

Operating System Structures

Week 1

S. Pradono Suryodiningrat

Session Learning Outcomes

Upon completion of this session, students are expected to be able to :

- Classify Operating System techniques according to its principles.

Objectives

- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

What Does the Term Operating System Mean?

- An operating system is “fill in the blanks”
- What about:
 - Car
 - Airplane
 - Printer
 - Washing Machine
 - Toaster
 - Compiler
 - Etc.

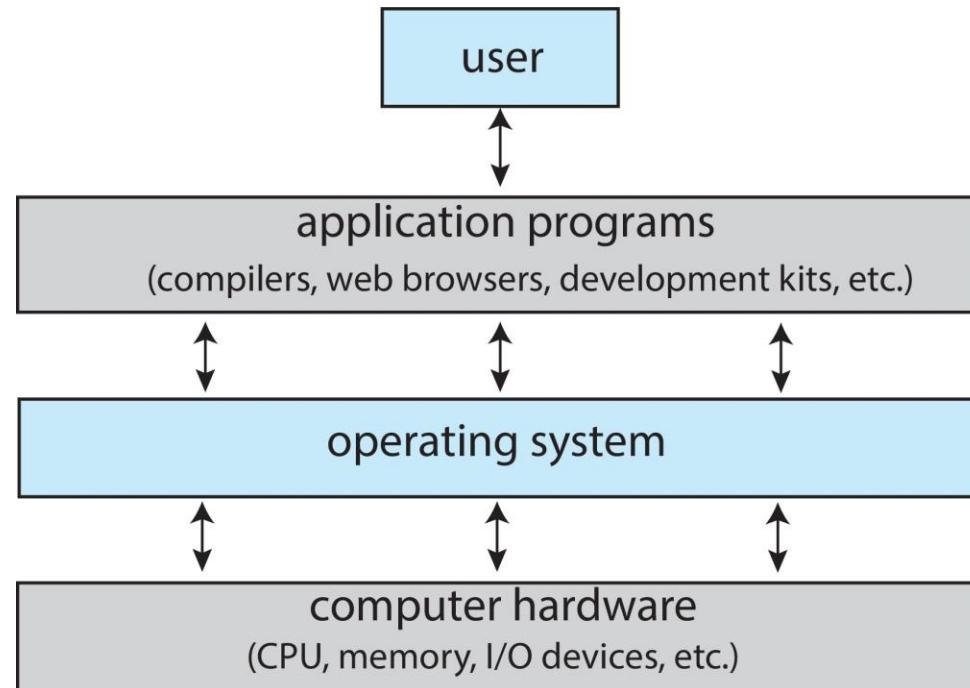
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
 - Execute user programs and make solving user problems easier
 - Make the computer system convenient to use
 - Use the computer hardware in an efficient manner

Computer System Structure

- Computer system can be divided into four components:
 - Hardware – provides basic computing resources
 - CPU, memory, I/O devices
 - Operating system
 - Controls and coordinates use of hardware among various applications and users
 - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
 - Word processors, compilers, web browsers, database systems, video games
 - Users
 - People, machines, other computers

Abstract View of Components of Computer



What Operating Systems Do

- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
 - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
 - Operating system is a **resource allocator** and **control program** making efficient use of HW and managing execution of user programs
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Mobile devices like smartphones and tables are resource poor, optimized for usability and battery life
 - Mobile user interfaces such as touch screens, voice recognition
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
 - Run primarily without user intervention

Defining Operating Systems

- Term OS covers many roles
 - Because of myriad designs and uses of OSes
 - Present in toasters through ships, spacecraft, game machines, TVs and industrial control systems
 - Born when fixed use computers for military became more general purpose and needed resource management and program control

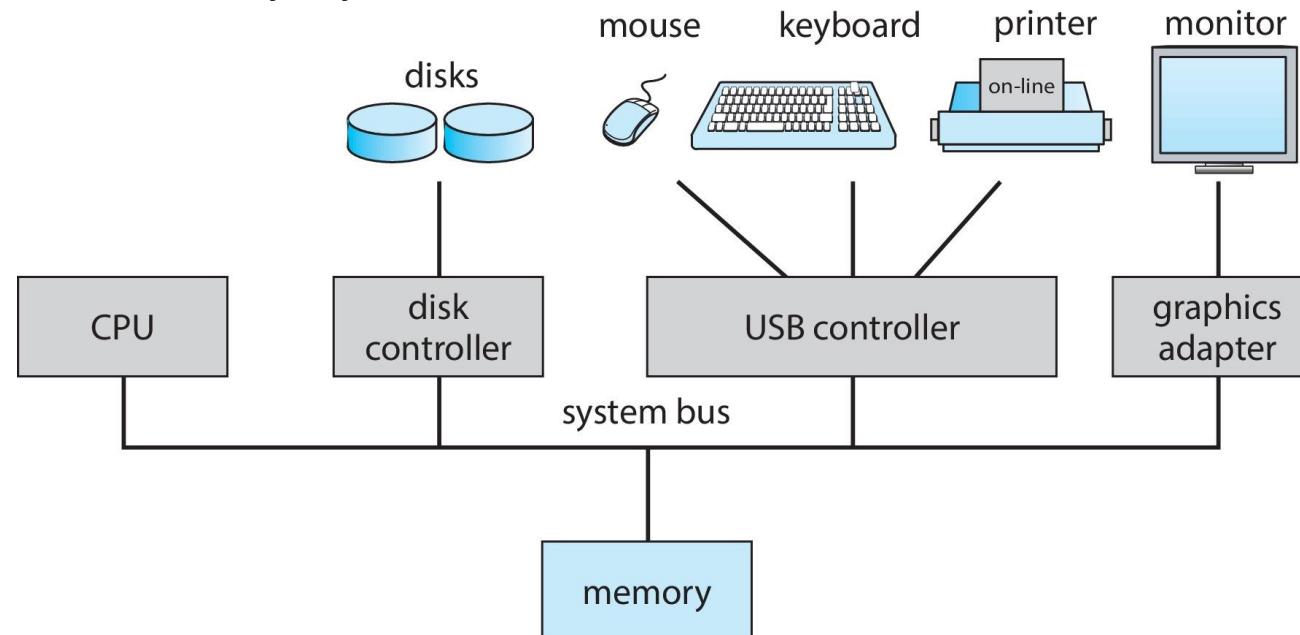
Operating System Definition

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
 - But varies wildly
- “The one program running at all times on the computer” is the **kernel**, part of the operating system
- Everything else is either
 - A **system program** (ships with the operating system, but not part of the kernel), or
 - An **application program**, all programs not associated with the operating system
- Today’s OSes for general purpose and mobile computing also include **middleware** – a set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics

Overview of Computer System Structure

Computer System Organization

- Computer-system operation
 - One or more CPUs, device controllers connect through common **bus** providing access to shared memory
 - Concurrent execution of CPUs and devices competing for memory cycles



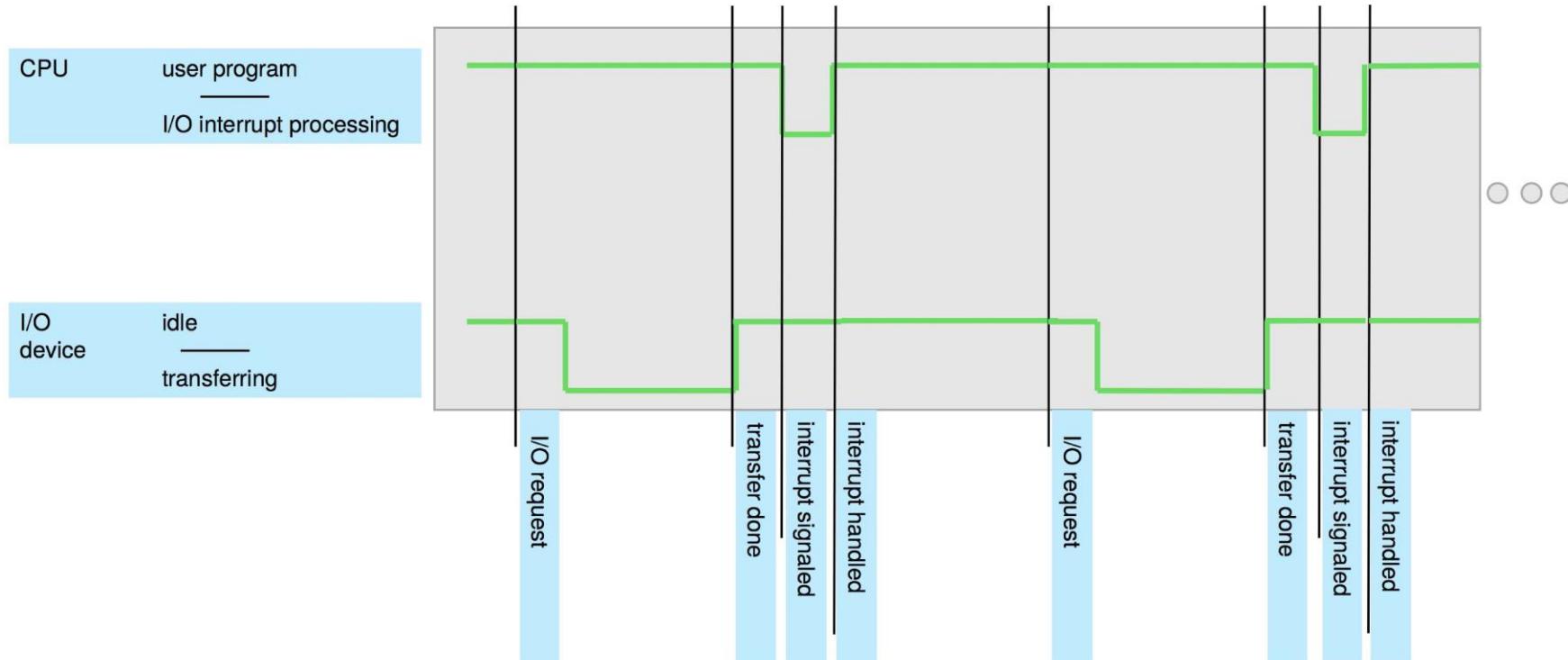
Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

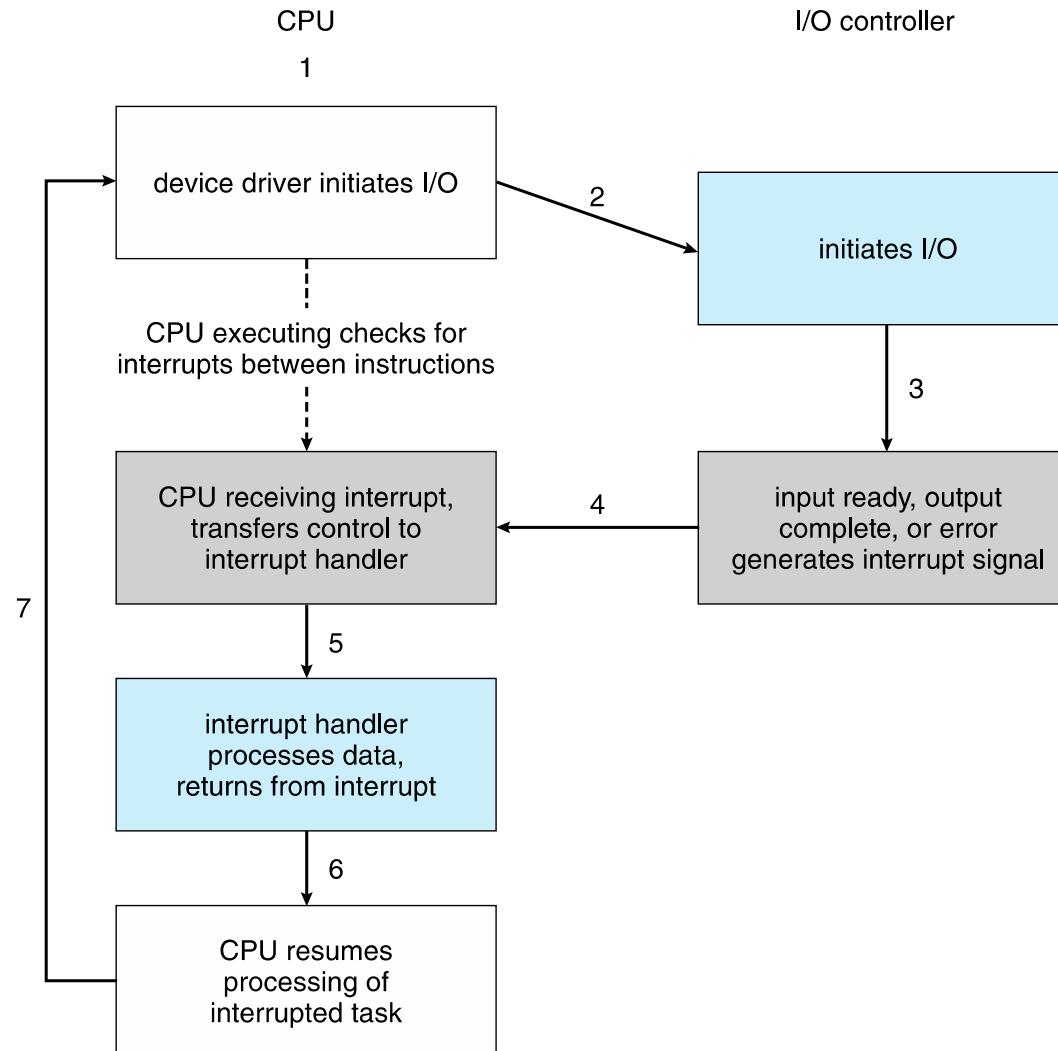
Interrupt Timeline



Interrupt Handling

- The operating system preserves the state of the CPU by storing the registers and the program counter
- Determines which type of interrupt has occurred:
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt-drive I/O Cycle



I/O Structure

- Two methods for handling I/O
 - After I/O starts, control returns to user program only upon I/O completion
 - After I/O starts, control returns to user program without waiting for I/O completion

I/O Structure (Cont.)

- After I/O starts, control returns to user program only upon I/O completion
 - Wait instruction idles the CPU until the next interrupt
 - Wait loop (contention for memory access)
 - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- After I/O starts, control returns to user program without waiting for I/O completion
 - **System call** – request to the OS to allow user to wait for I/O completion
 - **Device-status table** contains entry for each I/O device indicating its type, address, and state
 - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

Storage Structure

Storage Structure

- Main memory – only large storage media that the CPU can access directly
 - Random access
 - Typically volatile
 - Typically random-access memory in the form of **Dynamic Random-access Memory (DRAM)**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity

Storage Structure (Cont.)

- **Hard Disk Drives (HDD)** – rigid metal or glass platters covered with magnetic recording material
 - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
 - The **disk controller** determines the logical interaction between the device and the computer
- **Non-volatile memory (NVM)** devices – faster than hard disks, nonvolatile
 - Various technologies
 - Becoming more popular as capacity and performance increases, price drops

Storage Definitions and Notation Review

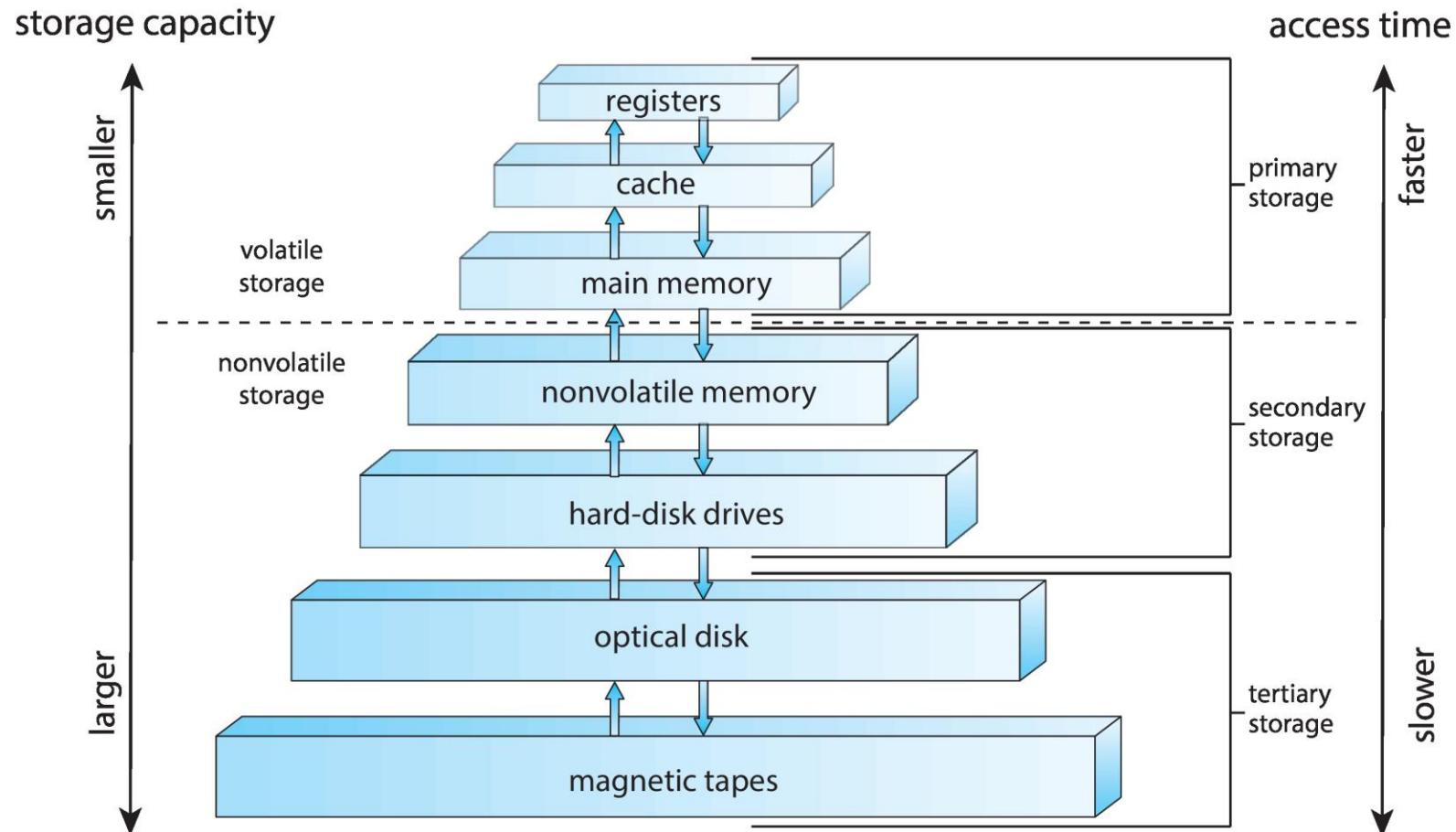
The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes. A **kilobyte**, or KB, is 1,024 bytes; a **megabyte**, or MB, is $1,024^2$ bytes; a **gigabyte**, or GB, is $1,024^3$ bytes; a **terabyte**, or TB, is $1,024^4$ bytes; and a **petabyte**, or PB, is $1,024^5$ bytes. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

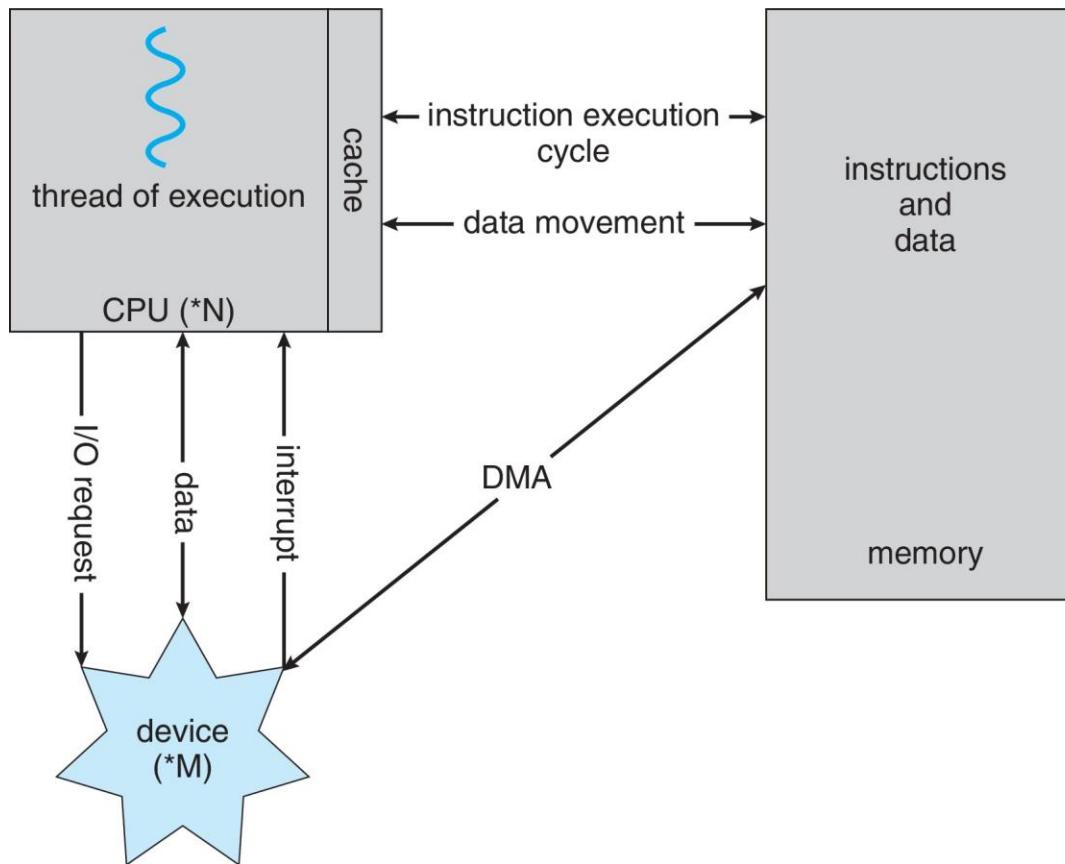
Storage Hierarchy

- Storage systems organized in hierarchy
 - Speed
 - Cost
 - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
 - Provides uniform interface between controller and kernel

Storage-Device Hierarchy



How a Modern Computer Works



A von Neumann architecture

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Operating-System Operations

- Bootstrap program – simple code to initialize the system, load the kernel
- Kernel loads
- Starts **system daemons** (services provided outside of the kernel)
- Kernel **interrupt driven** (hardware and software)
 - Hardware interrupt by one of the devices
 - Software interrupt (**exception** or **trap**):
 - Software error (e.g., division by zero)
 - Request for operating system service – **system call**
 - Other process problems include infinite loop, processes modifying each other or the operating system

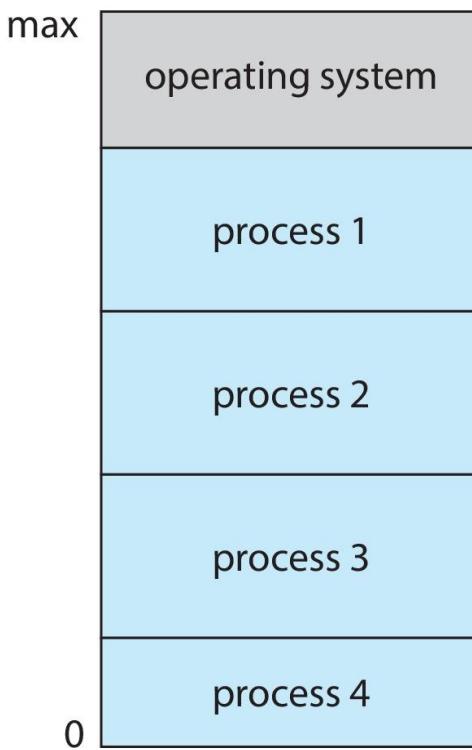
Multiprogramming (Batch system)

- Single user cannot always keep CPU and I/O devices busy
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When job has to wait (for I/O for example), OS switches to another job

Multitasking (Timesharing)

- A logical extension of Batch systems— the CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
 - **Response time** should be < 1 second
 - Each user has at least one program executing in memory ⇒ **process**
 - If several jobs ready to run at the same time ⇒ **CPU scheduling**
 - If processes don't fit in memory, **swapping** moves them in and out to run
 - **Virtual memory** allows execution of processes not completely in memory

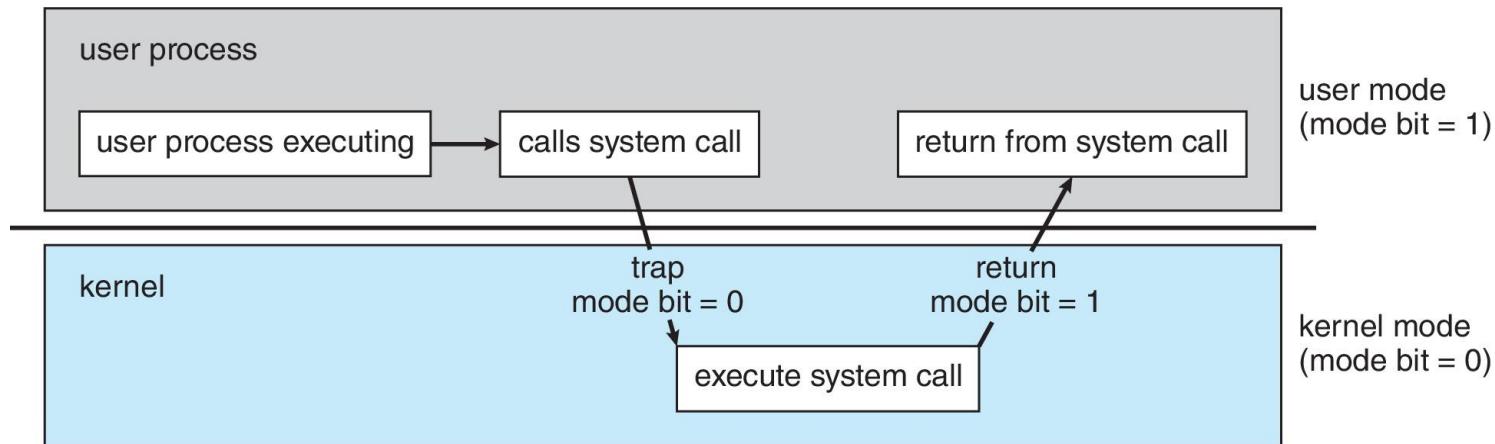
Memory Layout for Multiprogrammed System



Dual-mode Operation

- Dual-mode operation allows OS to protect itself and other system components
 - User mode and kernel mode
- Mode bit provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code.
 - When a user is running \Rightarrow mode bit is “user”
 - When kernel code is executing \Rightarrow mode bit is “kernel”
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
 - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as privileged, only executable in kernel mode

Transition from User to Kernel Mode



Timer

- Timer to prevent infinite loop (or process hogging resources)
 - Timer is set to interrupt the computer after some time period
 - Keep a counter that is decremented by the physical clock
 - Operating system set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity***; process is an ***active entity***.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
 - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users
- Memory management activities
 - Keeping track of which parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed

File-system Management

- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and directories
 - Mapping files onto secondary storage
 - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Mounting and unmounting
 - Free-space management
 - Storage allocation
 - Disk scheduling
 - Partitioning
 - Protection

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
 - If it is, information used directly from the cache (fast)
 - If not, data copied to cache and used there
- Cache smaller than storage being cached
 - Cache management important design problem
 - Cache size and replacement policy

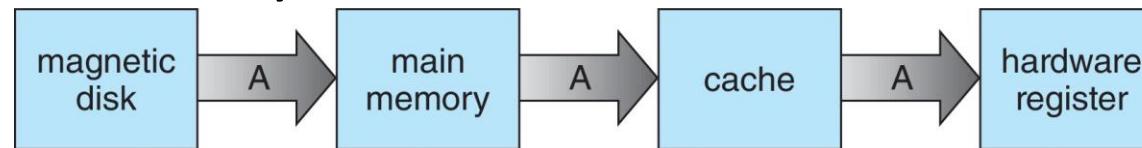
Characteristics of Various Types of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
 - Several copies of a datum can exist
 - Various solutions covered in Chapter 19

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
 - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
 - General device-driver interface
 - Drivers for specific hardware devices

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
 - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
 - User identities (**user IDs**, security IDs) include name and associated number, one per user
 - User ID then associated with all files, processes of that user to determine access control
 - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
 - **Privilege escalation** allows user to change to effective ID with more rights

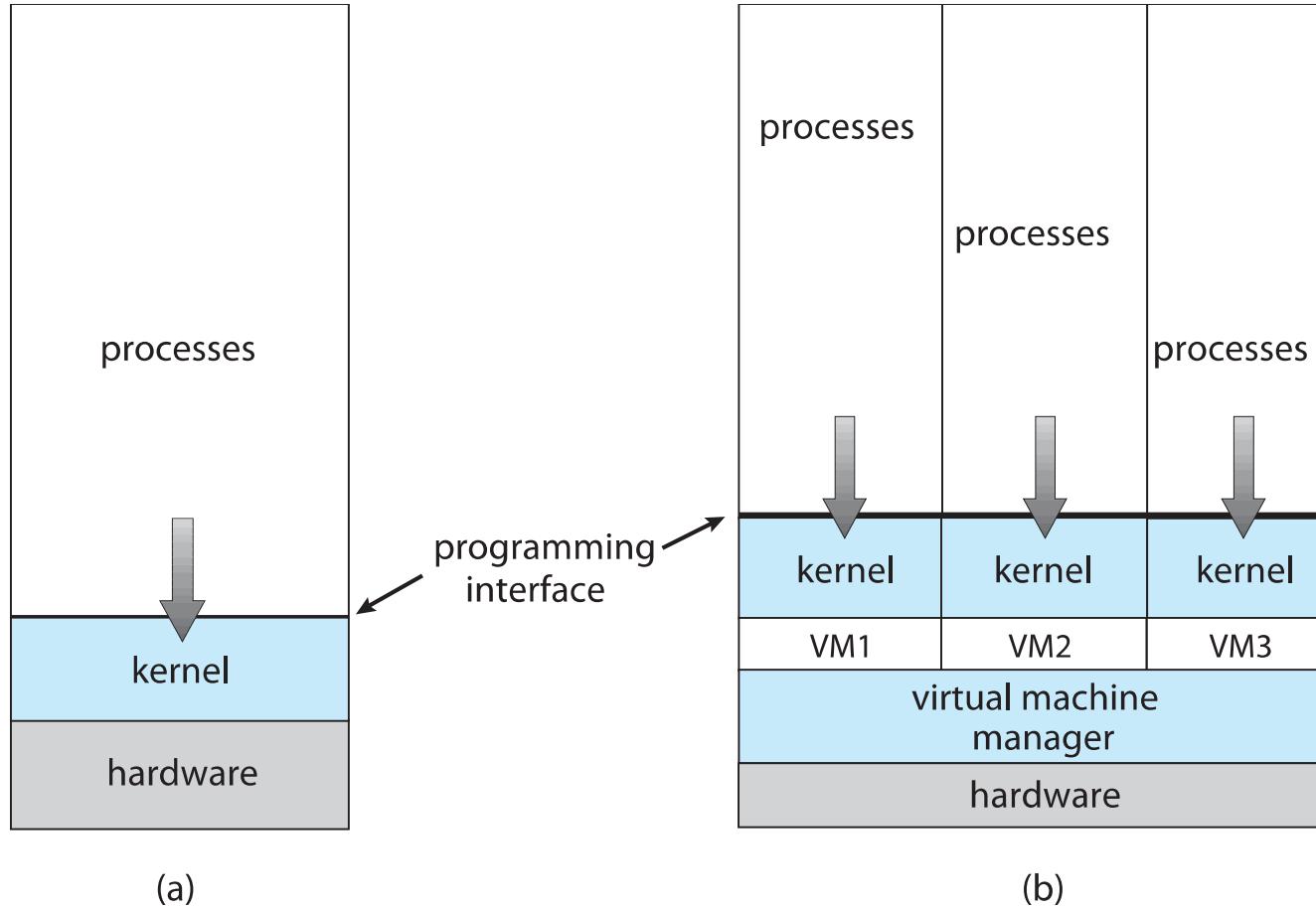
Virtualization

- Allows operating systems to run applications within other OSes
 - Vast and growing industry
- **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
 - Generally slowest method
 - When computer language not compiled to native code – **Interpretation**
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
 - Consider VMware running WinXP guests, each running applications, all on native WinXP **host** OS
 - **VMM** (virtual machine Manager) provides virtualization services

Virtualization (cont.)

- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
 - Apple laptop running Mac OS X host, Windows as a guest
 - Developing apps for multiple OSes without having multiple systems
 - Quality assurance testing applications without having multiple systems
 - Executing and managing compute environments within data centers
- VMM can run natively, in which case they are also the host
 - There is no general-purpose host then (VMware ESX and Citrix XenServer)

Computing Environments - Virtualization



Distributed Systems

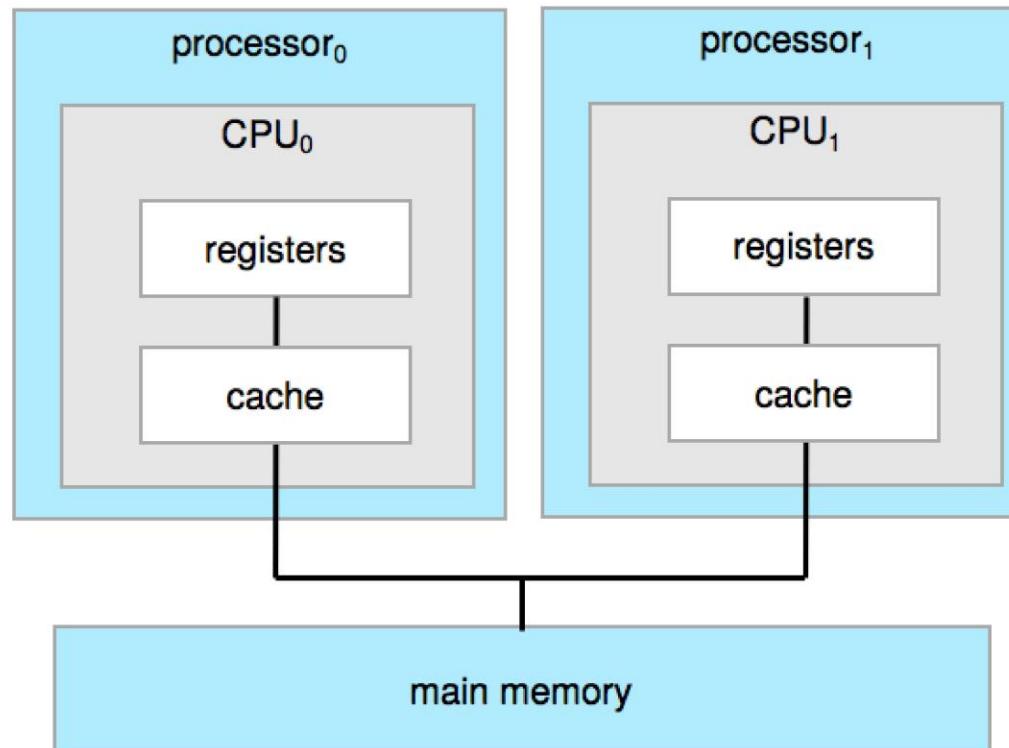
- Collection of separate, possibly heterogeneous, systems networked together
 - Network is a communications path, TCP/IP most common
 - Local Area Network (LAN)
 - Wide Area Network (WAN)
 - Metropolitan Area Network (MAN)
 - Personal Area Network (PAN)
 - Network Operating System provides features between systems across network
 - Communication scheme allows systems to exchange messages
 - Illusion of a single system

Computer System Architecture

Computer-System Architecture

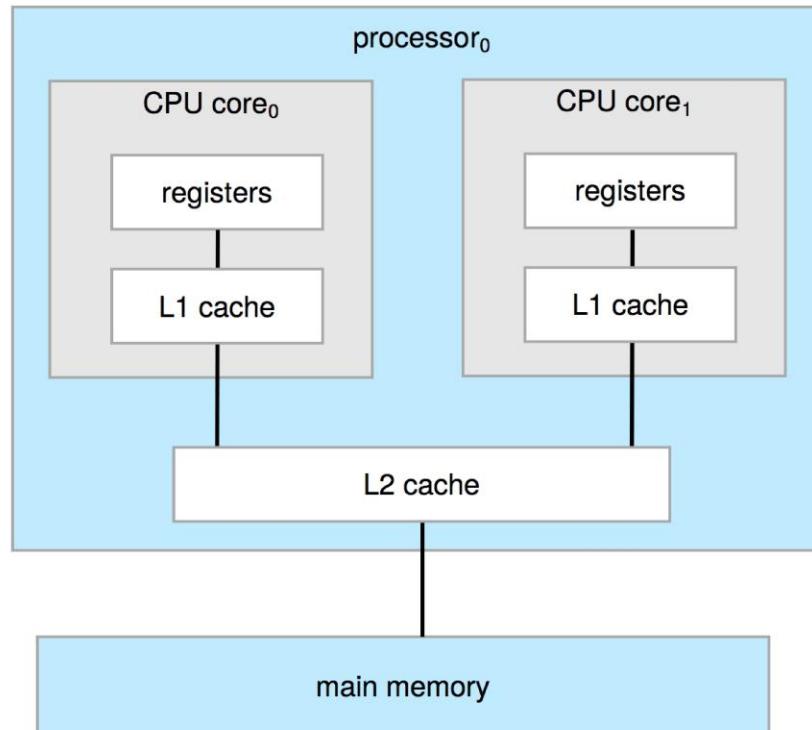
- Most systems use a single general-purpose processor
 - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
 - Also known as **parallel systems, tightly-coupled systems**
 - Advantages include:
 1. Increased throughput
 2. Economy of scale
 3. Increased reliability – graceful degradation or fault tolerance
 - Two types:
 1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
 2. **Symmetric Multiprocessing** – each processor performs all tasks

Symmetric Multiprocessing Architecture

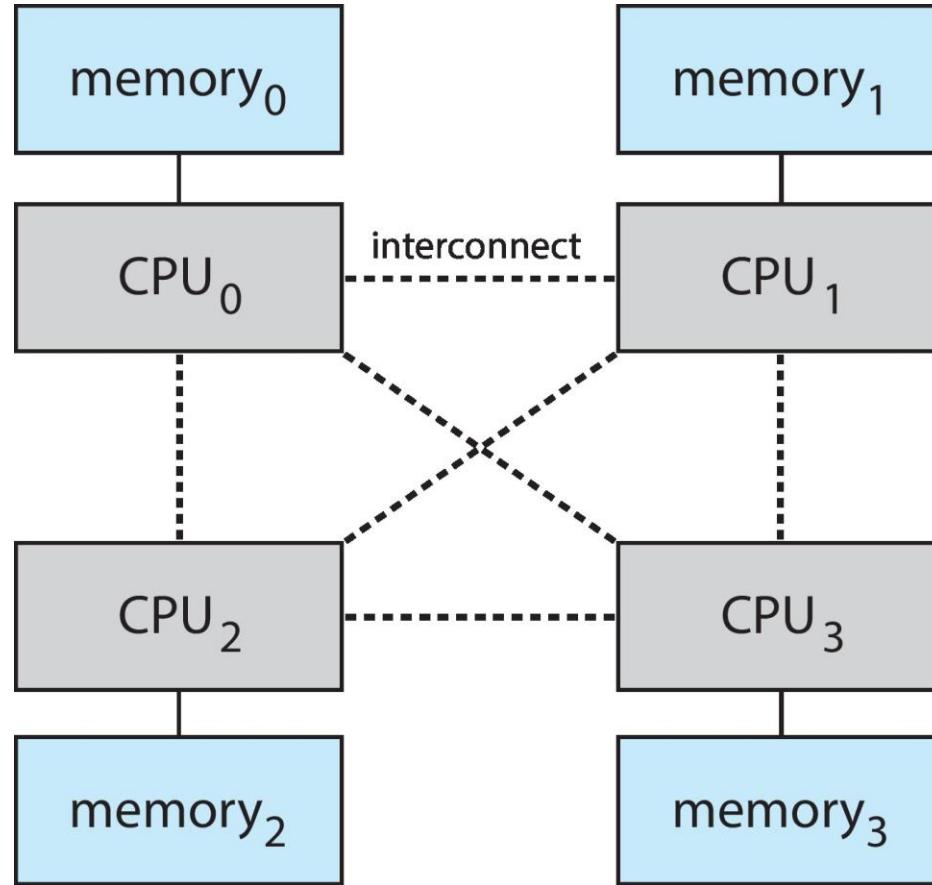


Dual-Core Design

- Multi-chip and **multicore**
- Systems containing all chips
 - Chassis containing multiple separate systems



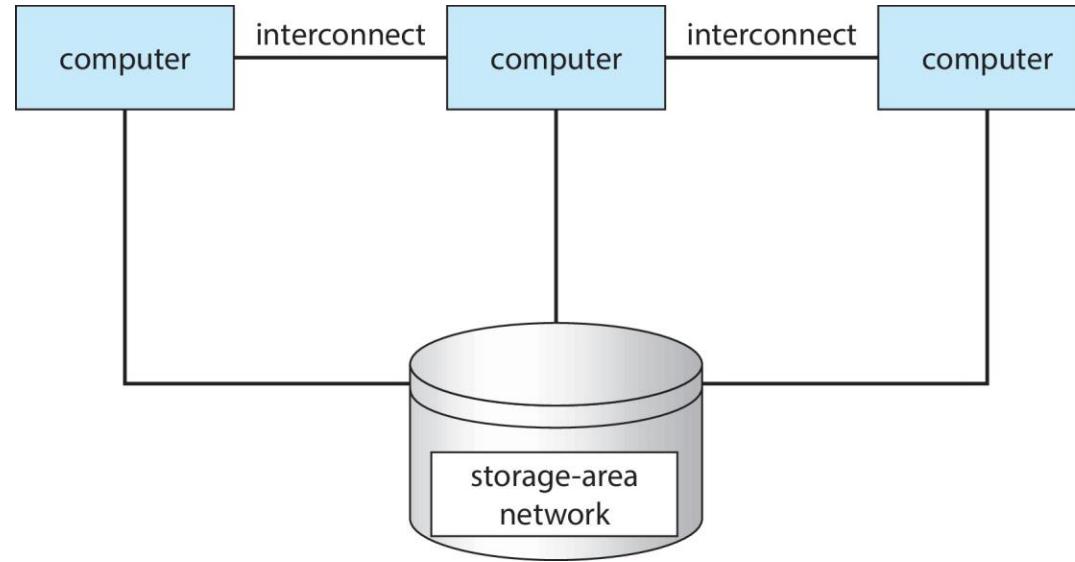
Non-Uniform Memory Access System



Clustered Systems

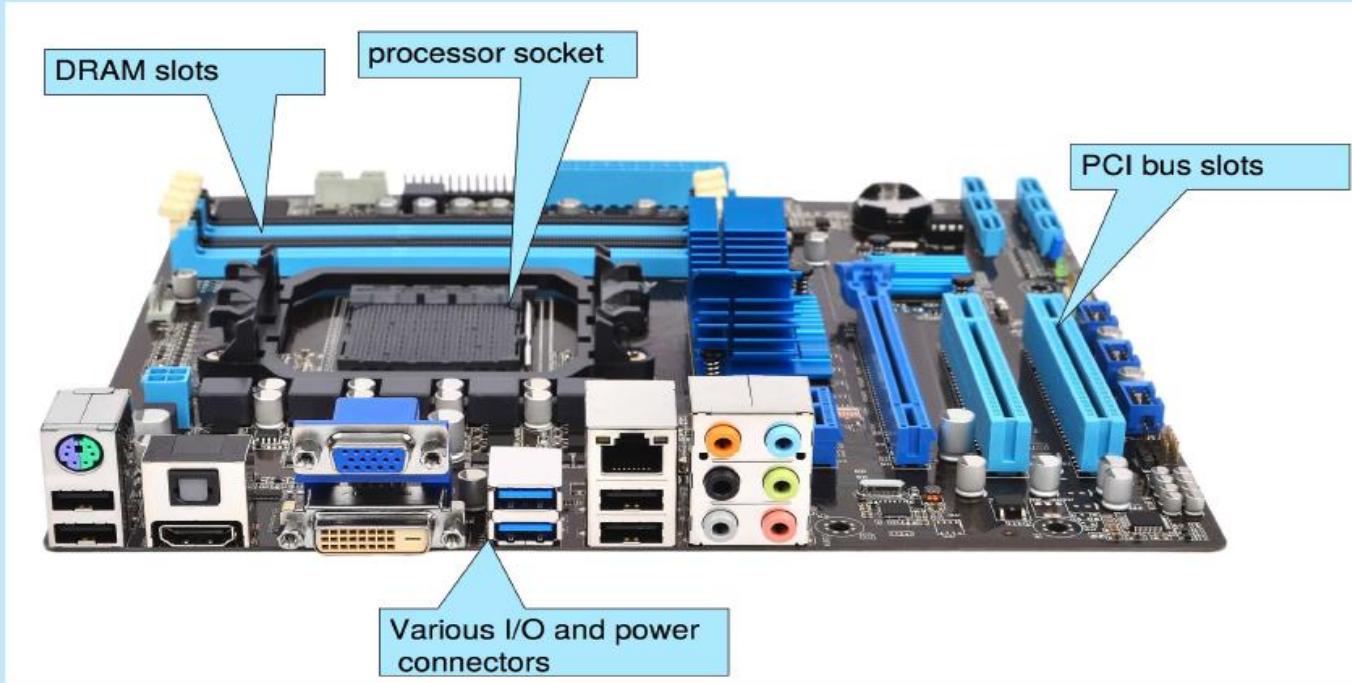
- Like multiprocessor systems, but multiple systems working together
 - Usually sharing storage via a **storage-area network (SAN)**
 - Provides a **high-availability** service which survives failures
 - **Asymmetric clustering** has one machine in hot-standby mode
 - **Symmetric clustering** has multiple nodes running applications, monitoring each other
 - Some clusters are for **high-performance computing (HPC)**
 - Applications must be written to use parallelization
 - Some have **distributed lock manager (DLM)** to avoid conflicting operations

Clustered Systems



PC Motherboard

Consider the desktop PC motherboard with a processor socket shown below:



This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

Computer System Environments

Computing Environments

- Traditional
- Mobile
- Client Server
- Peer-to-Peer
- Cloud computing
- Real-time Embedded

Traditional

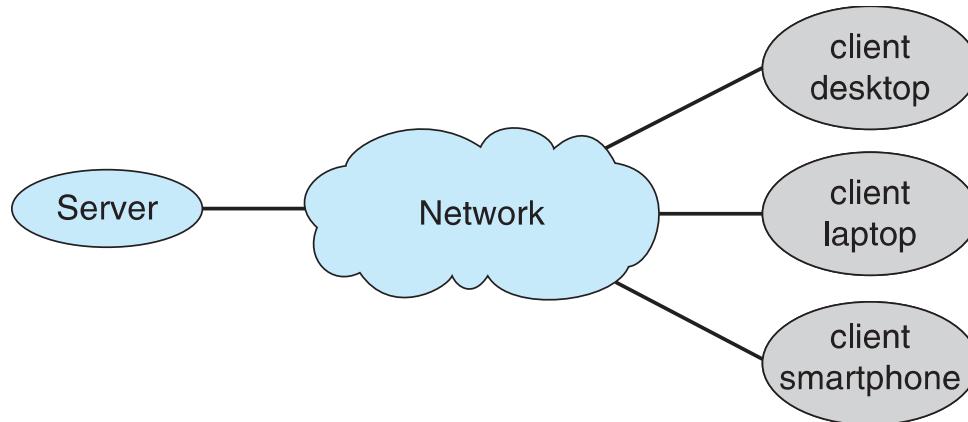
- Stand-alone general-purpose machines
- But blurred as most systems interconnect with others (i.e., the Internet)
- Portals provide web access to internal systems
- Network computers (thin clients) are like Web terminals
- Mobile computers interconnect via wireless networks
- Networking becoming ubiquitous – even home systems use firewalls to protect home computers from Internet attacks

Mobile

- Handheld smartphones, tablets, etc.
- What is the functional difference between them and a “traditional” laptop?
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like *augmented reality*
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are [Apple iOS](#) and [Google Android](#)

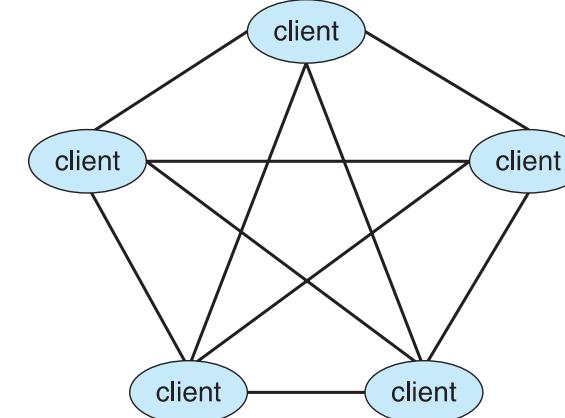
Client Server

- Client-Server Computing
 - Dumb terminals supplanted by smart PCs
 - Many systems now **servers**, responding to requests generated by **clients**
 - ▶ Compute-server system provides an interface to client to request services (i.e., database)
 - ▶ File-server system provides interface for clients to store and retrieve files



Peer-to-Peer

- Another model of distributed system
- P2P does not distinguish clients and servers
 - Instead all nodes are considered peers
 - May each act as client, server or both
 - Node must join P2P network
 - Registers its service with central lookup service on network, or
 - Broadcast request for service and respond to requests for service via ***discovery protocol***
 - Examples include Napster and Gnutella, [Voice over IP \(VoIP\)](#) such as Skype



Cloud Computing

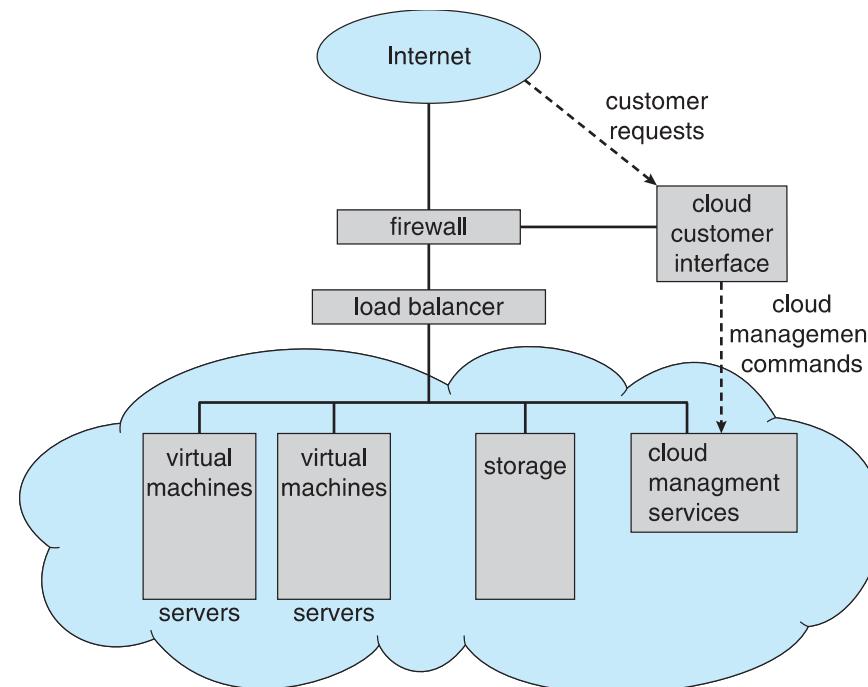
- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for its functionality.
 - Amazon [EC2](#) has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage

Cloud Computing (Cont.)

- Many types
 - **Public cloud** – available via Internet to anyone willing to pay
 - **Private cloud** – run by a company for the company's own use
 - **Hybrid cloud** – includes both public and private cloud components
 - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
 - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
 - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)

Cloud Computing (cont.)

- Cloud computing environments composed of traditional OSes, plus VMs, plus cloud management tools
 - Internet connectivity requires security like firewalls
 - Load balancers spread traffic across multiple applications



Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
 - Vary considerable, special purpose, limited purpose OS, **real-time OS**
 - Use expanding
- Many other special computing environments as well
 - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
 - Processing **must** be done within constraint
 - Correct operation only if constraints met

Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source** and **proprietary**
- Counter to the **copy protection** and **Digital Rights Management (DRM)** movement
- Started by **Free Software Foundation (FSF)**, which has “copyleft” **GNU Public License (GPL)**
 - Free software and open-source software are two different ideas championed by different groups of people
 - <https://www.gnu.org/philosophy/open-source-misses-the-point.en.html>
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - <http://www.virtualbox.com>)
 - Use to run guest operating systems for exploration

The Study of Operating Systems

There has never been a more interesting time to study operating systems, and it has never been easier. The open-source movement has overtaken operating systems, causing many of them to be made available in both source and binary (executable) format. The list of operating systems available in both formats includes Linux, BSD UNIX, Solaris, and part of macOS.

The availability of source code allows us to study operating systems from the inside out.

Questions that we could once answer only by looking at documentation or the behavior of an operating system we can now answer by examining the code itself.

Operating systems that are no longer commercially viable have been open-sourced as well, enabling us to study how systems operated in a time of fewer CPU, memory, and storage resources.

An extensive but incomplete list of open-source operating-system projects is available from https://curlie.org/Computers/Software/Operating_Systems/Open_Source/

In addition, the rise of virtualization as a mainstream (and frequently free) computer function makes it possible to run many operating systems on top of one core system. For example, VMware (<http://www.vmware.com>) provides a free “player” for Windows on which hundreds of free “virtual appliances” can run. Virtualbox (<http://www.virtualbox.com>) provides a free, open-source virtual machine manager on many operating systems. Using such tools, students can try out hundreds of operating systems without dedicated hardware.

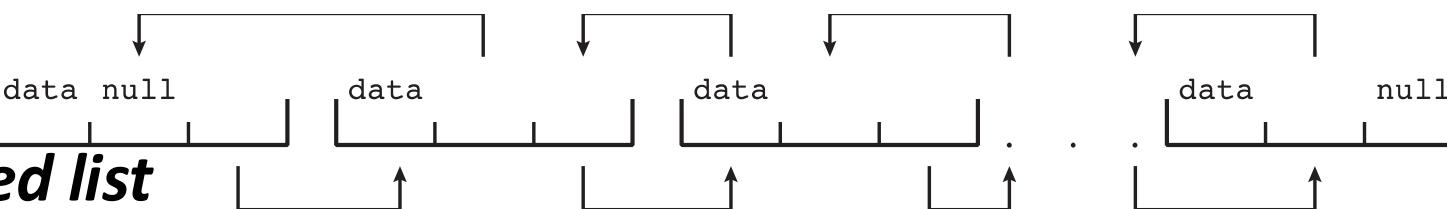
The advent of open-source operating systems has also made it easier to make the move from student to operating-system developer. With some knowledge, some effort, and an Internet connection, a student can even create a new operating-system distribution. Just a few years ago, it was difficult or impossible to get access to source code. Now, such access is limited only by how much interest, time, and disk space a student has.

Kernel Data Structure

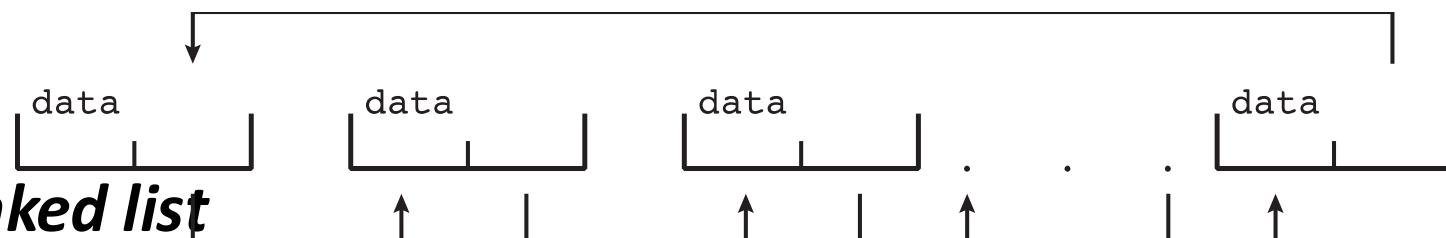
- Many similar to standard programming data structures



- *Singly linked list*



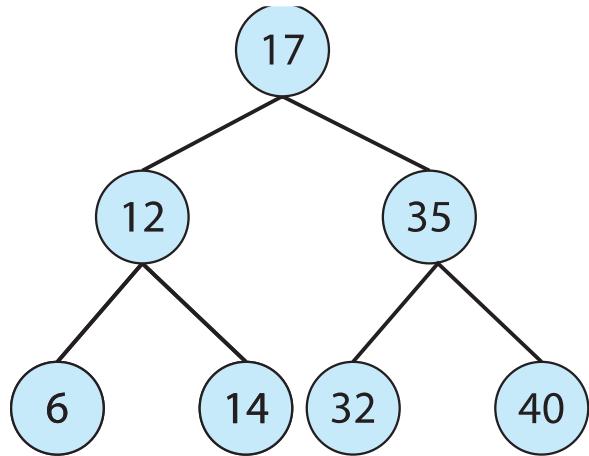
- *Doubly linked list*



- *Circular linked list*

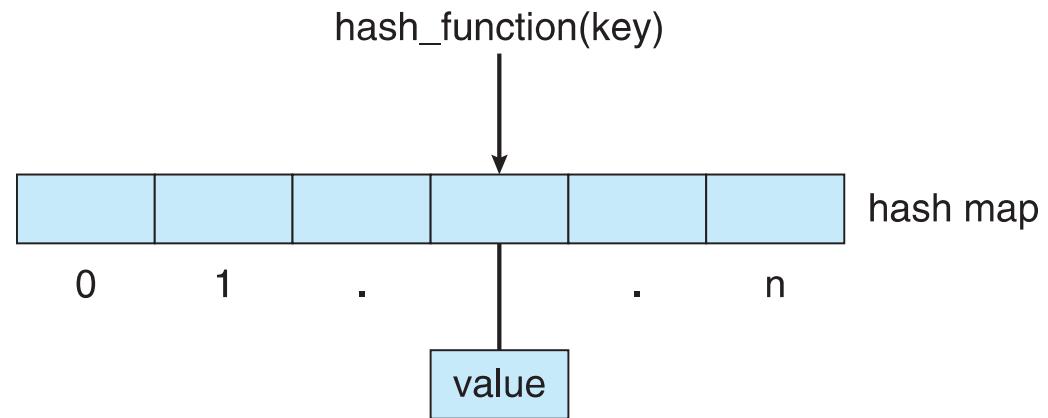
Kernel Data Structures

- **Binary search tree**
left \leq right
 - Search performance is $O(n)$
 - **Balanced binary search tree** is $O(\lg n)$



Kernel Data Structures

- **Hash function** can create a **hash map**



- **Bitmap** – string of n binary digits representing the status of n items
- Linux data structures defined in **include** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Objectives

Identify services provided by an operating system

- Illustrate how system calls are used to provide operating system services
- Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems
- Illustrate the process for booting an operating system
- Apply tools for monitoring operating system performance
- Design and implement kernel modules for interacting with a Linux kernel

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:
 - **User interface** - Almost all operating systems have a user interface ([UI](#)).
 - Varies between [Command-Line \(CLI\)](#), [Graphics User Interface \(GUI\)](#), [touch-screen](#), [Batch](#)
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
 - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file information, permission management.

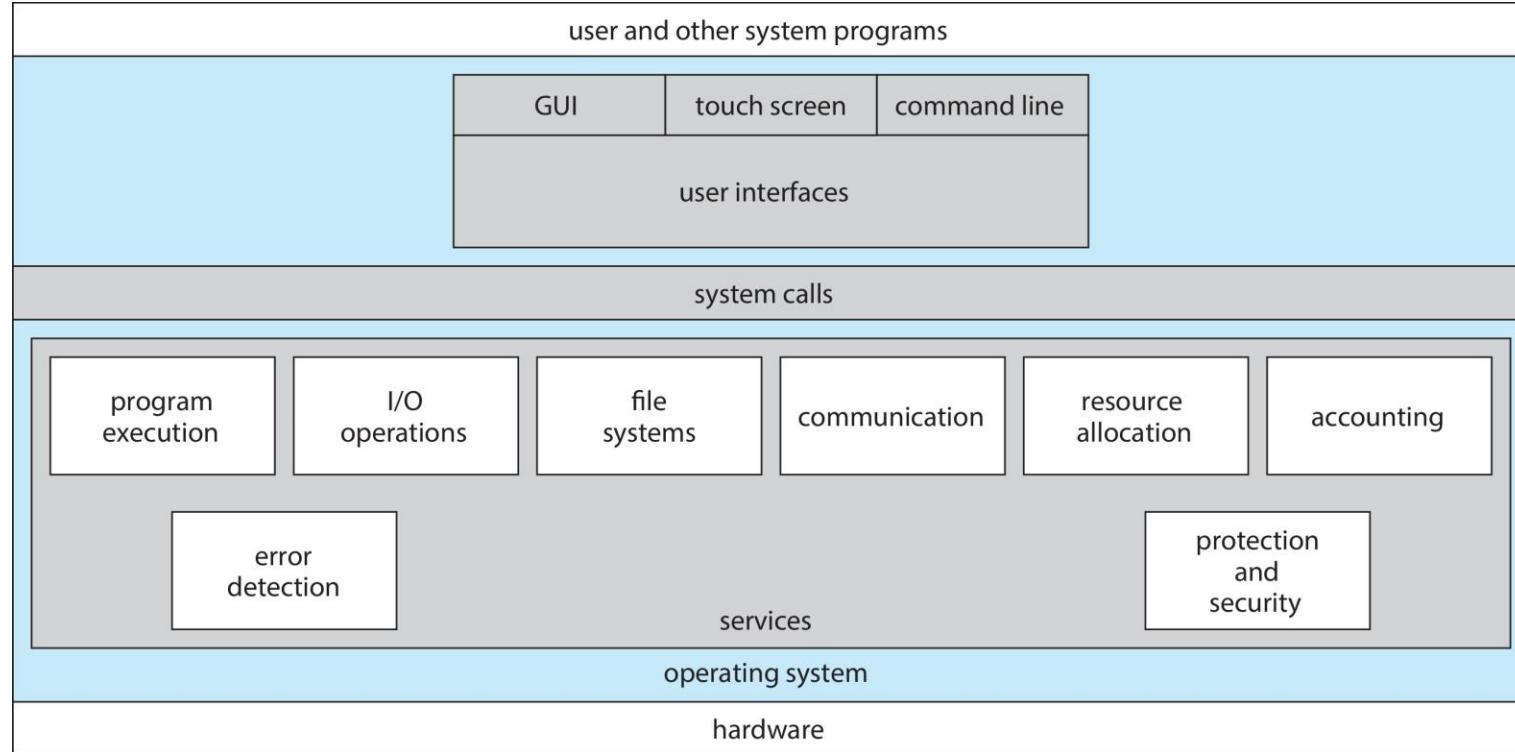
Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
 - **Communications** – Processes may exchange information, on the same computer or between computers over a network
 - Communications may be via shared memory or through message passing (packets moved by the OS)
 - **Error detection** – OS needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
 - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
 - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
 - **Logging** - To keep track of which users use how much and what kinds of computer resources
 - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - **Protection** involves ensuring that all access to system resources is controlled
 - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



Command Line interpreter

- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented
 - **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter

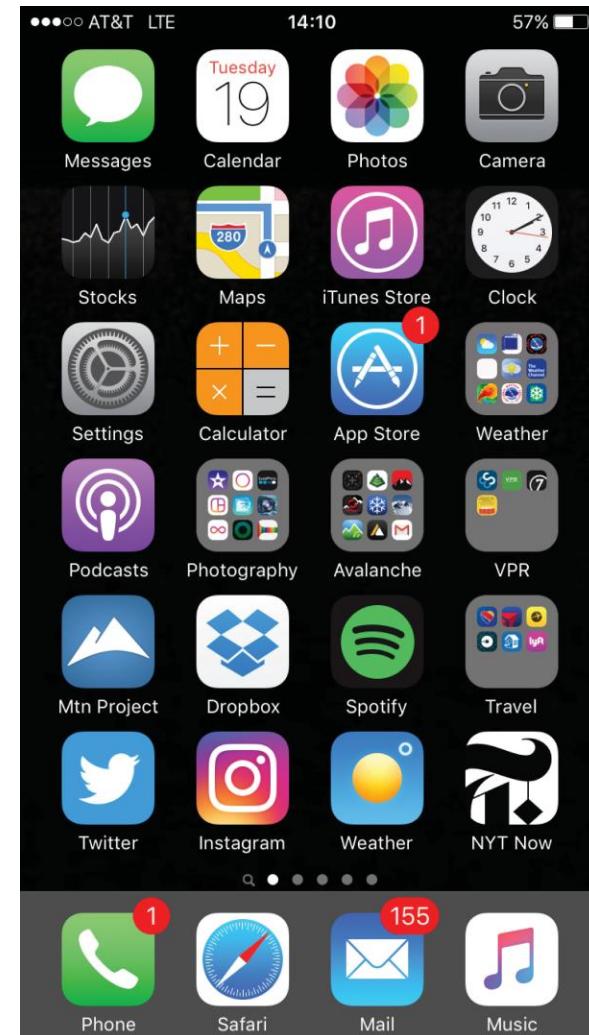
```
1. root@r6181-d5-us01:~ (ssh)
root@r6181-d5-us01:~ (ssh) × root@r6181-d5-u... ● %1 × ssh ☀ %2 × root@r6181-d5-us01... %3
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                  50G   19G   28G  41% /
tmpfs           127G  520K  127G   1% /dev/shm
/dev/sda1        477M   71M   381M  16% /boot
/dev/dssd0000    1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                  12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test    23T  1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?  S<Ll Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0     0     0 ?      S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0     0     0 ?      S    Jul12 177:42 [vpthread-1-2]
root      3829  3.0  0.0     0     0 ?      S   Jun27 730:04 [rp_thread 7:0]
root      3826  3.0  0.0     0     0 ?      S   Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x----- 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
```

User Operating System Interface - GUI

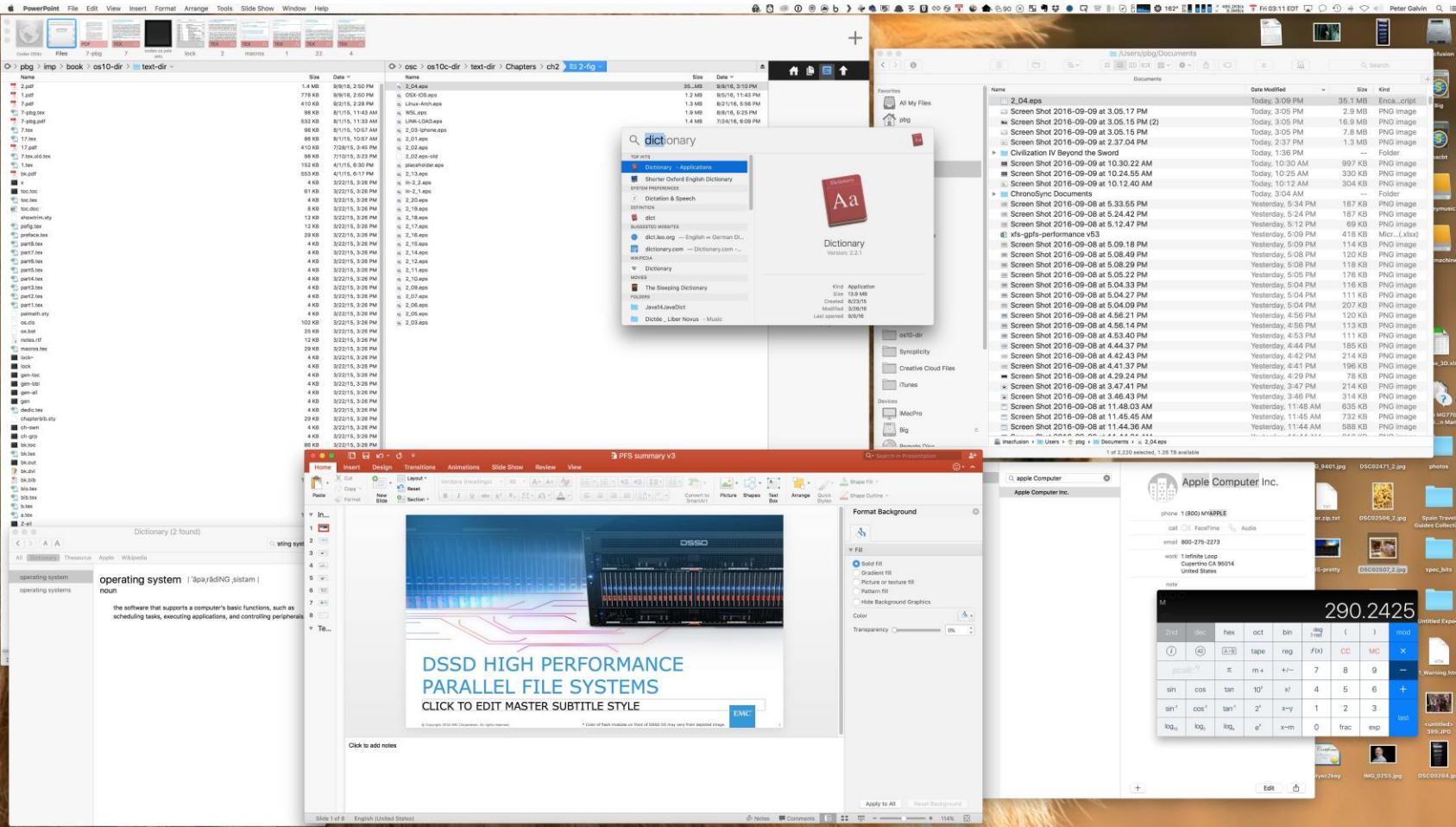
- User-friendly **desktop** metaphor interface
 - Usually mouse, keyboard, and monitor
 - **Icons** represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
 - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen Interfaces

- Touchscreen devices require new interfaces
 - Mouse not possible or not desired
 - Actions and selection based on gestures
 - Virtual keyboard for text entry
- Voice commands



The Mac OS X GUI



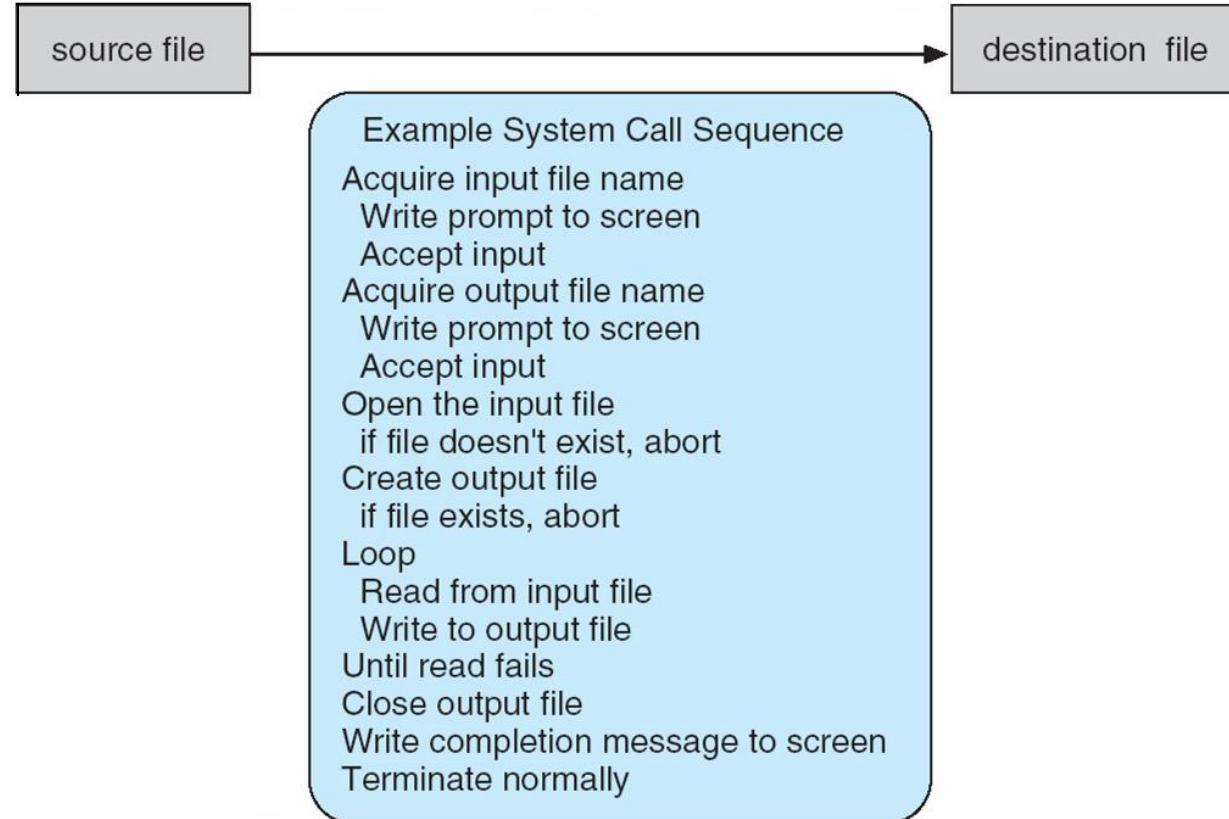
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout this text are generic

Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t      read(int fd, void *buf, size_t count)
```

return value function name parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

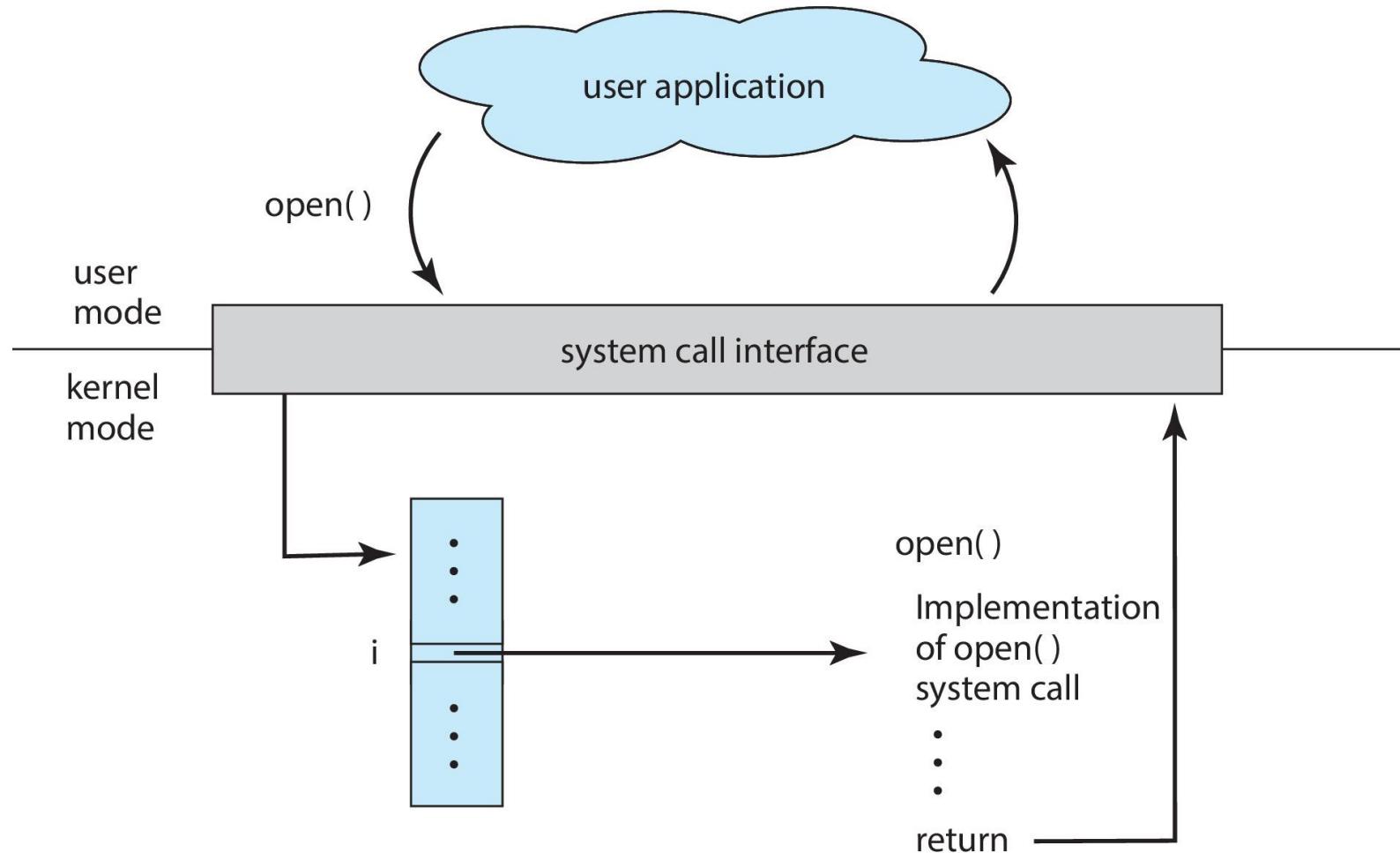
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

System Call Implementation

- Typically, a number is associated with each system call
 - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most details of OS interface hidden from programmer by API
 - Managed by run-time support library (set of functions built into libraries included with compiler)

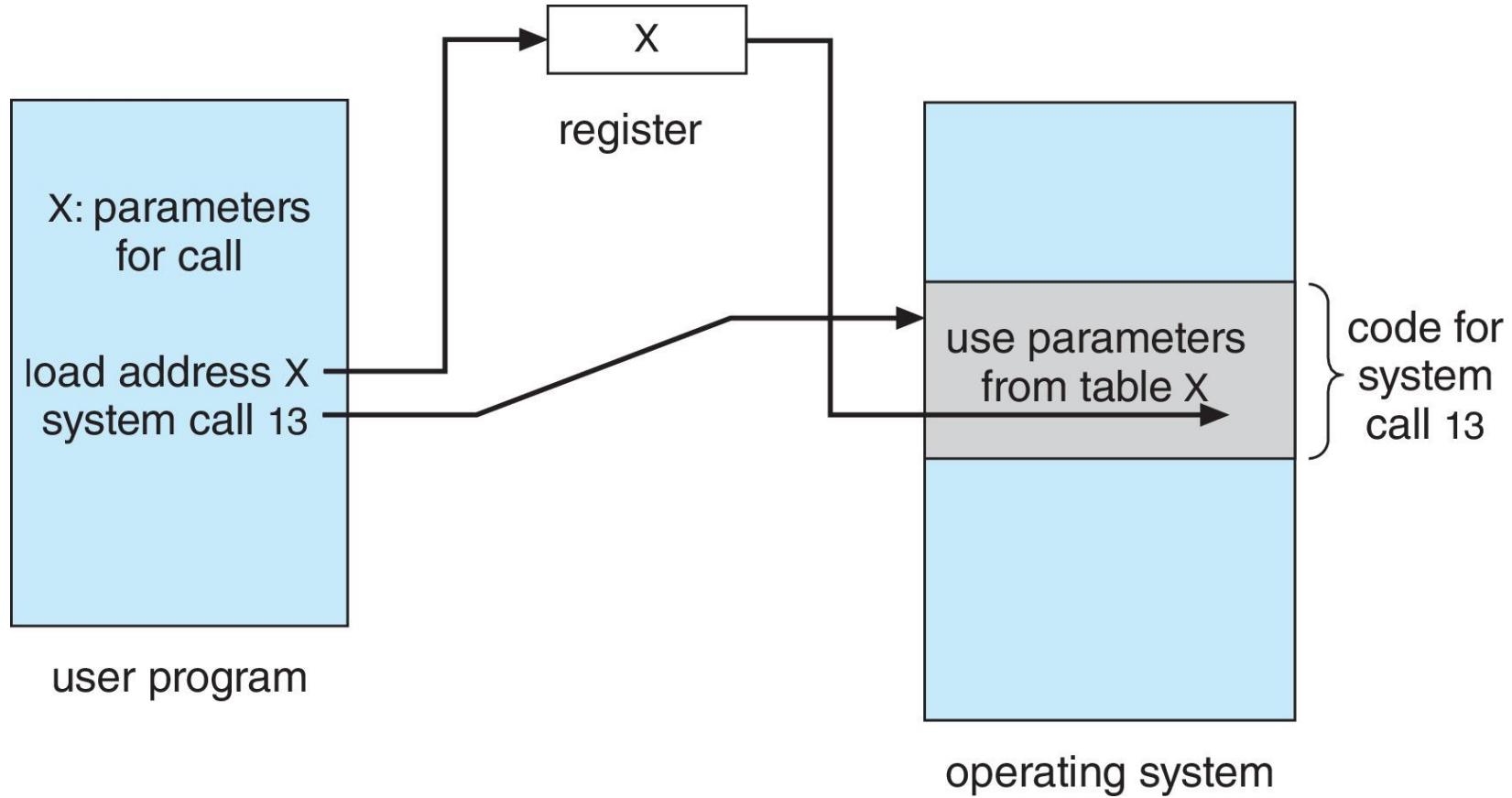
API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
 - Simplest: pass the parameters in registers
 - In some cases, may be more parameters than registers
 - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
 - This approach taken by Linux and Solaris
 - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
 - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls

- Process control
 - create process, terminate process
 - end, abort
 - load, execute
 - get process attributes, set process attributes
 - wait for time
 - wait event, signal event
 - allocate and free memory
 - Dump memory if error
 - **Debugger** for determining **bugs, single step** execution
 - **Locks** for managing access to shared data between processes

Types of System Calls (Cont.)

- File management
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes
- Device management
 - request device, release device
 - read, write, reposition
 - get device attributes, set device attributes
 - logically attach or detach devices

Types of System Calls (Cont.)

- Information maintenance
 - get time or date, set time or date
 - get system data, set system data
 - get and set process, file, or device attributes
- Communications
 - create, delete communication connection
 - send, receive messages if **message passing model** to **host name or process name**
 - From **client** to **server**
 - **Shared-memory model** create and gain access to memory regions
 - transfer status information
 - attach and detach remote devices

Types of System Calls (Cont.)

- Protection
 - Control access to resources
 - Get and set permissions
 - Allow and deny user access

Examples of Windows and Unix System Calls

GREATER JAKARTA • BANDUNG • MALANG

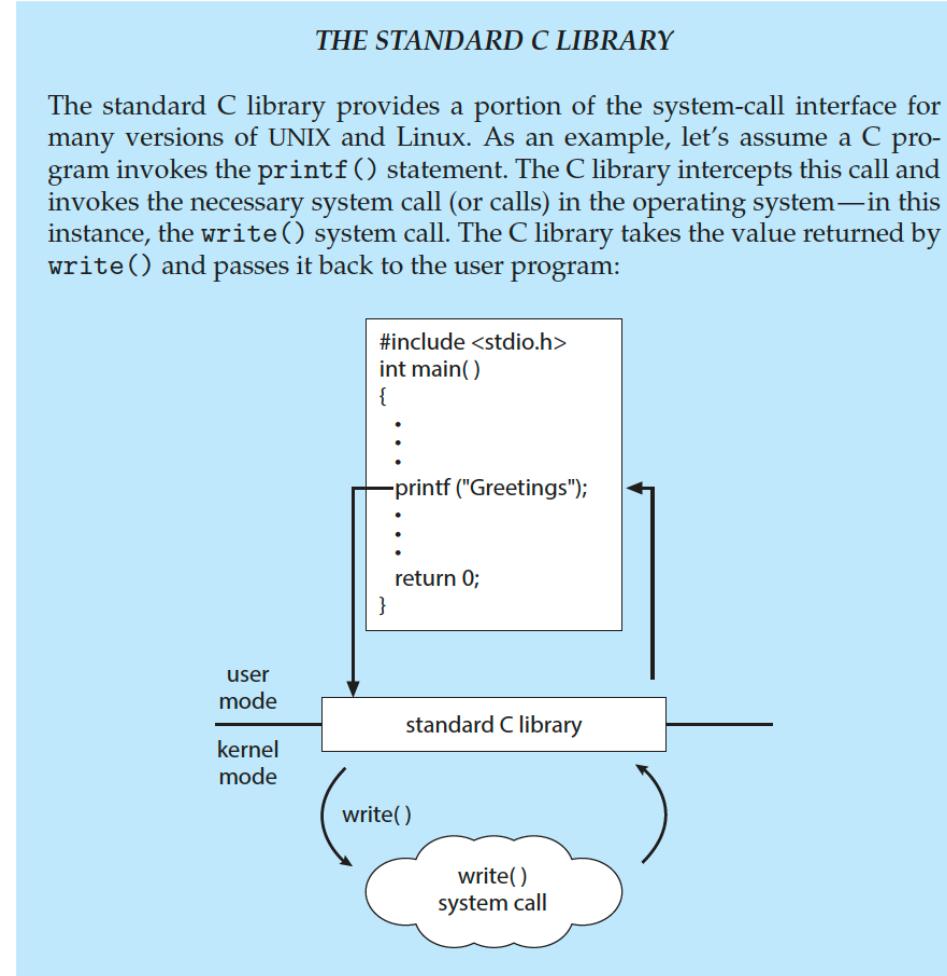
EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

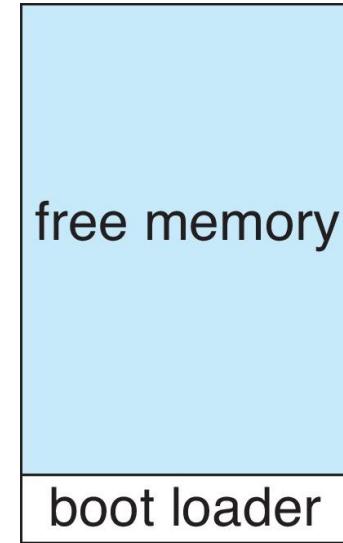
Standard C Library Example

- C program invoking `printf()` library call, which calls `write()`



Example: Arduino

- Single-tasking
- No operating system
- Programs (sketch) loaded via USB into flash memory
- Single memory space
- Boot loader loads program
- Program exit -> shell reloaded



(a)

At system startup

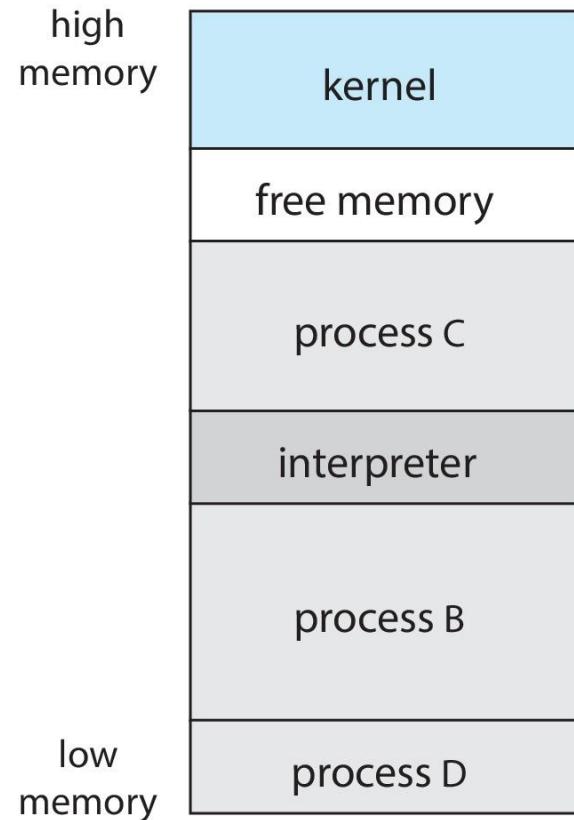


(b)

running a program

Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
 - Executes exec() to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with:
 - code = 0 – no error
 - code > 0 – error code



System Services

- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation
 - Status information sometimes stored in a file
 - Programming language support
 - Program loading and execution
 - Communications
 - Background services
 - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls

System Services (Cont.)

- Provide a convenient environment for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
 - Some ask the system for info - date, time, amount of available memory, disk space, number of users
 - Others provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a **registry** - used to store and retrieve configuration information

System Services (Cont.)

- **File modification**
 - Text editors to create and modify files
 - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

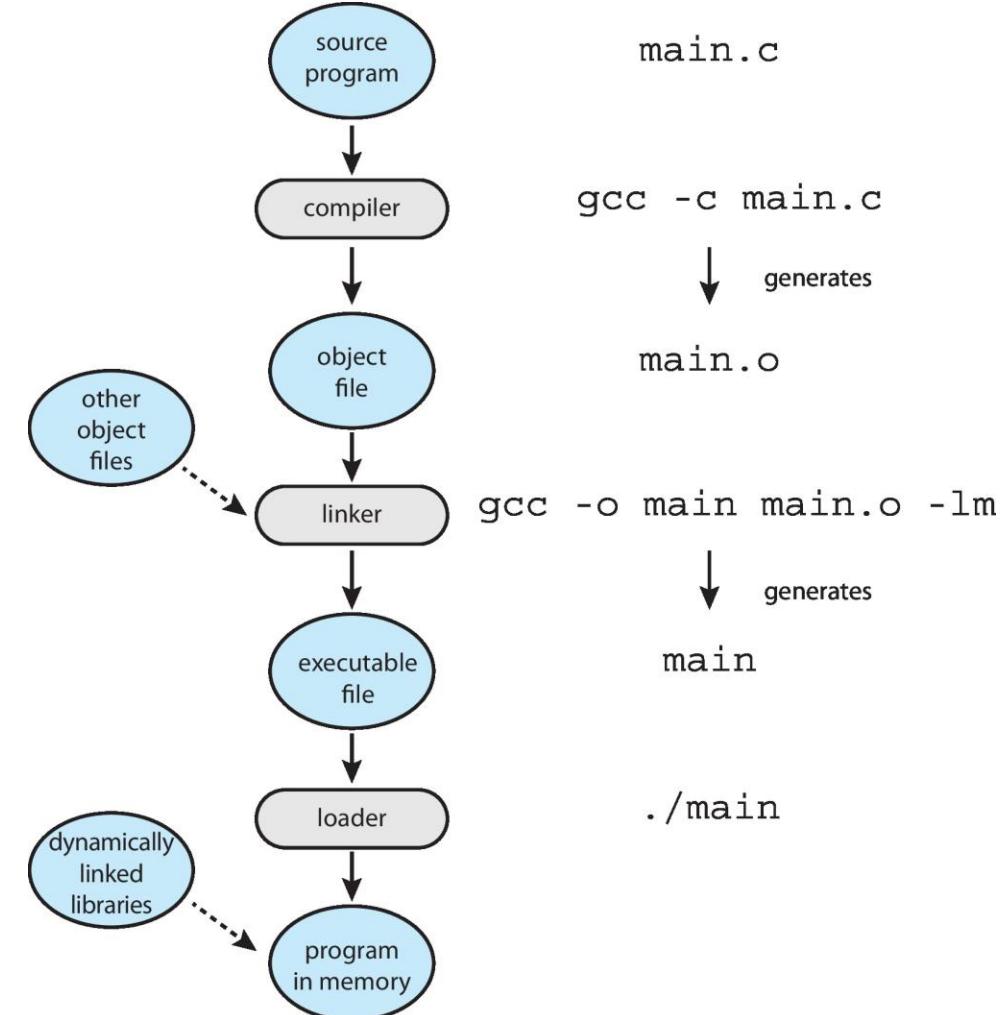
System Services (Cont.)

- **Background Services**
 - Launch at boot time
 - Some for system startup, then terminate
 - Some from system boot to shutdown
 - Provide facilities like disk checking, process scheduling, error logging, printing
 - Run in user context not kernel context
 - Known as **services**, **subsystems**, **daemons**
- **Application programs**
 - Don't pertain to system
 - Run by users
 - Not typically considered part of OS
 - Launched by command line, mouse click, finger poke

Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
 - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
 - **Relocation** assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
 - Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

The Role of the Linker and Loader



Why Applications are Operating System Specific

GREATER JAKARTA • BANDUNG • MALANG



- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
 - Own file formats, etc.
- Apps can be multi-operating system
 - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
 - App written in language that includes a VM containing the running app (like Java)
 - Use standard language (like C), compile separately on each operating system to run on each
- **Application Binary Interface (ABI)** is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.

Design and Implementation

- Design and Implementation of OS is not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals
 - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
 - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Specifying and designing an OS is highly creative task of **software engineering**

Policy and Mechanism

- **Policy:** **What** needs to be done?
 - Example: Interrupt after every 100 seconds
- **Mechanism:** **How** to do something?
 - Example: timer
- Important principle: separate policy from mechanism
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.
 - Example: change 100 to 200

Implementation

- Much variation
 - Early OSes in assembly language
 - Then system programming languages like Algol, PL/1
 - Now C, C++
- Actually usually a mix of languages
 - Lowest levels in assembly
 - Main body in C
 - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
 - But slower
- **Emulation** can allow an OS to run on non-native hardware

Operating System Structure

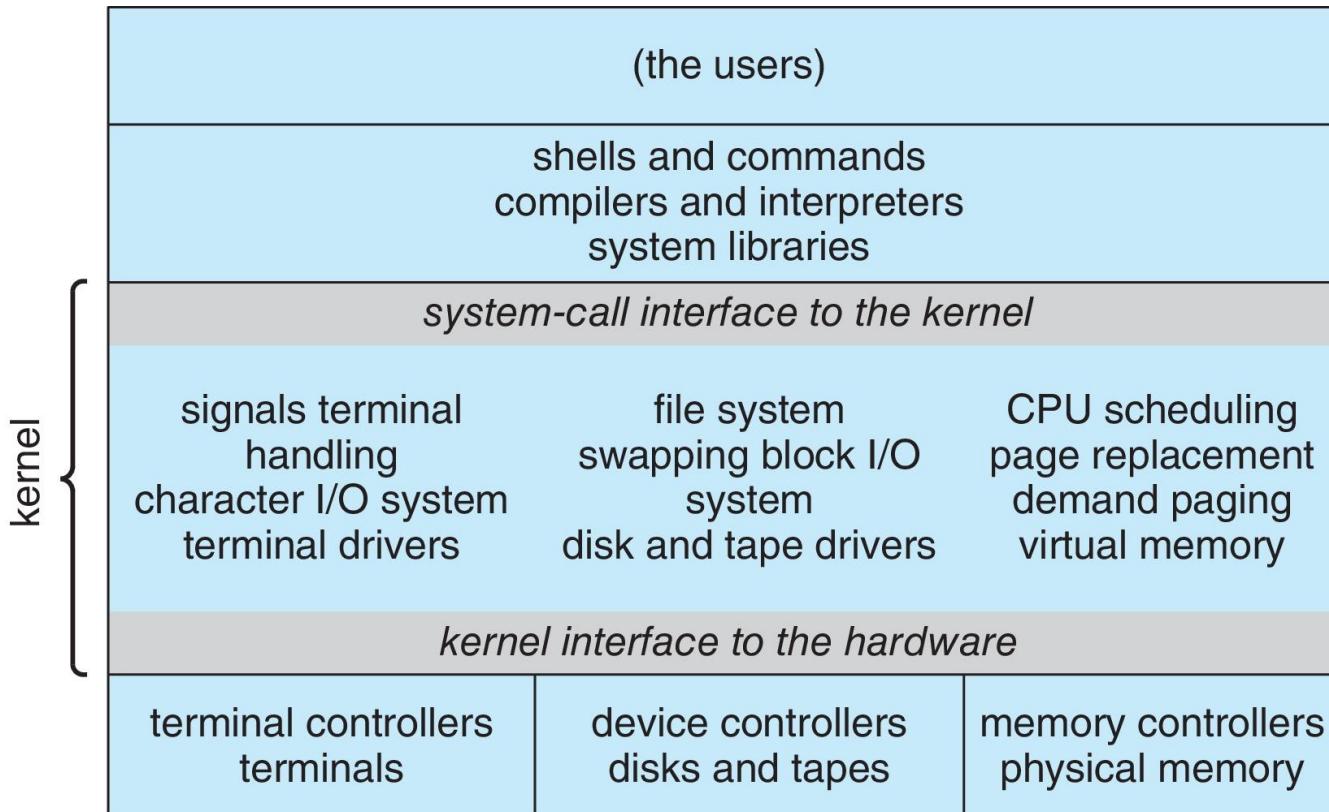
- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure – MS-DOS
 - More complex – UNIX
 - Layered – an abstraction
 - Microkernel – Mach

Monolithic Structure – Original UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
 - Systems programs
 - The kernel
 - Consists of everything below the system-call interface and above the physical hardware
 - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

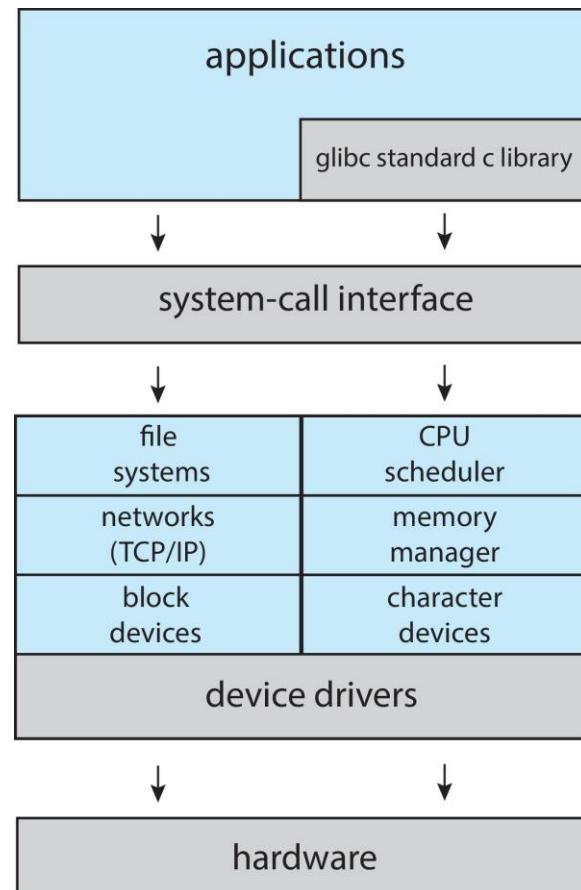
Traditional UNIX System Structure

Beyond simple but not fully layered



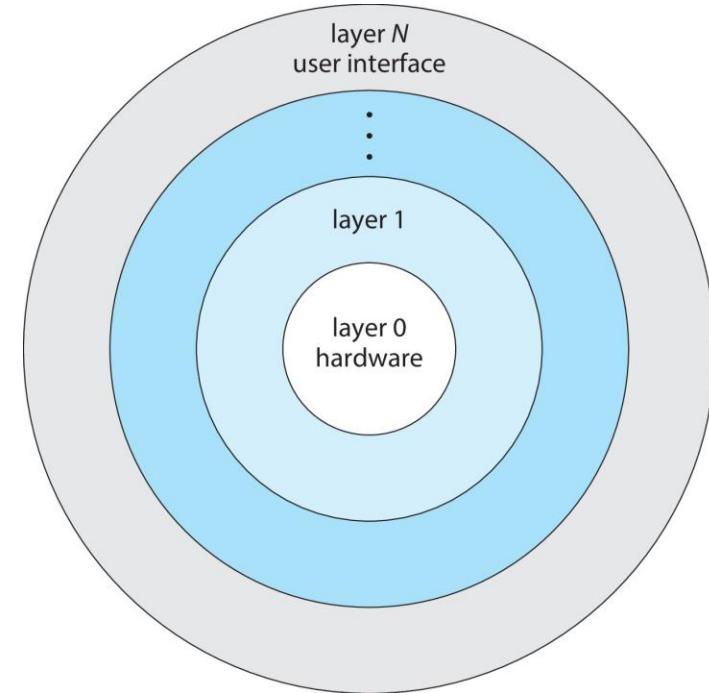
Linux System Structure

Monolithic plus modular design



Layered Approach

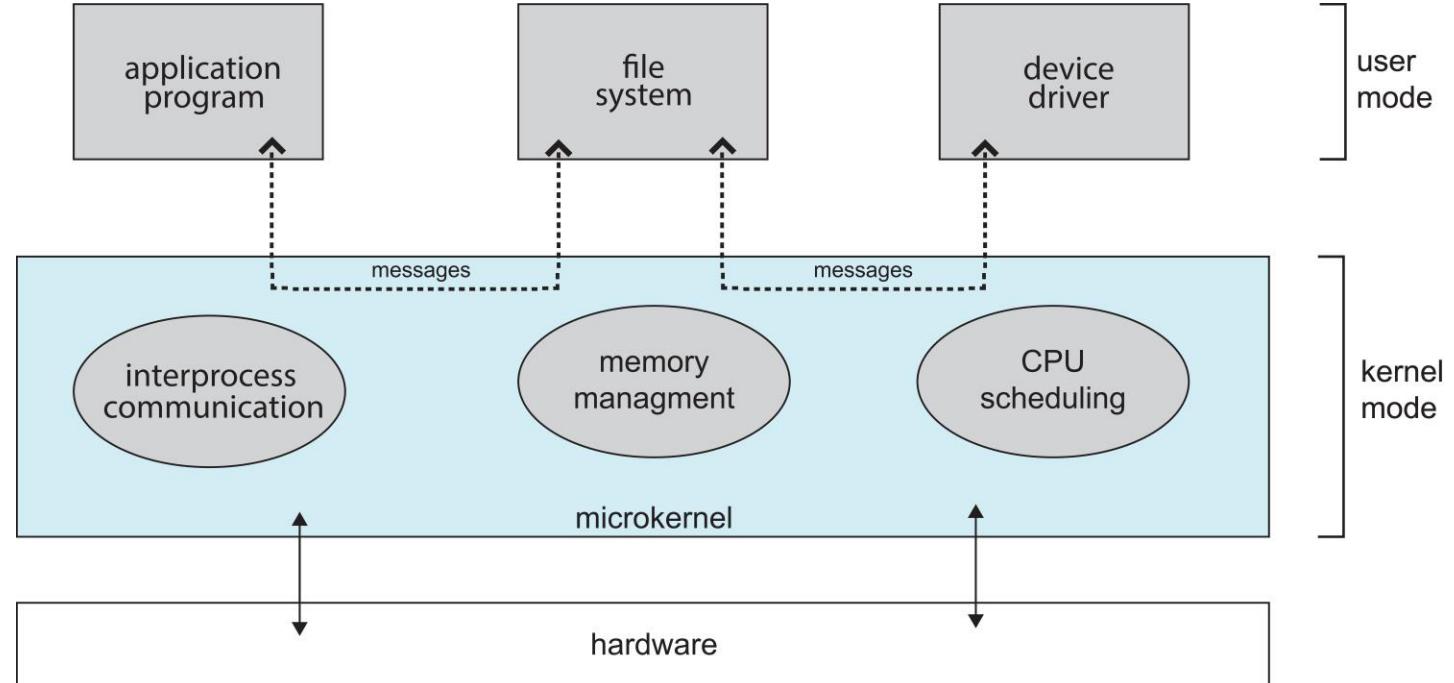
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Microkernels

- Moves as much from the kernel into user space
- **Mach** is an example of **microkernel**
 - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments:
 - Performance overhead of user space to kernel space communication

Microkernel System Structure



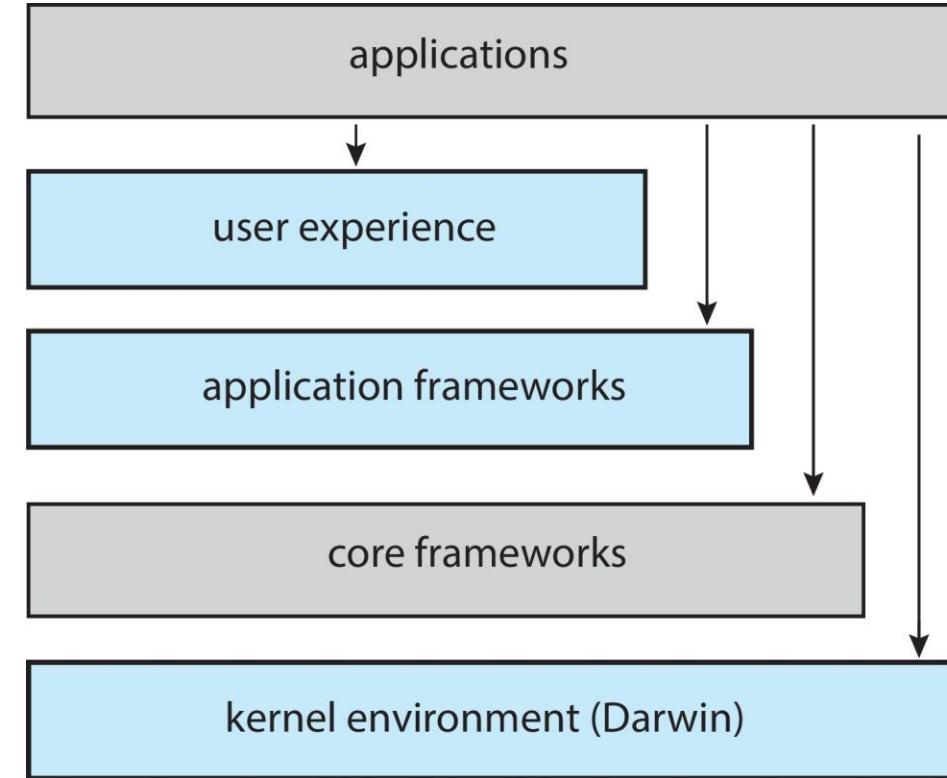
Modules

- Many modern operating systems implement **loadable kernel modules (LKMs)**
 - Uses object-oriented approach
 - Each core component is separate
 - Each talks to the others over known interfaces
 - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
 - Linux, Solaris, etc.

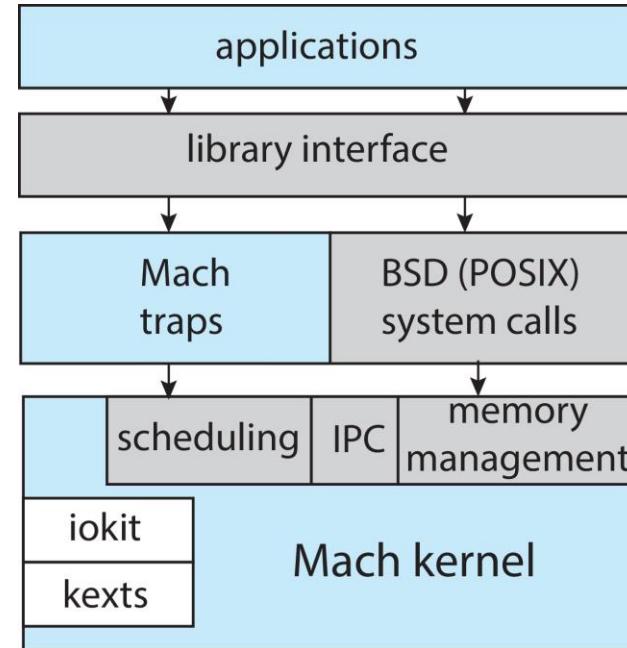
Hybrid Systems

- Most modern operating systems are not one pure model
 - Hybrid combines multiple approaches to address performance, security, usability needs
 - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
 - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
 - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

macOS and iOS Structure

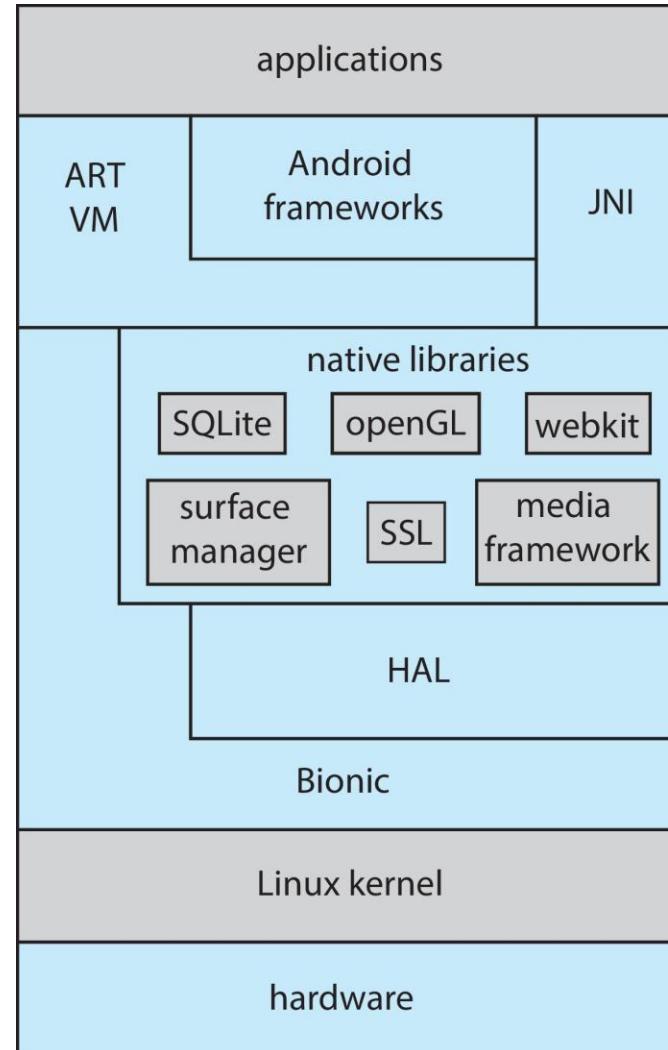


Darwin



- Developed by Open Handset Alliance (mostly Google)
 - Open Source
- Similar stack to iOS
- Based on Linux kernel but modified
 - Provides process, memory, device-driver management
 - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
 - Apps developed in Java plus Android API
 - Java class files compiled to Java bytecode then translated to executable then runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
 - But can build and install some other operating systems
 - If generating an operating system from scratch
 - Write the operating system source code
 - Configure the operating system for the system on which it will run
 - Compile the operating system
 - Install the operating system
 - Boot the computer and its new operating system

Building and Booting Linux

- Download Linux source code (<http://www.kernel.org>)
- Configure kernel via “make menuconfig”
- Compile the kernel using “make”
 - Produces vmlinuz, the kernel image
 - Compile kernel modules via “make modules”
 - Install kernel modules into vmlinuz via “make modules_install”
 - Install new kernel on the system via “make install”

System Boot

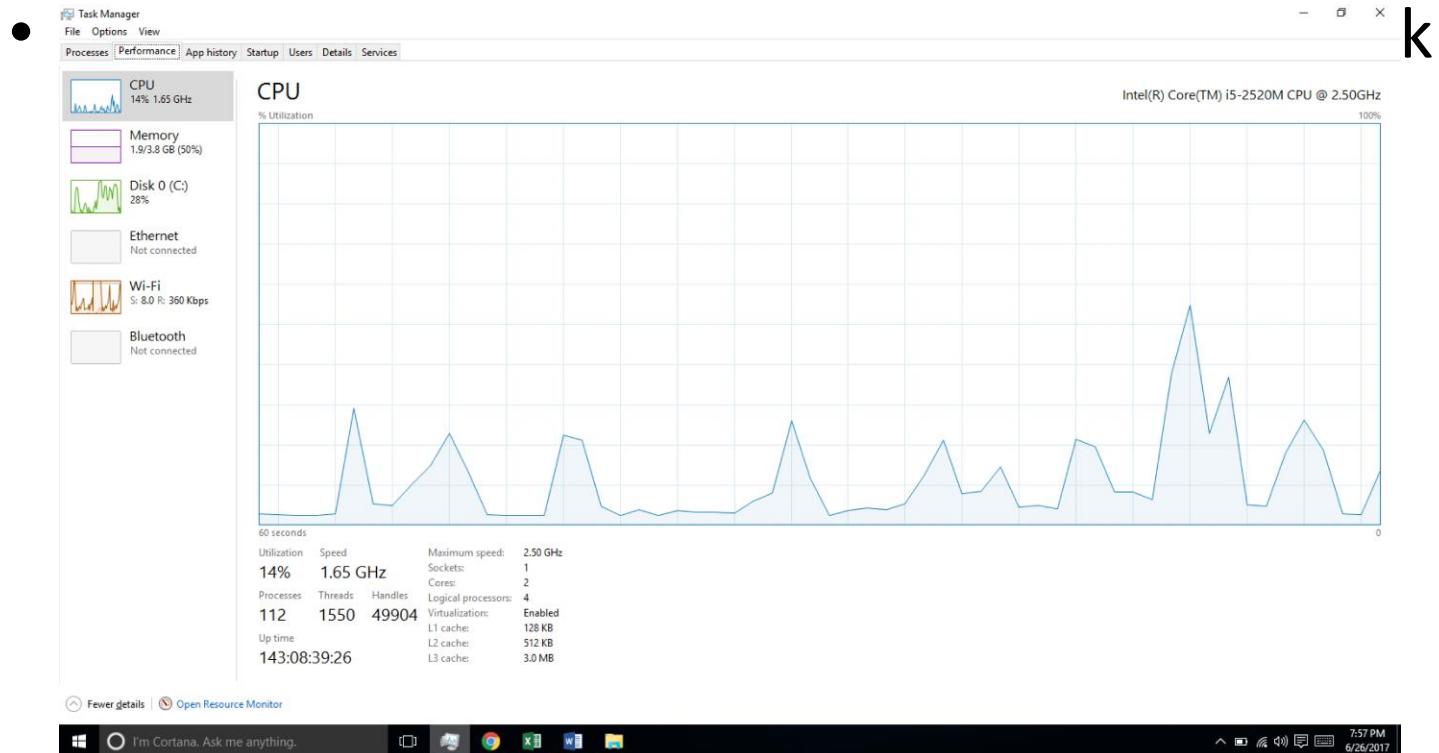
- When power initialized on system, execution starts at a fixed memory location
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap loader**, **BIOS**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
 - Modern systems replace BIOS with **Unified Extensible Firmware Interface (UEFI)**
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**
- Boot loaders frequently allow various boot states, such as single user mode

- Debugging is finding and fixing errors, or bugs
- Also performance tuning
- OS generate log files containing error information
- Failure of an application can generate core dump file capturing memory of the process
- Operating system failure can generate crash dump file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
 - Sometimes using *trace listings* of activities, recorded for analysis
 - Profiling is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law: “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior



Tracing

- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include
 - strace – trace system calls invoked by a process
 - gdb – source-level debugger
 - perf – collection of Linux performance tools
 - tcpdump – collects network packets

BCC

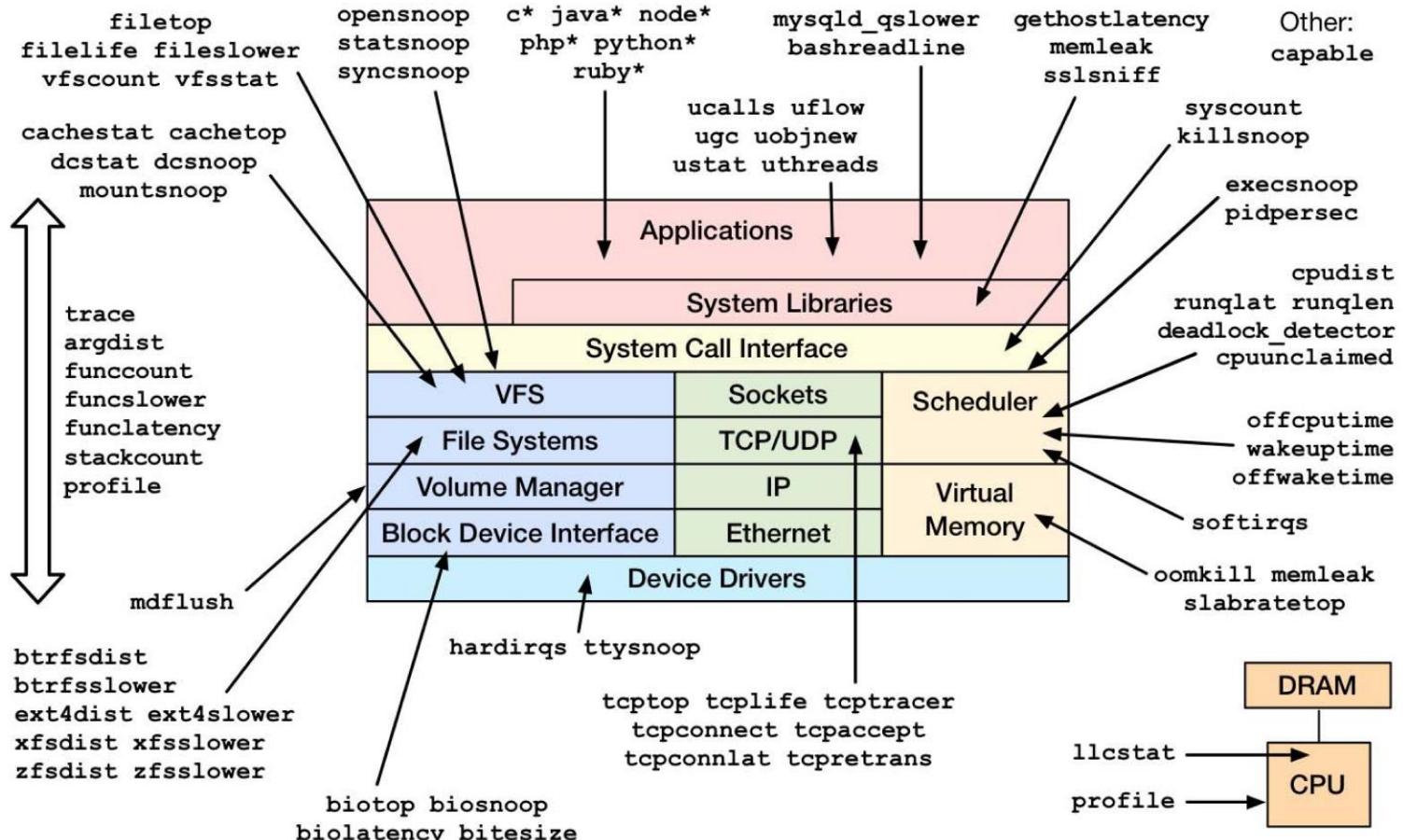
- Debugging interactions between user-level and kernel code nearly impossible without toolset that understands both and an instrument their actions
- BCC (BPF Compiler Collection) is a rich toolkit providing tracing features for Linux
 - See also the original DTrace
- For example, disksnoop.py traces disk I/O activity

TIME(s)	T	BYTES	LAT(ms)
1946.29186700	R	8	0.27
1946.33965000	R	8	0.26
1948.34585000	W	8192	0.96
1950.43251000	R	4096	0.56
1951.74121000	R	4096	0.35

- Many other tools (next slide)

Linux bcc/BPF Tracing Tools

Linux bcc/BPF Tracing Tools



Cheers!