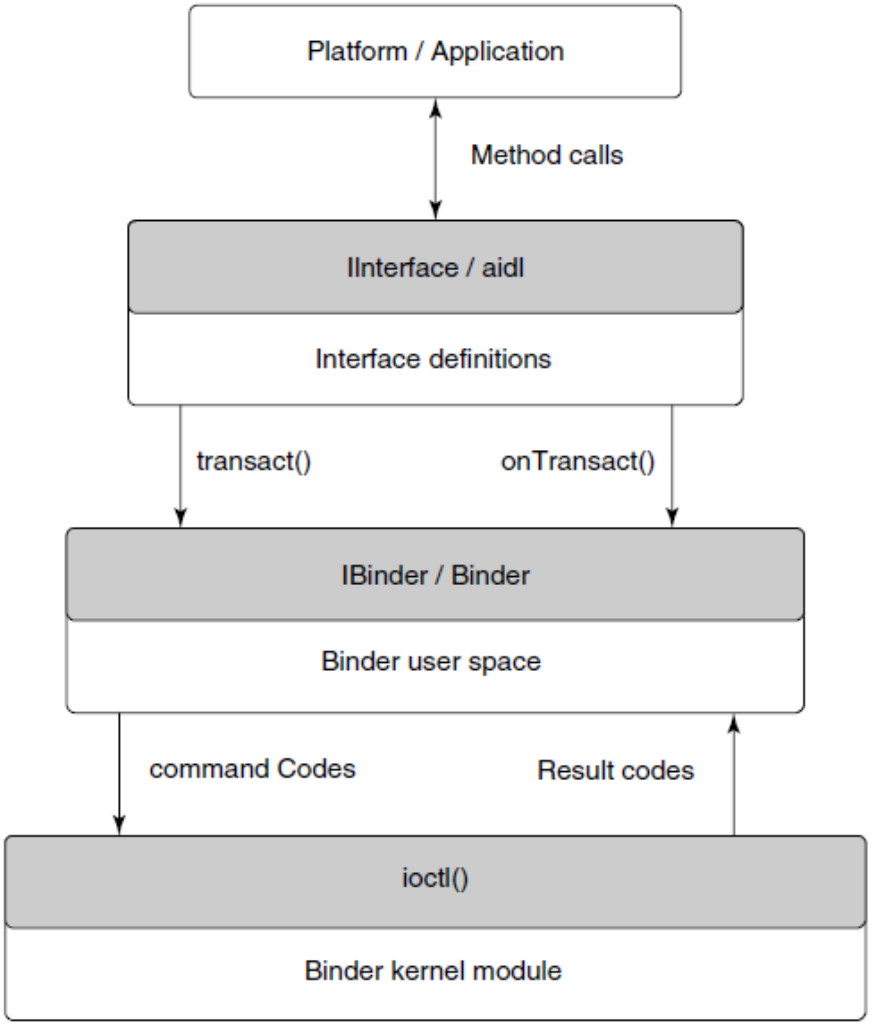
Package Manager Example

- The package manager provides a framework API for apps to call in their local process.
  - The **`PackageManager`** class connects to the corresponding service in the system server.
  - The **`PackageManager`** makes calls on the service.

- Calls are implemented as interprocess communication.
  - Android Binder IPC mechanism.

Binder IPC

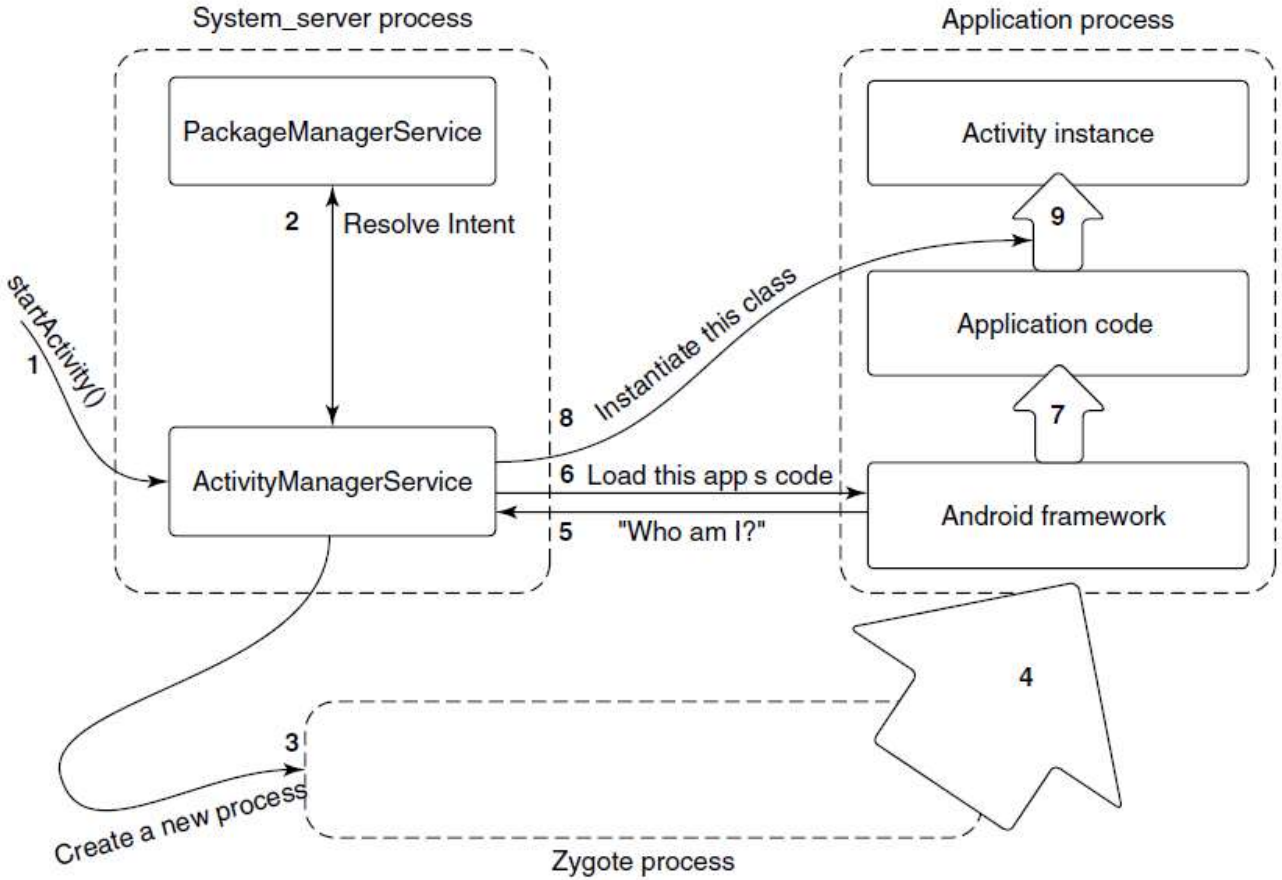
- Relies on remote procedure calls (RPC).



Modern Operating Systems, 4th edition
by Andrew Tanenbaum & Herbert Bos
Pearson, 2014, ISBN 978-0133591620

San José State University

Launching a New Application Process



Process Lifecycle

Category	Description	oom_adj
SYSTEM	The system and daemon processes	–16
PERSISTENT	Always-running application processes	–12
FOREGROUND	Currently interacting with user	0
VISIBLE	Visible to user	1
PERCEPTIBLE	Something the user is aware of	2
SERVICE	Running background services	3
HOME	The home/launcher process	4
CACHED	Processes not in use	5

Process importance categories.

oom_adj: a strict ordering to determine which processes to kill first in an out-of-memory situation.