VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SWE3001-Operating Systems

Prepared By
Dr. L. Mary Shamala
Assistant Professor
SCOPE/VIT

Objectives

- To describe the services an operating system provides to users, processes, and other systems.
- To understand the design of the operating system.
- To understand the structure and organization of the file system.
- To understand the process, synchronization, and scheduling.
- To understand the different approaches to memory management.
- To use system calls for managing processes, memory, and file system.

Syllabus

1. Introduction
2. Processes
3. Process Synchronization
4. CPU Scheduling
5. Memory Management
6. Virtual Memory
7. Mass Storage Structure
8. Applications of Operating Systems in Industry

Text Book

- A. Silberschatz, P.B. Galvin & G. Gagne, “ Operating System Concepts”, Ninth Edition, John Wiley, 2013.

Reference Books

1. William Stallings, “ Operating Systems: Internals and Design Principles”, Seventh Edition, Prentice Hall, 2012.
2. Andrew S. Tanenbaum, “Modern Operating Systems”, Third Edition, Prentice Hall, 2015

Module 1: Introduction

- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Operating-System Services
- User and Operating-System Interface
- System Calls
- Operating-System Generation
- System Boot

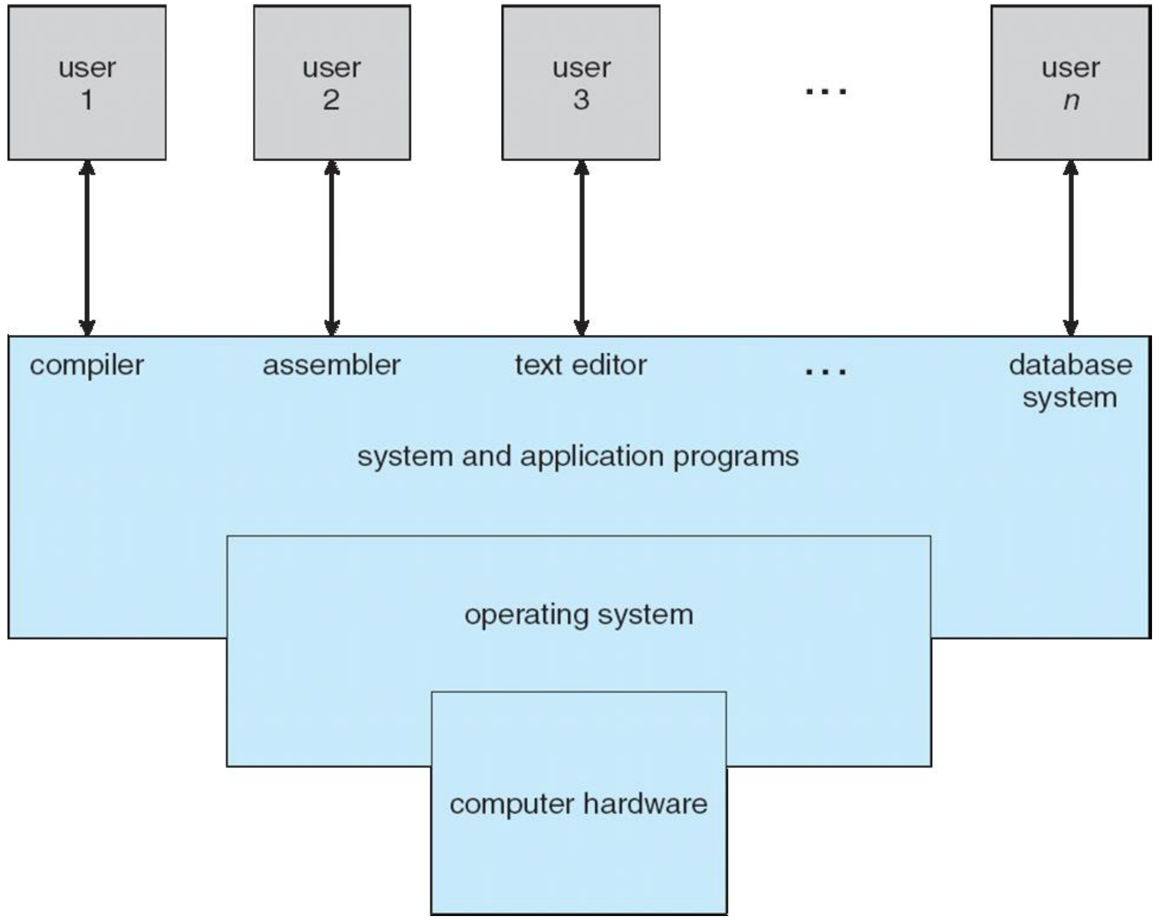
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
- 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
  - Ability to evolve

Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating Systems – controls and coordinates the use of resources
  - 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
- **Operating system controls and coordinates use of hardware among various applications and users**

Four Components of a Computer System



What Operating Systems Do?

- Depends on the point of view
  - Users
  - System

User View: OS as an **Extended machine** or **Virtual Machine**

- Users want convenience, **ease of use** and **good performance**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in home devices and automobiles

What Operating Systems Do…

System View

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Manages execution of programs to prevent errors and improper use of the computer

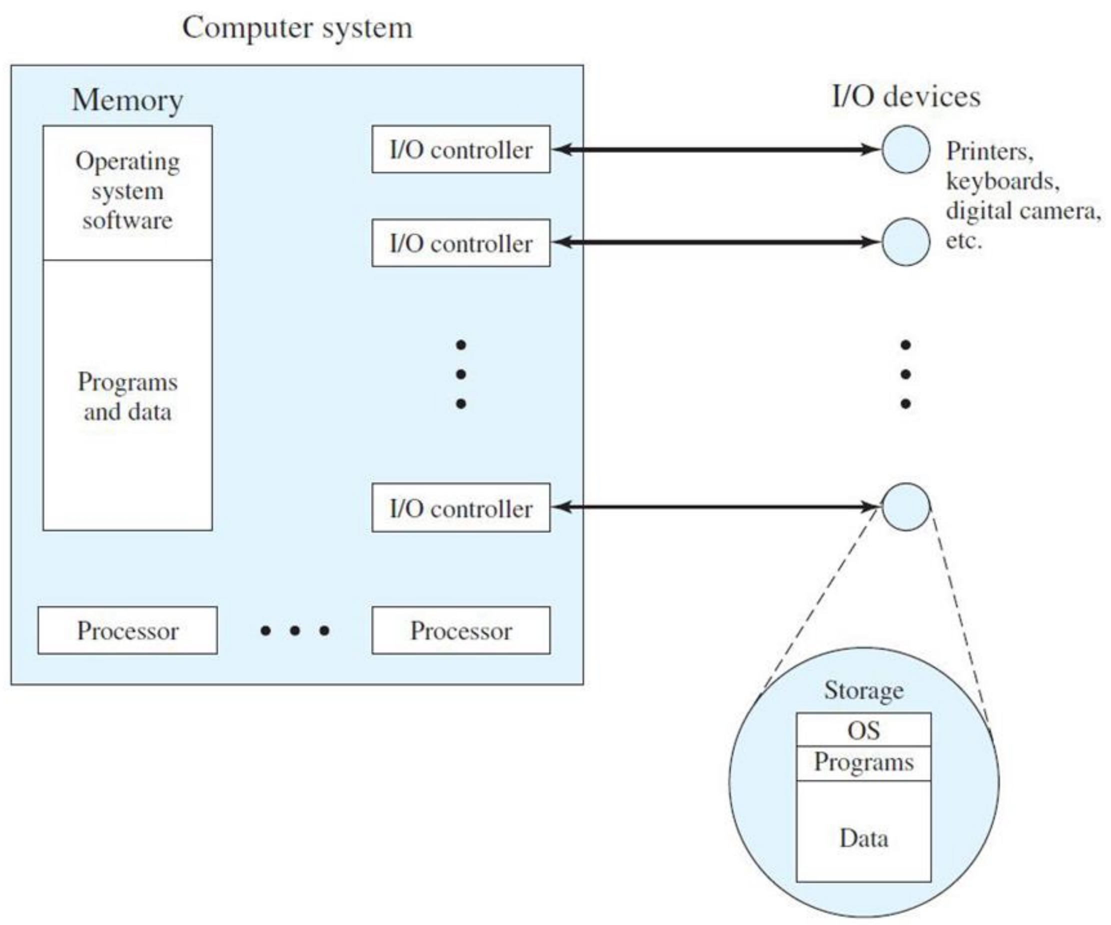
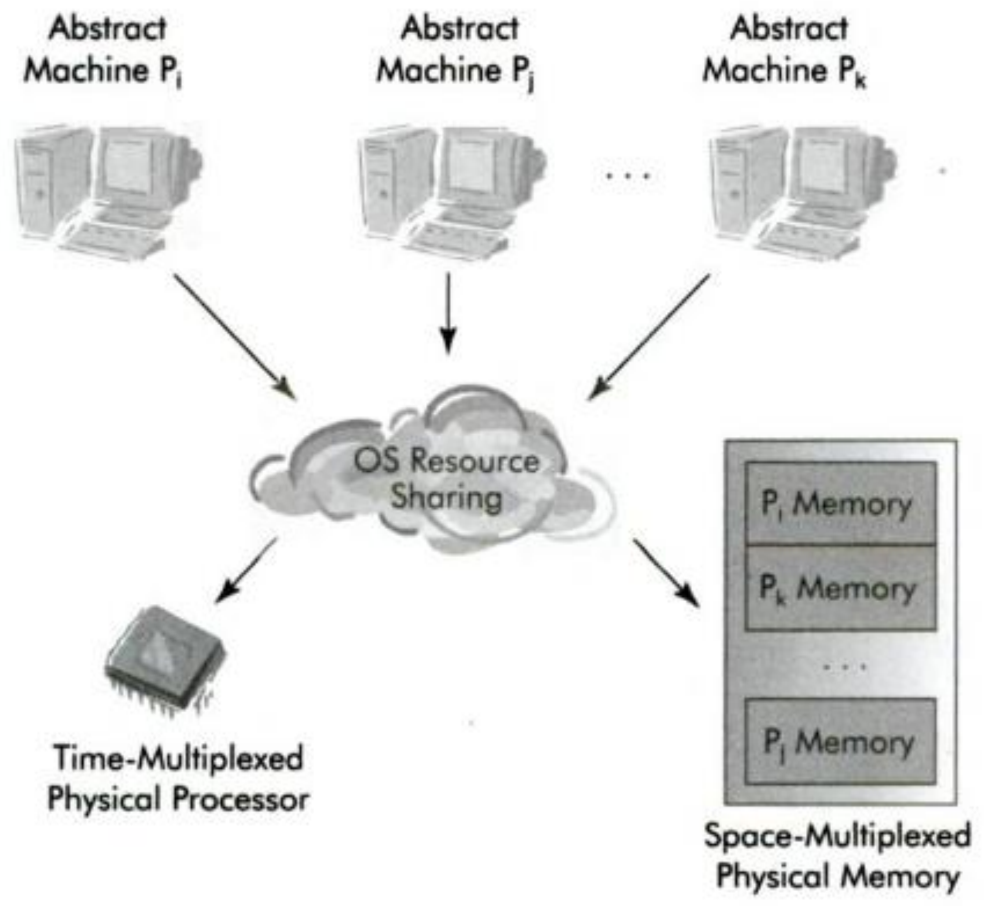


Figure 2.2 The Operating System as Resource Manager

What Operating Systems Do…

- Resource management is done in two different ways
  - Time multiplexed
    - Different programs or users take turns using a resource
    - Example: CPU, Printer
  - Space multiplexed
    - Each one gets part of the resource
    - Examples: Memory, Disk

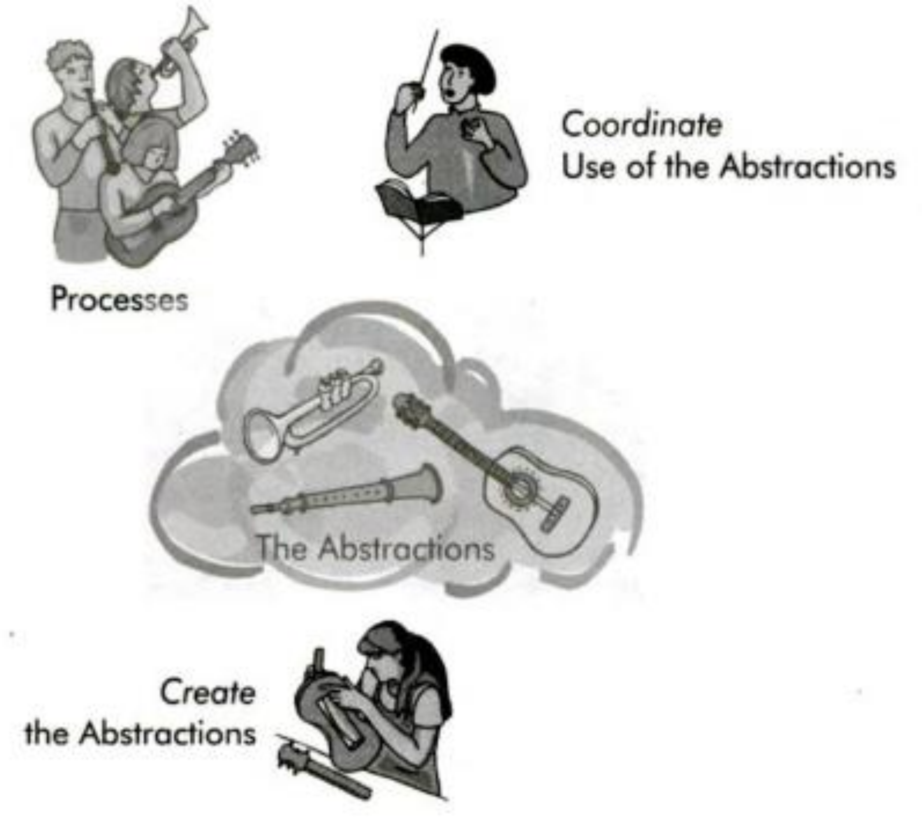


Operating System Definition

- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a system program (associated with the operating system), or
  - an application program.
- Mobile operating systems often include not only a core kernel but also **middleware**—a set of software frameworks that provide additional services to application developers.
- Examples of OS: Windows, UNIX, Linux, OS/2, MacOS, iOS, Android...

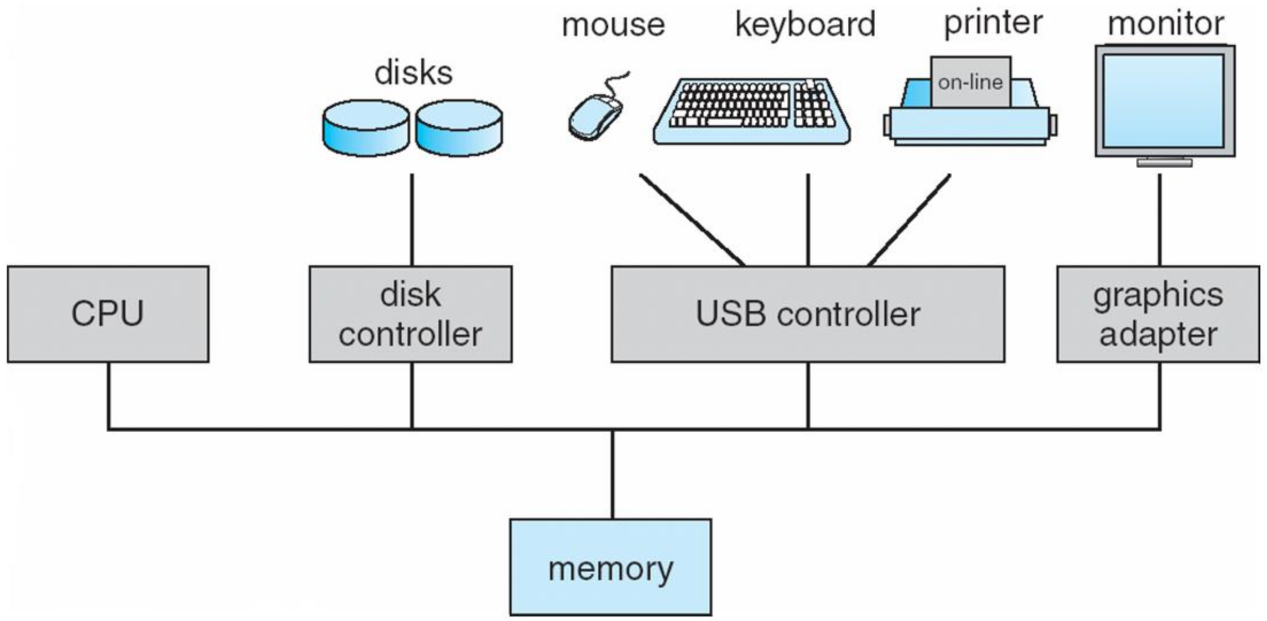
Functions of OS

- Two basic responsibilities
  1. Create an abstract machine environment
  2. Coordinate the use of components
- Functions
  - Process, thread, and resource management
  - Device management
  - Memory management
  - File management



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 Organization

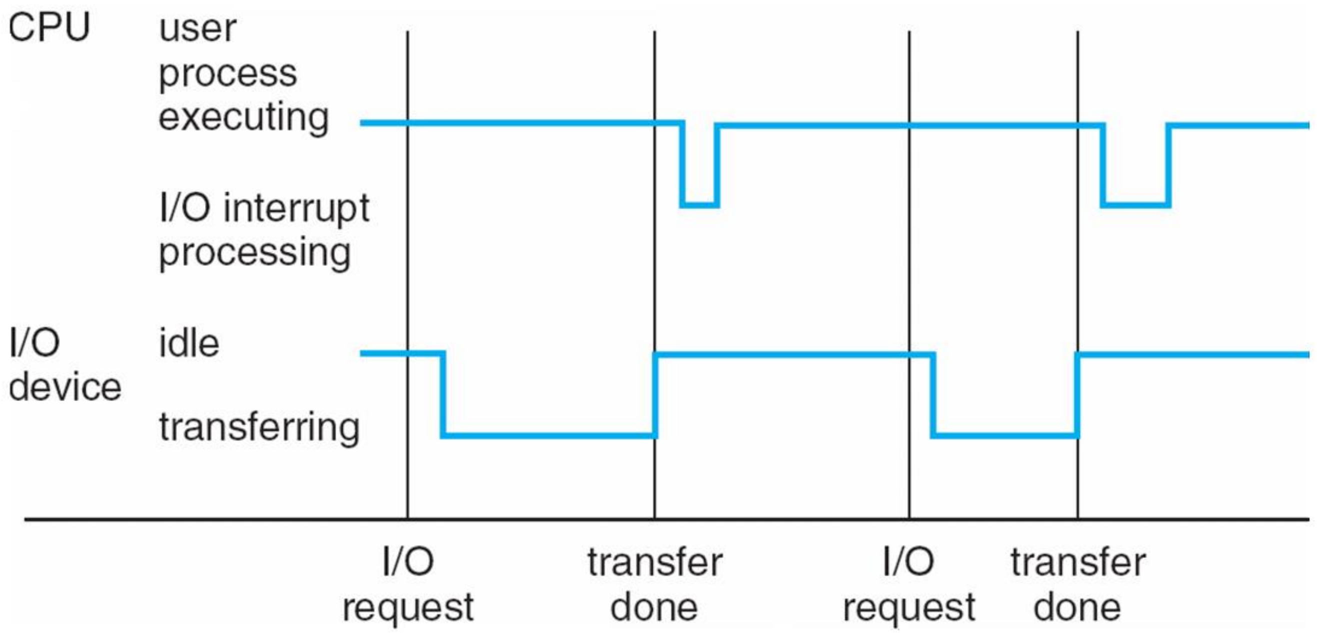
Computer Startup

- BIOS (Basic Input Output System) contains low-level I/O software
- **Bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

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
- 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

Interrupt Timeline



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 Handling

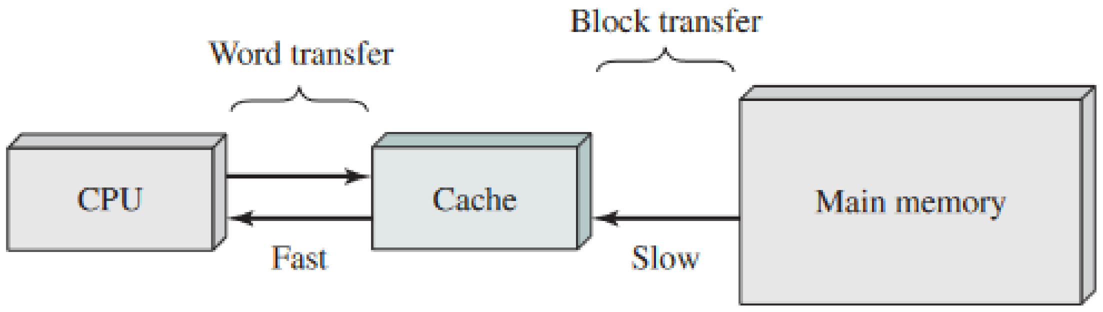
- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity.
  - Example : Magnetic disks
- Hard disks – 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
- **Solid-state disks** – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

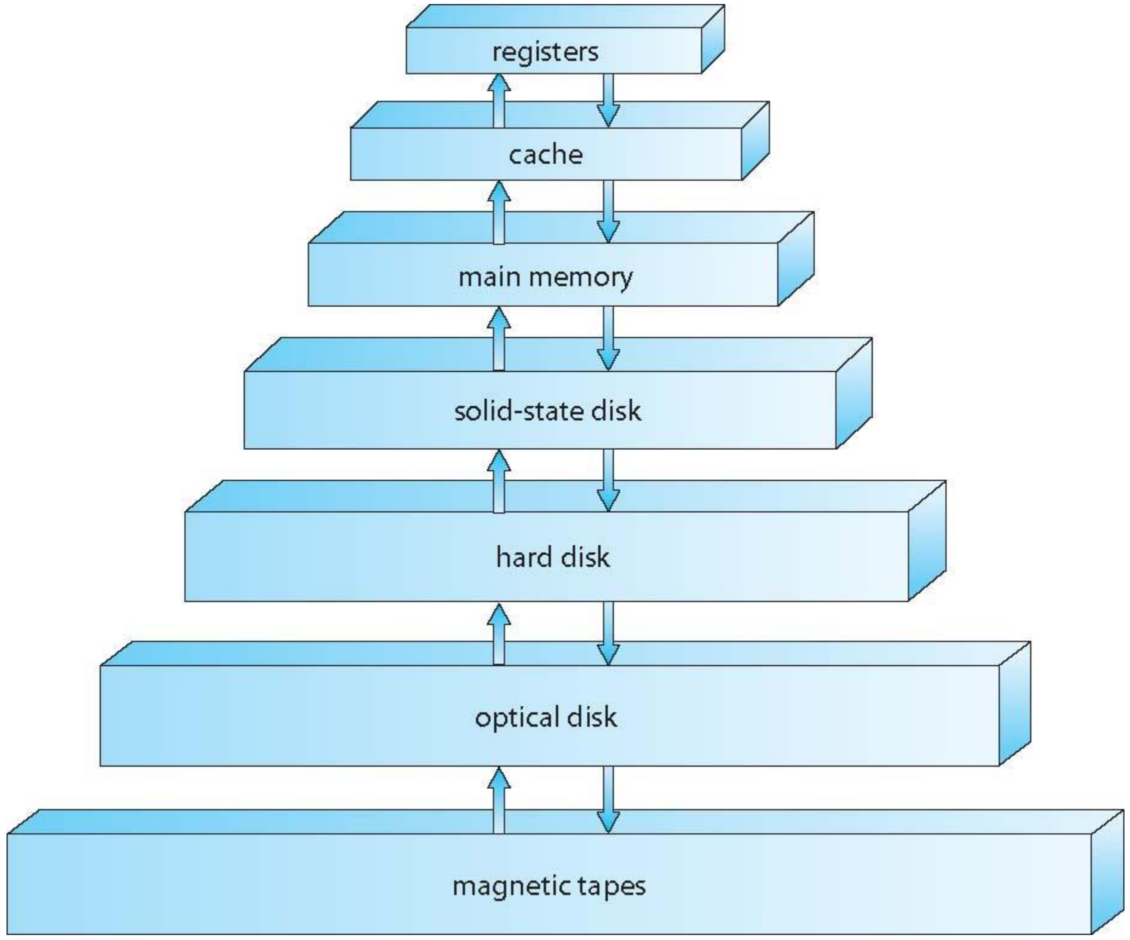
Storage Hierarchy

- 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

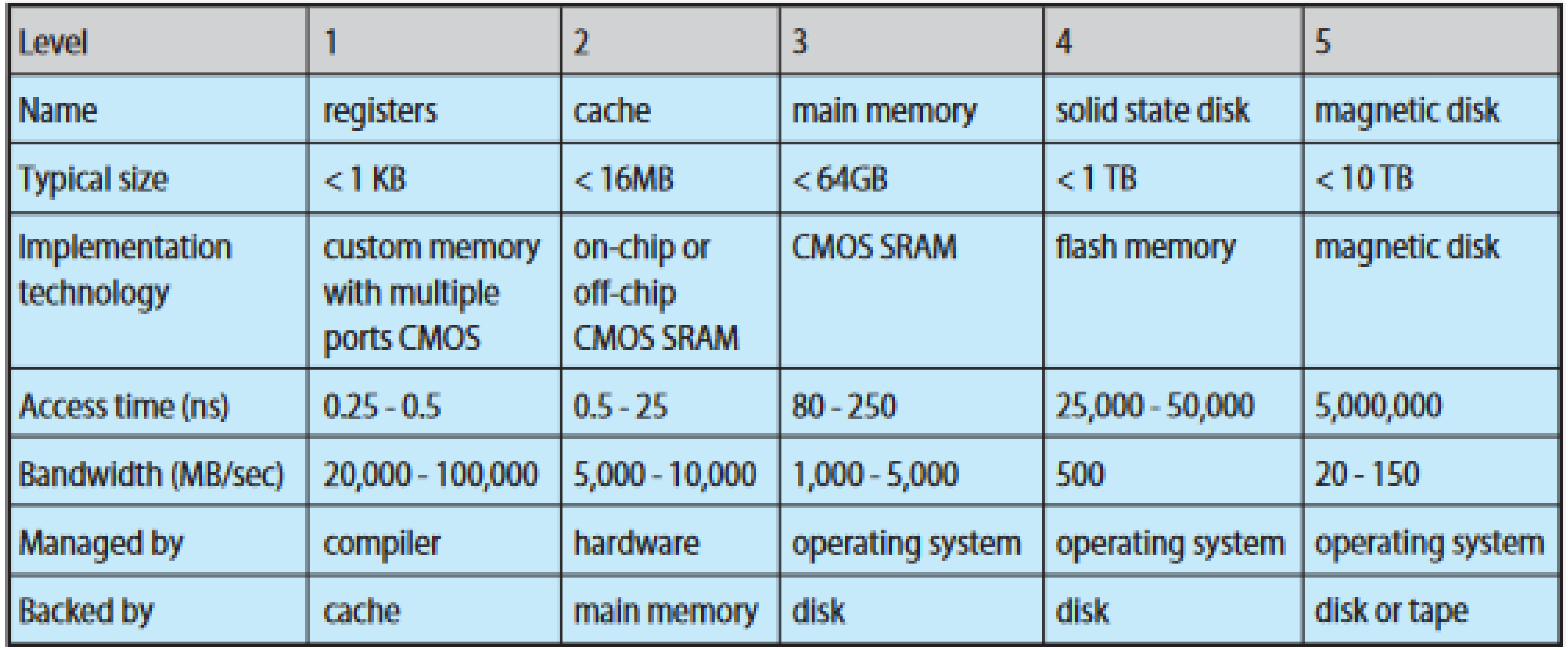


(a) Single cache

Storage-Device Hierarchy



Performance of various levels 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

I/O Structure

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus.
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- The device controller moves data between the peripheral devices and its local buffer storage.
- Operating systems have a **device driver** for each device controller.
- I/O is from the device to local buffer of controller

I/O Structure

- 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

Working of various components of Computer System

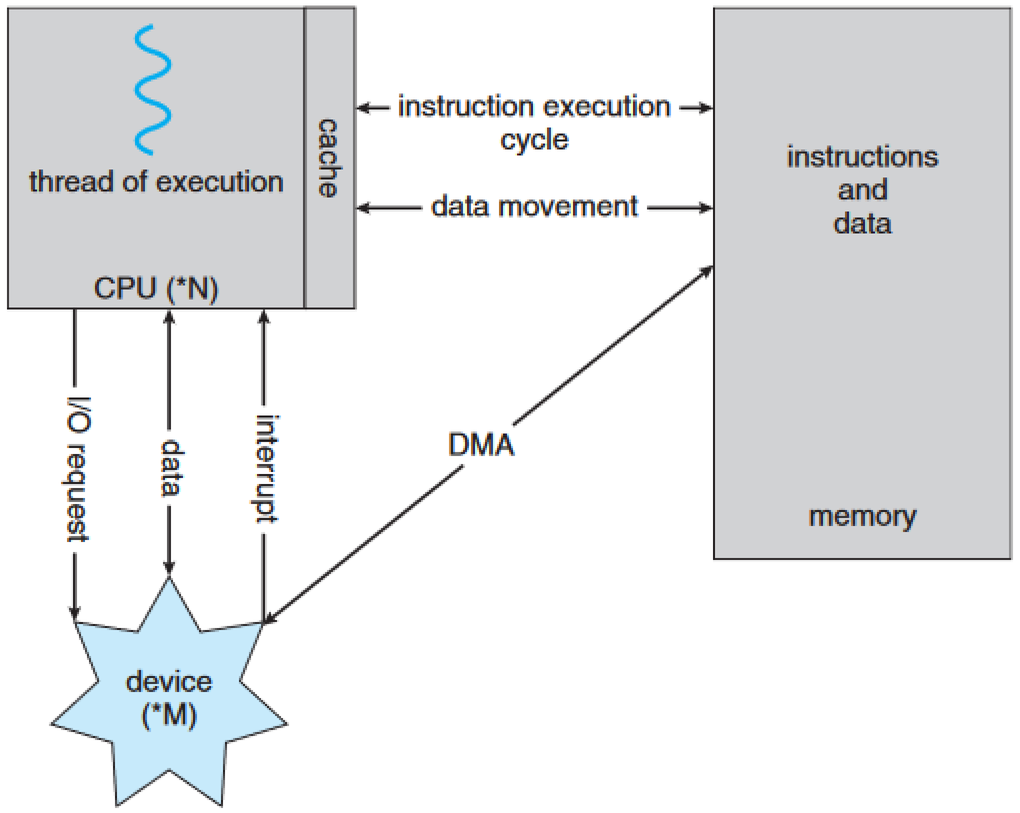


Figure 1.5 How a modern computer system works.

Computer-System Architecture

- According to the number of general-purpose processors, computer systems are categorized into:
  - Single-Processor Systems
  - Multiprocessor Systems
  - Clustered Systems

1. Single-Processor Systems

- Most Computer systems use a single general-purpose processor
- On a single- processor system, there is one main CPU capable of executing a general-purpose instruction set & instructions from user processes.
- Most single- processor systems have special-purpose processors as well.
  - Run a limited instruction set and do not run user processes.
  - Sometimes, they are managed by the operating system.

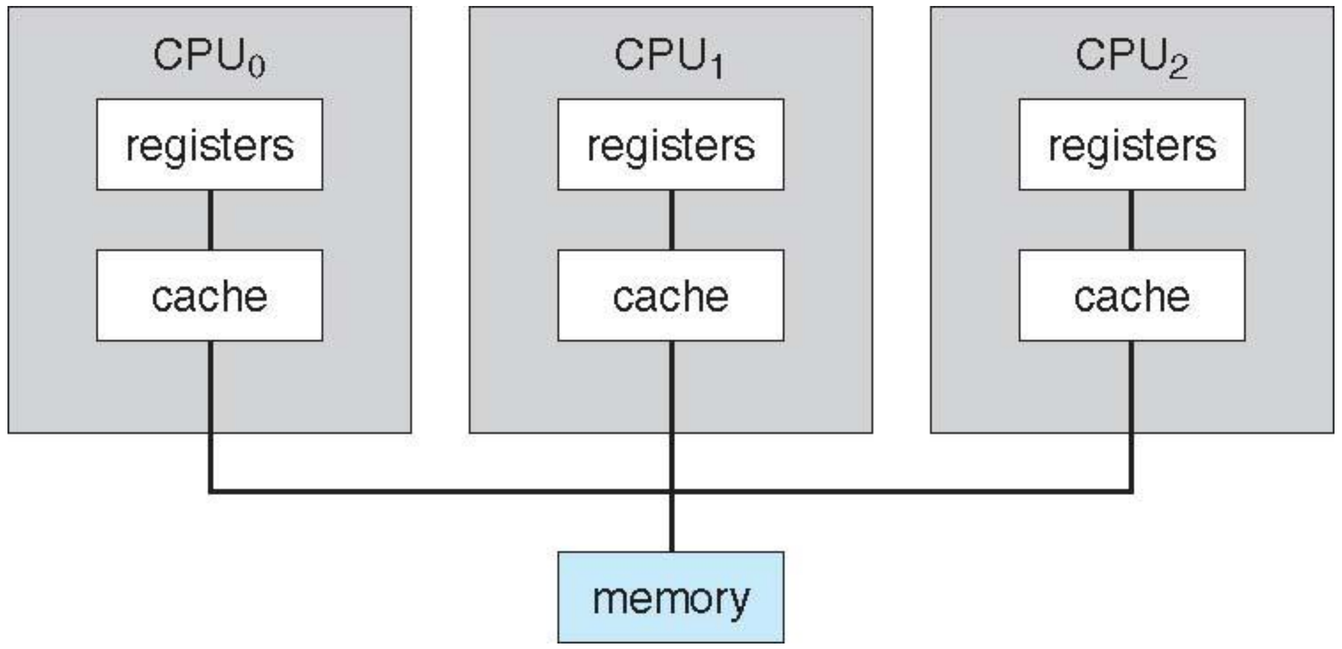
2. Multiprocessor Systems

- **Multiprocessors** systems growing in use and importance
- Also known as **parallel systems**, **tightly-coupled systems, multicore systems**
- Multiprocessor systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices
- Multiprocessing adds CPUs to increase computing power.
- Advantages include:
  1. **Increased throughput**
  2. **Economy of scale**
  3. **Increased reliability** – graceful degradation or fault tolerance
- The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation
- Fault-tolerant systems can suffer a failure of any single component but still continue operation.

Multiprocessor Systems…

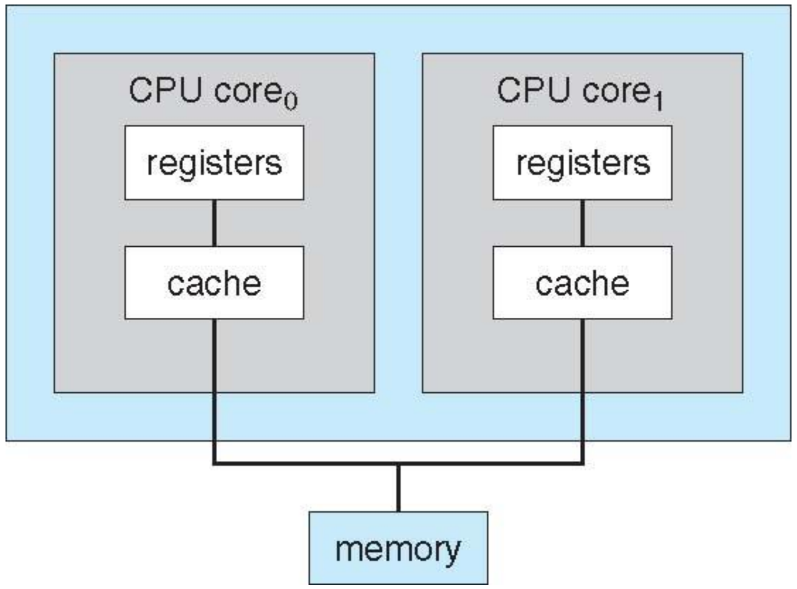
- Two types:
  1. **Asymmetric Multiprocessing** – each processor is assigned a special task.
     - A master processor controls the system
     - the other processors either look to the master for instruction or have predefined tasks.
     - This scheme defines a Master–Slave relationship.
     - The master processor schedules and allocates work to the slave processors
  2. **Symmetric Multiprocessing(SMP)** – each processor performs all tasks
     - all processors are peers; no boss–worker relationship exists between processors
- The difference between symmetric and asymmetric multiprocessing may result from either hardware or software.
  - Special hardware can differentiate the multiple processors or the software can be written to allow only one boss and multiple workers.

Symmetric Multiprocessing Architecture



A Dual-Core Design

- Multi-chip and **multicore-** multiple computing cores on a single chip
- More efficient than multiple chips with single cores
- One chip with multiple cores uses significantly less power than multiple single-core chips
- Well suited for server systems



Blade Servers

- Multiple processor boards, I/O boards, and networking boards are placed in the same chassis.
- Each blade-processor board boots independently and runs its own operating system.
- Some blade-server boards are multiprocessors as well.
- These servers consist of multiple independent multiprocessor systems.

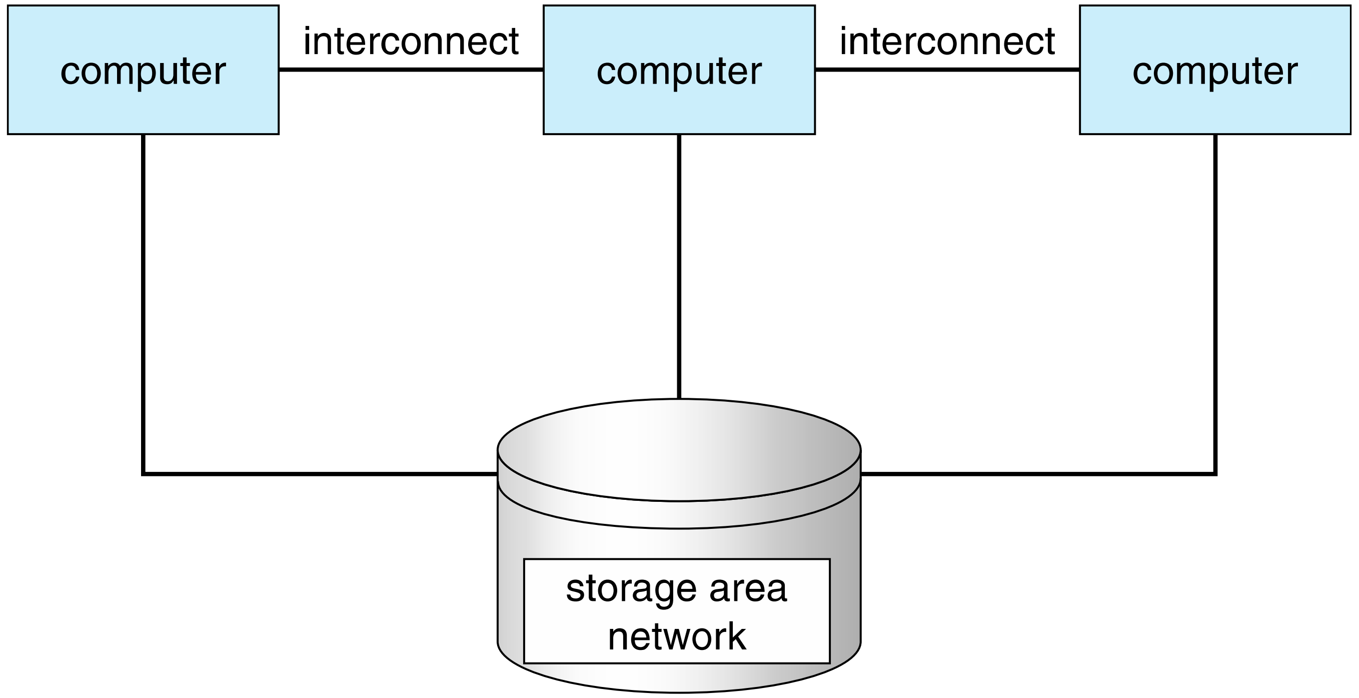
3.Clustered Systems

- Like multiprocessor systems, but multiple systems working together.
- Two or more individual systems or nodes are joined together. Such systems are considered **loosely coupled**.
- Each node may be a single processor system or a multicore system
- Clustered computers share storage and are closely linked via a local-area network LAN or a faster interconnect, such as InfiniBand
- Provides a **high-availability** service which survives failures
- Clustering can be structured into:
  - **Asymmetric clustering** has one machine in hot-standby mode
  - **Symmetric clustering** has multiple nodes running applications, monitoring each other

3.Clustered Systems…

- Some clusters are for **high-performance computing (HPC)**
  - Applications must be written to use **parallelization**
- Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN)
- Parallel clusters allow multiple hosts to access the same data on shared storage.
  - To provide this shared access, the system must also supply access control and locking to ensure that no conflicting operations occur.
  - **Distributed lock manager** (**DLM**) to avoid conflicting operations
- Many improvements in cluster technology by sharing storage via a **storage-area network (SAN)**

Clustered Systems



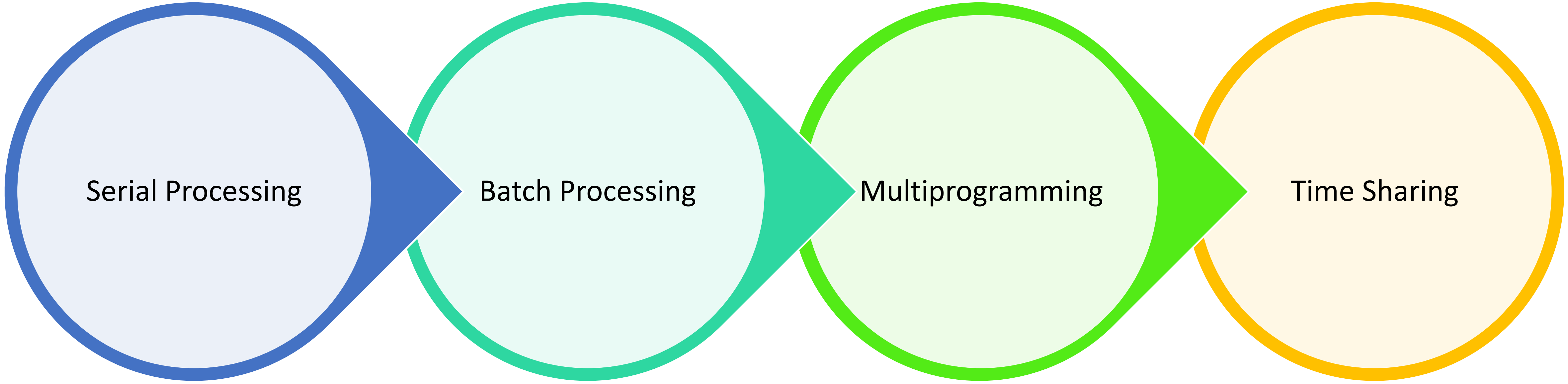
History of Operating Systems

Generation	Year	Electronic Device used	Types of OS and Devices
First	1945-55	Vacuum Tubes	Plug Boards
Second	1955-65	Transistors	Batch Systems
Third	1965-80	Integrated Circuits(Ics)	Multiprogramming
Fourth	1980- 90	Large Scale Integration	Personal Computers
Fifth	Since 1990	Handheld Computers	Mobile Systems

Evolution of Operating System

- Serial Processing
- Batch systems
- Time Sharing Systems
- Personal Computers and workstations
- Embedded Systems
- Network and Distributed Systems
- Mobile Systems

Operating System Strategies/ Evolution of OS



Serial Processing

- No operating system
- Machines run from a console with display lights, toggle switches, input device, and printer
- Problems include:
  - Scheduling
  - Setup time

Batch Systems

- Early computers were extremely expensive
  - Important to maximize processor utilization
- Monitor
  - Software that controls the sequence of events
  - Batch jobs together
  - Program returns control to monitor when finished
- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

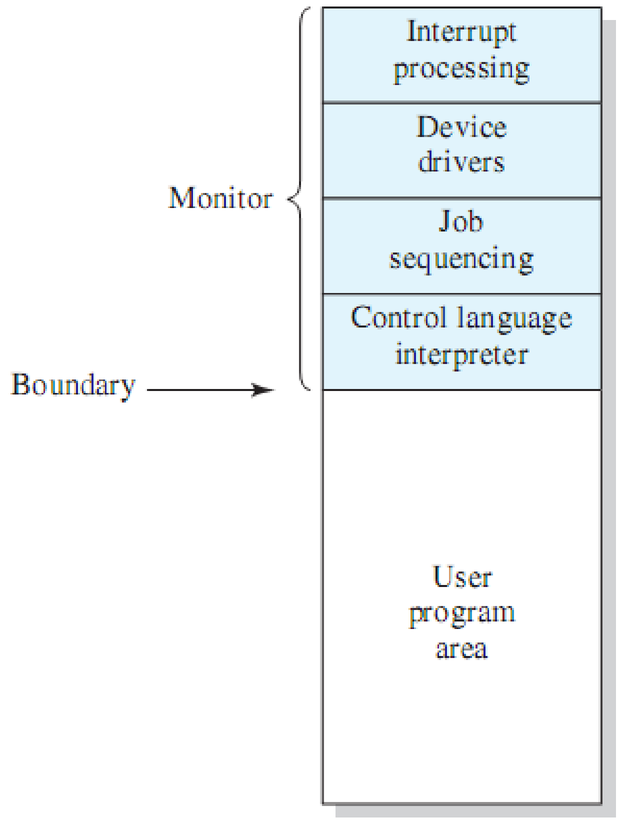
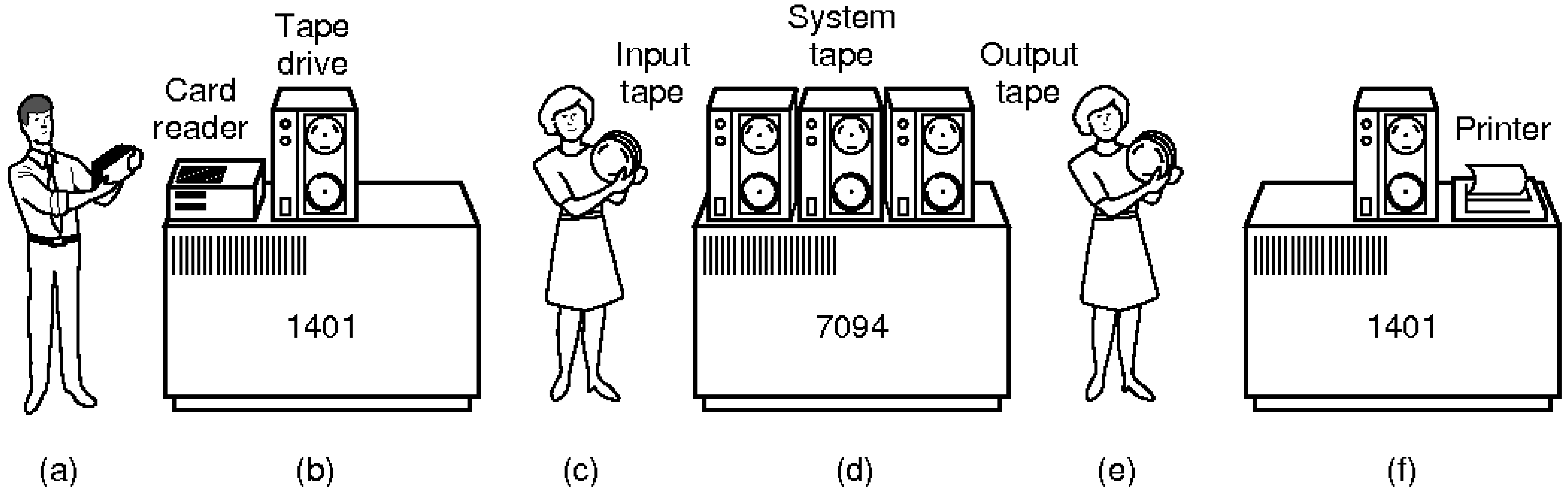


Figure 2.3 Memory Layout for a Resident Monitor

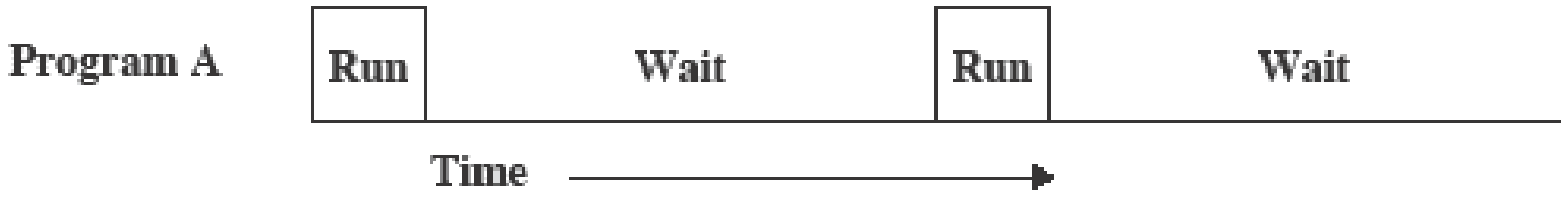
Batch Systems

- An early batch system



Multiprogrammed Batch Systems

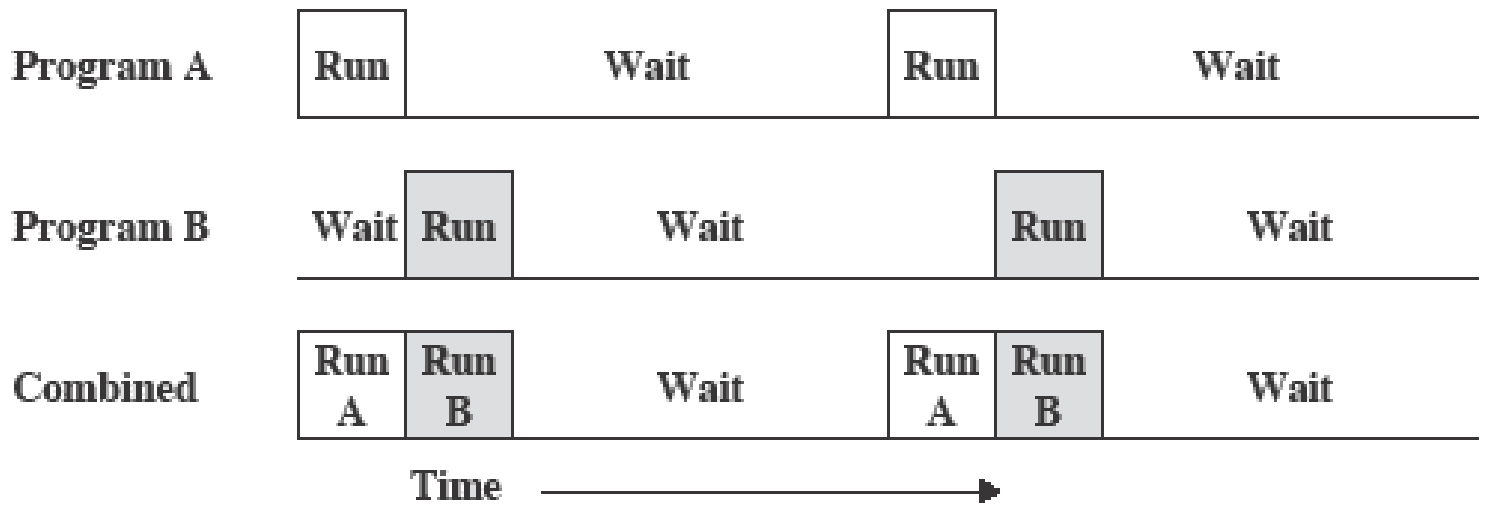
- CPU is often idle
  - Even with automatic job sequencing.
  - I/O devices are slow compared to processor
- Uniprogramming
  - Processor must wait for I/O instruction to complete before preceding



(a) Uniprogramming

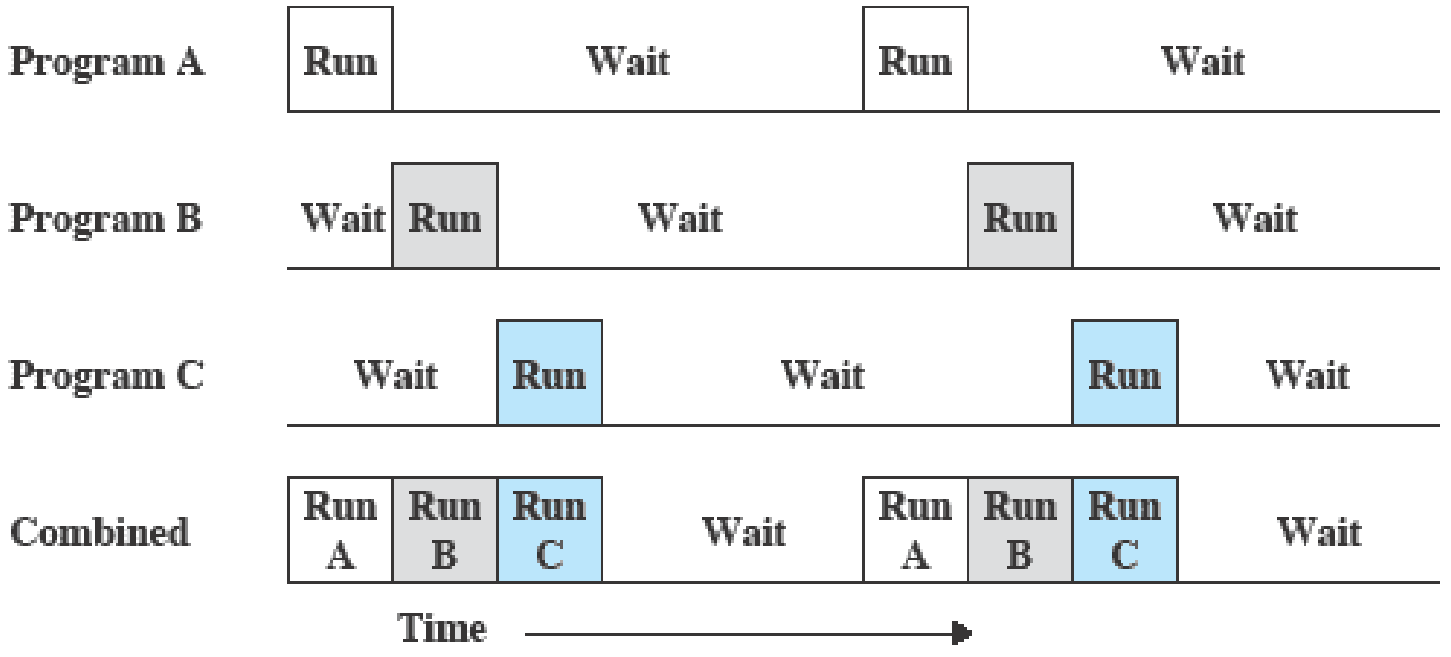
Multiprogrammed Batch Systems

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

Multiprogrammed Batch Systems…

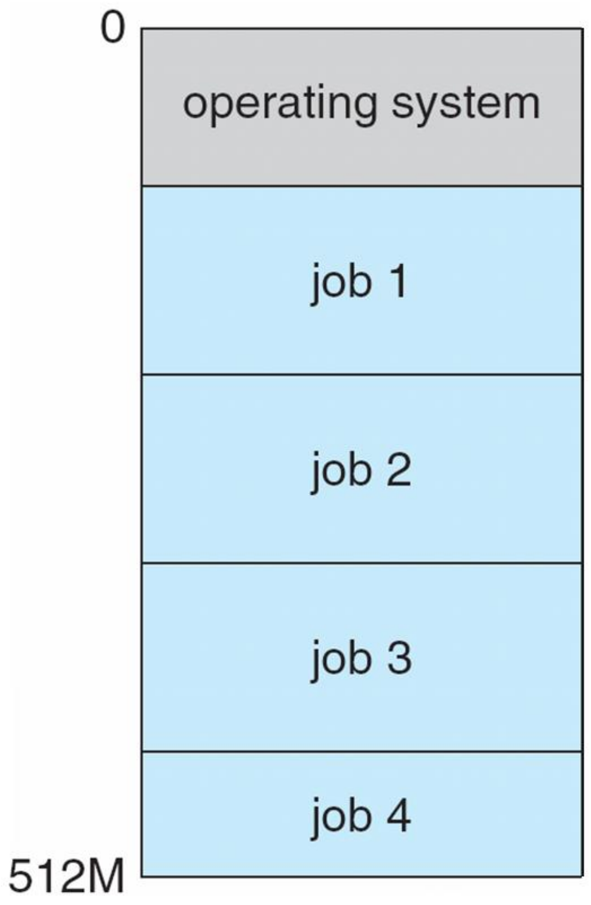


(c) Multiprogramming with three programs

Multiprogramming System

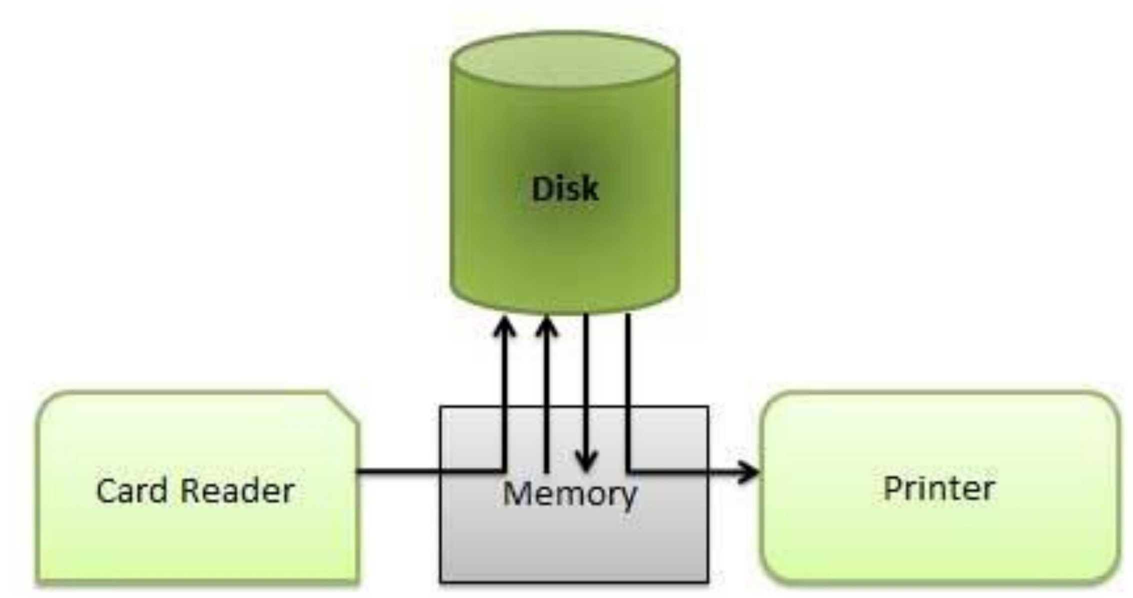
- **Multiprogramming** is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job is selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- Provide an environment in which system resources are utilized effectively
- They do not provide for user interaction with the computer system

Memory Layout for Multiprogrammed System



Spooling

- SPOOLing stands for Simultaneous Peripheral Operation On Line
- More than one I/O operation can be performed simultaneously



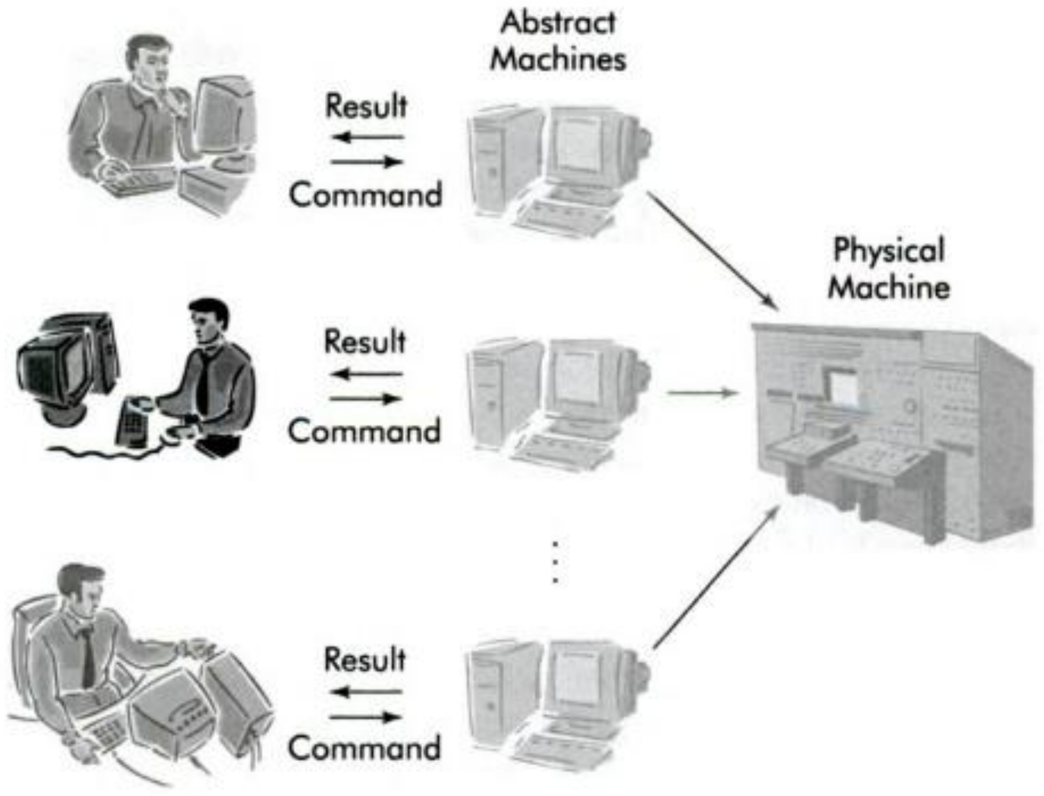
Spooling…

- Advantages
  - Since there is no interaction of I/O devices with CPU, the CPU need not wait for the I/O operation to take place
  - The CPU is kept busy most of the time and hence it is not in the idle state.
  - More than one I/O device can work simultaneously.

Time sharing System

- **Timesharing** (**multitasking**) is logical extension of multiprogramming.
- 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
- A time-shared operating system allows many users to share the computer simultaneously.
- Uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer
  - Each user has at least one program executing in memory ⇨**process**

Time Sharing Systems



Time sharing System…

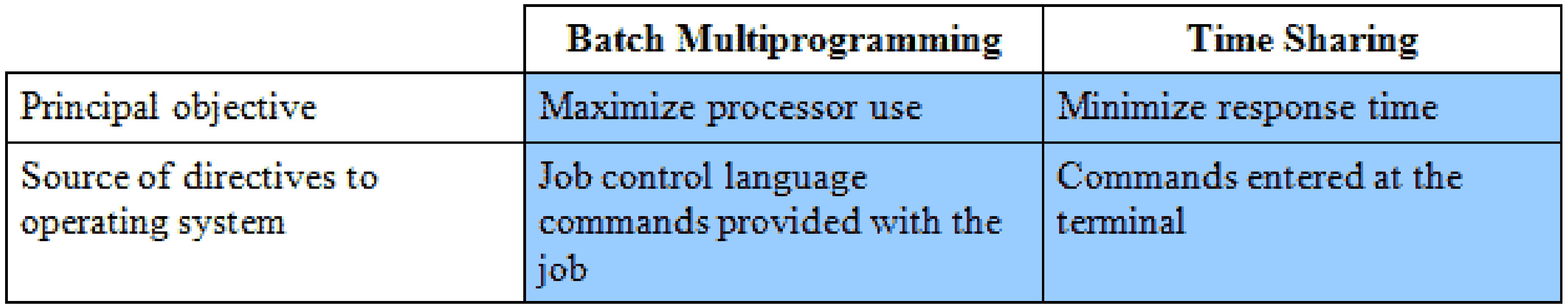
- If several jobs are ready to be brought into memory⇨ **Job scheduling**
- Having several programs in memory at the same time requires some form of **memory management**
- If several jobs ready to run at the same time ⇨ **CPU scheduling**
- To ensure reasonable response time:
  - If processes don't fit in memory, **swapping** moves them in and out of main memory to the disk
  - **Virtual memory** allows execution of processes not completely in memory
    - it enables users to run programs that are larger than actual physical memory.
    - it abstracts main memory into a large, uniform array of storage, separating logical memory as viewed by the user from physical memory.

Time sharing System…

- Time-sharing systems must
  - Provide a file system
  - Provides a mechanism for protecting resources from inappropriate use
  - To ensure orderly execution, the system must provide mechanisms for job synchronization and communication
  - Ensure that jobs do not get stuck in a deadlock, forever waiting for one another

Batch Multiprogramming vs. Time Sharing

Table 2.3 Batch Multiprogramming versus Time Sharing



	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Operating-System Operations

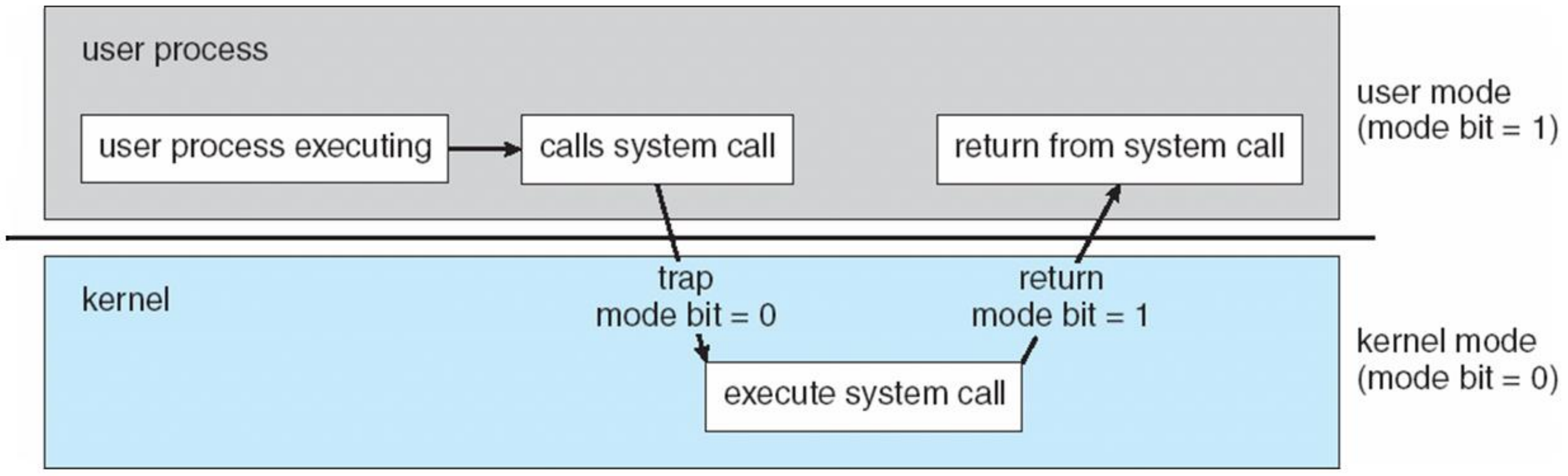
- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying each other or the operating system
- Without protection against these sorts of errors, either the computer must execute only one process at a time or all output must be suspect.
- A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

Operating-System Operations…

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - Kernel mode is also called **supervisor mode, system mode, or privileged mode.**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
  - i.e. **virtual machine manager** (**VMM**) mode for guest **VMs**

Dual Mode Operation

- Transition from user to kernel mode



User Mode Vs Kernel Mode

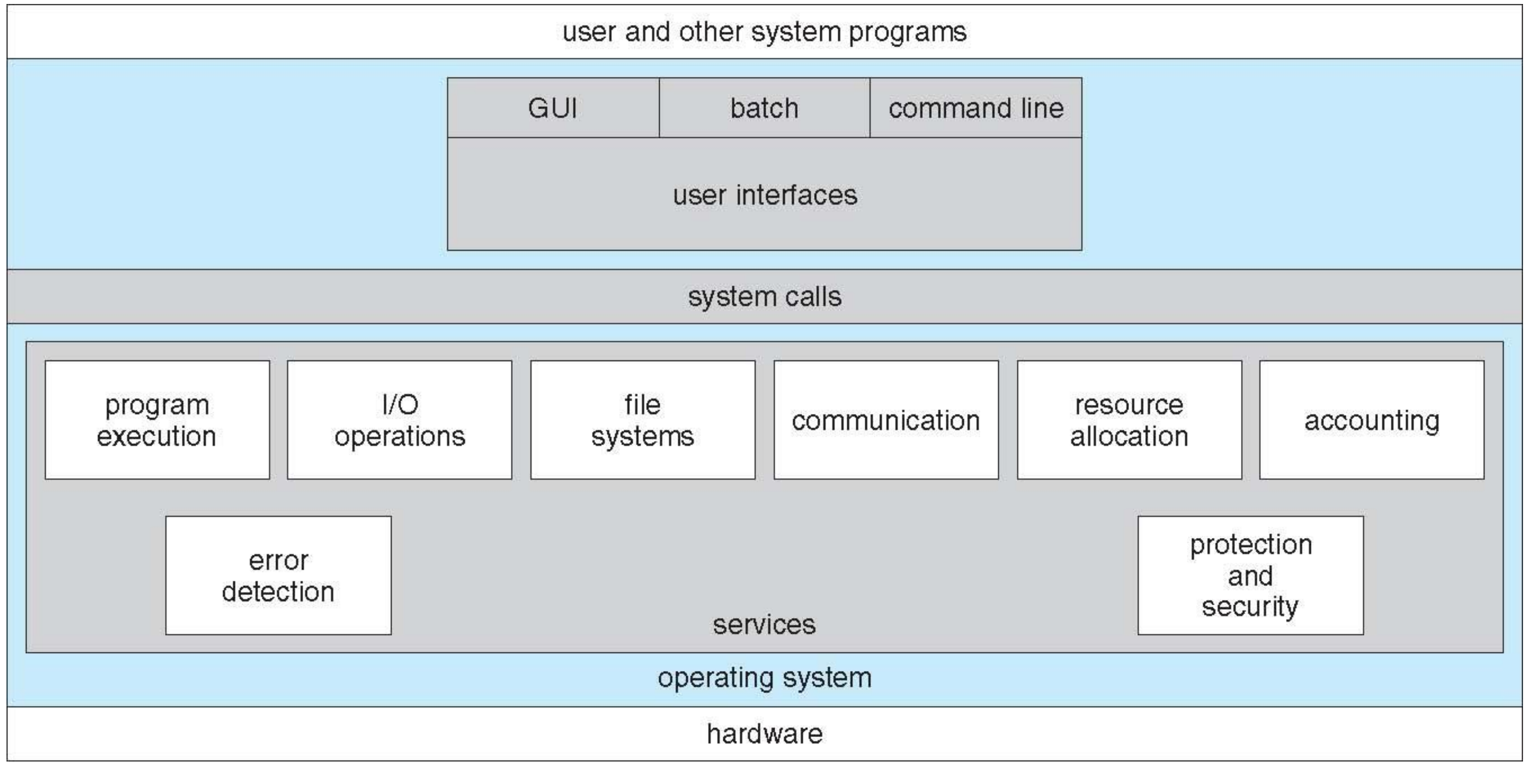
User Mode	Kernel Mode
User Program executes in this mode	Operating System executes in this mode
Certain memory area are protected from user’s use	Protected areas of memory may be accessed
Privileged instructions may not be executed	Privileged instructions may be executed

Operating-System Operations…

- **Timer** to prevent infinite loop / 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

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- A View of Operating System Services



Operating System Services…

- 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)**, **Batch Interface**
  - **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

Operating System Services…

- **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.
- **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…

- 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.
  - **Accounting -** 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

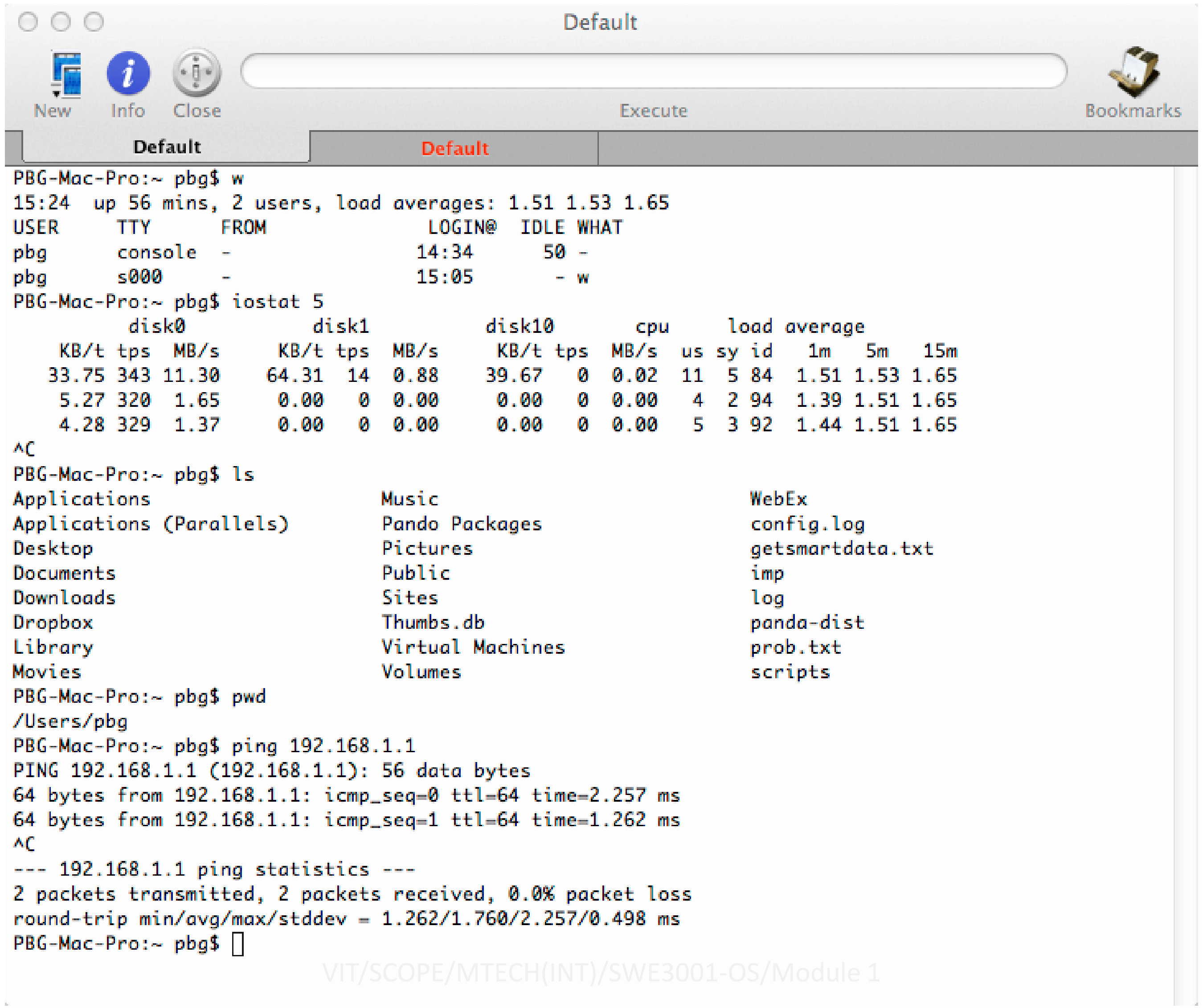
User and Operating System Interface

- There are several ways for users to interface with the operating system.
- Two fundamental approaches.
  - Command-line interface, or command interpreter, allows users to directly enter commands to be performed by the operating system.
  - The other allows users to interface with the operating system via a graphical user interface, or GUI.

User and Operating System Interface - CLI

- CLI or **command interpreter** 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
- Many of the commands given at this level manipulate files
- These commands can be implemented in two general ways.
  - **built-in**→the command interpreter itself contains the code to execute the command.
  - implements most commands through **system programs**
    - adding new features doesn't require shell modification
- Traditionally, UNIX systems have been dominated by command-line interfaces.

Bourne Shell Command Interpreter



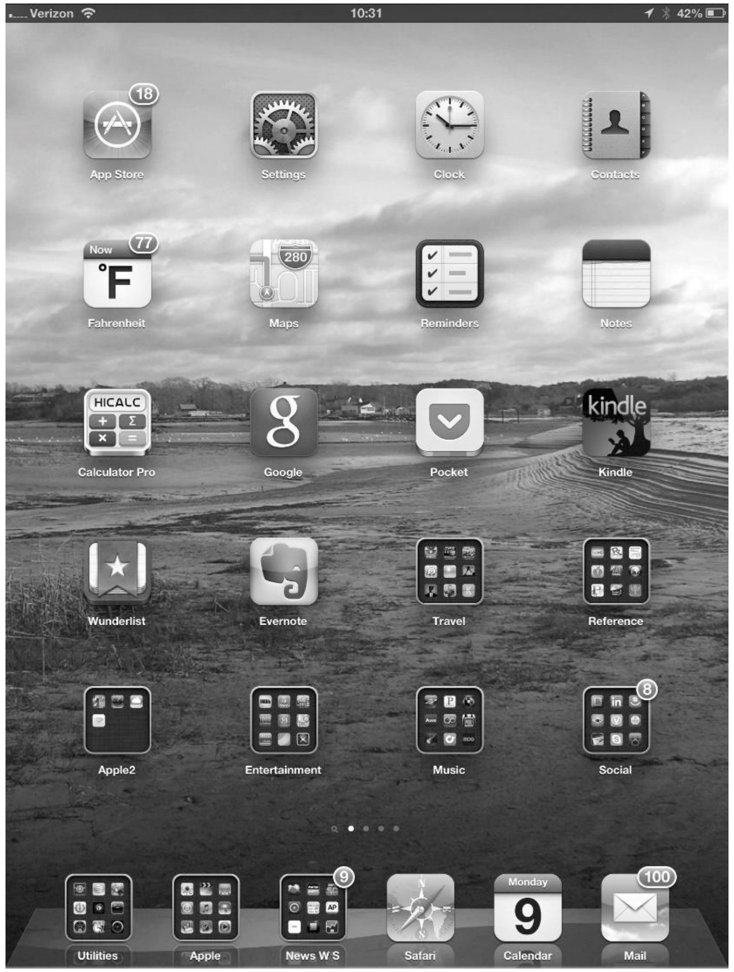
```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10        cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ 
```

User and Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, system functions, 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
- Example GUI interfaces
  - Common Desktop Environment (CDE) and X-Windows systems, are common on commercial versions of UNIX , such as Solaris and IBM's AIX system
  - K Desktop Environment (or KDE) and the GNOME desktop by the GNU project

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.



CLI vs GUI

Command Line Interface	Graphical User Interface
Console representation	Graphical representation
Difficult for beginners	Ease of use
Faster OS	OS is slower
Granular control	Lesser Control
Supports scripting or automation	Do not support scripting or automation

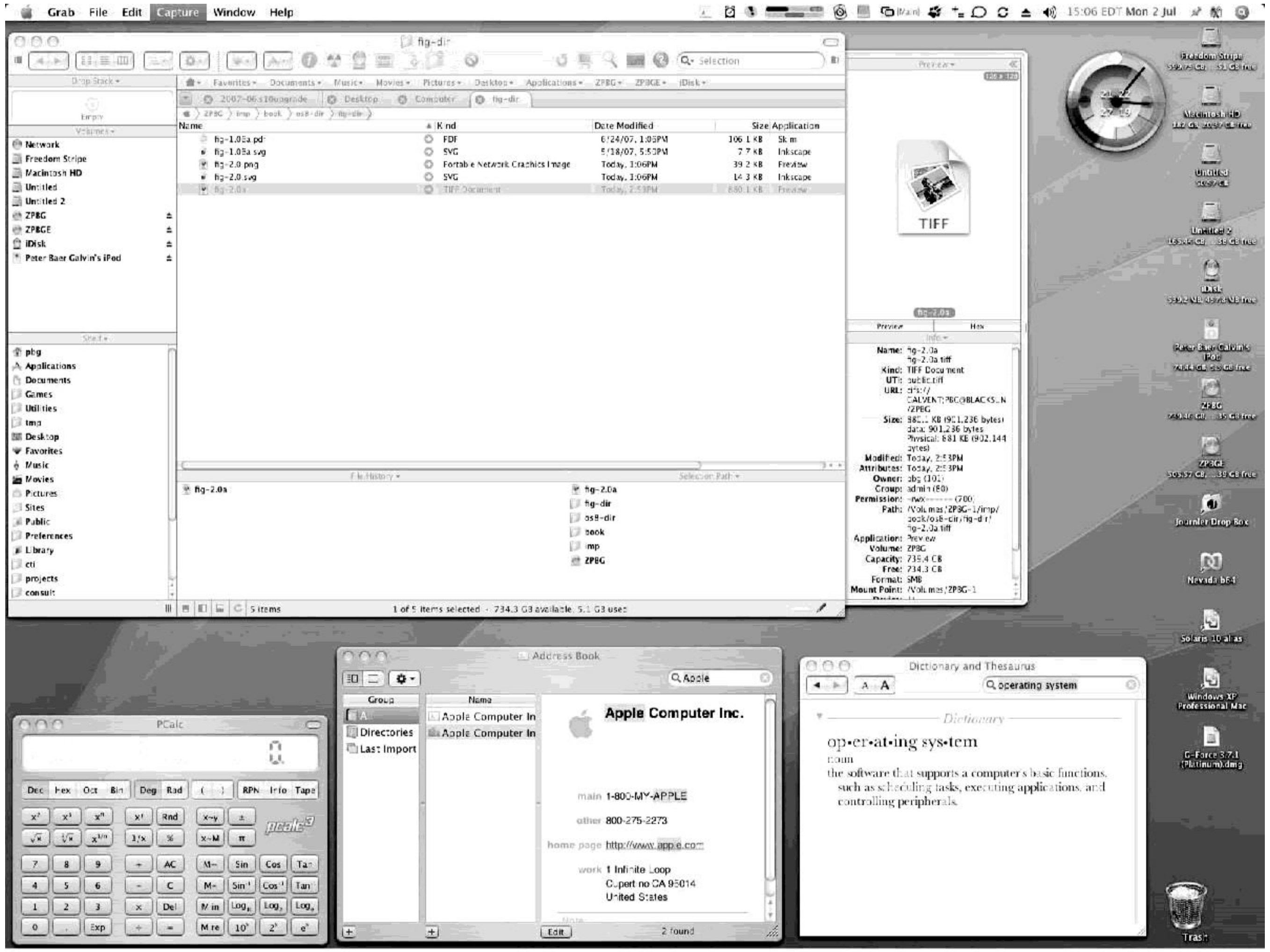
Choice of Interface

- The choice is mostly one of personal preference.
- **System administrators** and **power users** frequently use the command-line interface
- For them, it is more efficient, giving them faster access
- Command line interfaces usually make repetitive tasks easier ⇨ **Shell Scripts**
- Most Windows users are happy to use the Windows GUI environment
- The user interface can vary from system to system and even from user to user within a system.
- The design of a useful and friendly user interface is not a direct function of the operating system.

Choice of Interface

- 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)

The Mac OS X GUI



Name	Kind	Date Modified	Size	Application
fig-1.05a.pdf	PDF	6/24/07, 1:05PM	106.1 KB	Skim
fig-1.05a.svg	SVG	5/18/07, 5:53PM	7.7 KB	Inkscape
fig-2.0.png	Portable Network Graphics Image	Today, 1:06PM	39.2 KB	Preview
fig-2.0.svg	SVG	Today, 1:06PM	14.3 KB	Inkscape
fig-2.0a	TIFF Document	Today, 2:53PM	880.1 KB	Preview

5 items 1 of 5 items selected · 734.3 GB available, 5.1 GB used

Apple Computer Inc.
main 1-800-MY-APPLE
other 800-275-2273
home page http://www.apple.com
work 1 Infinite Loop
Cupertino CA 95014
United States

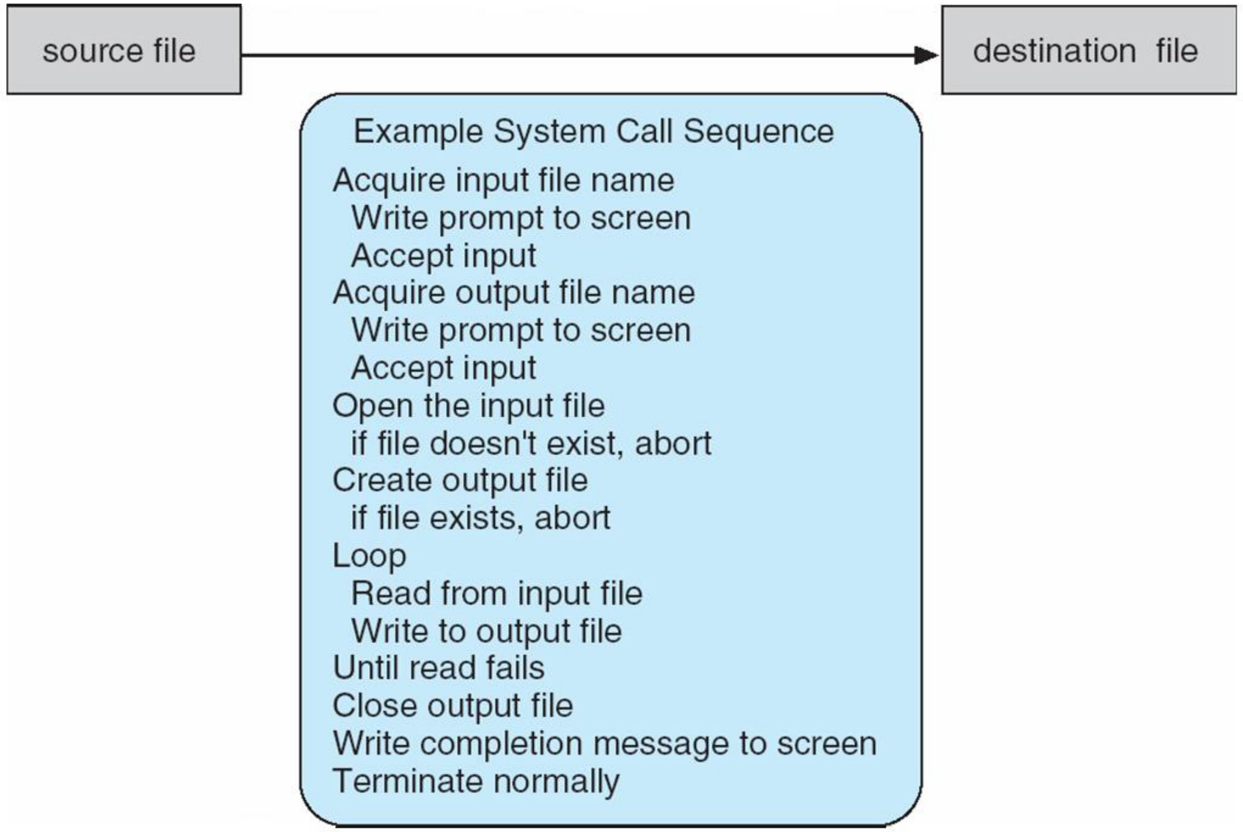
op•er•at•ing sys•tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

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)
    - Java API for the Java virtual

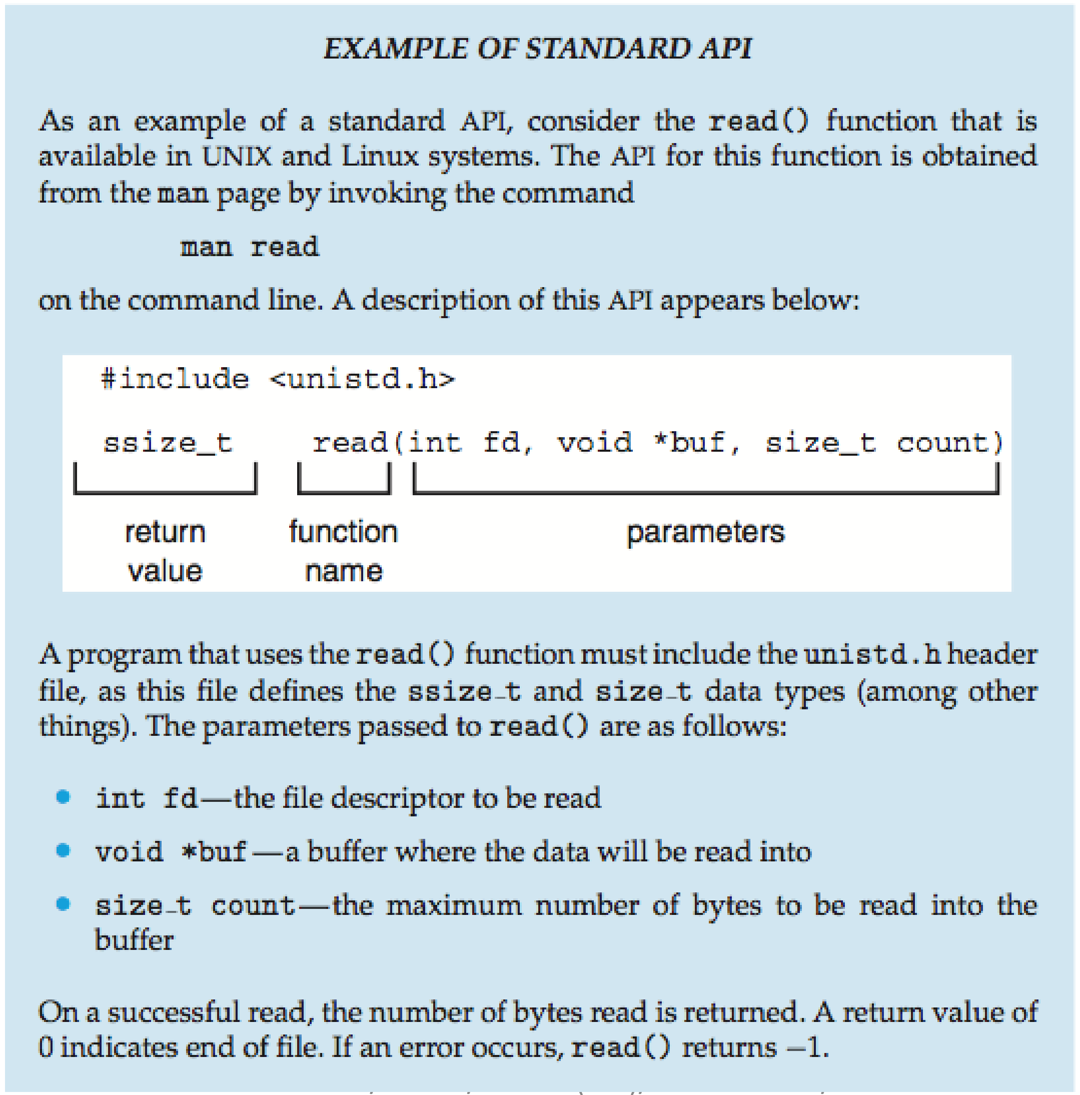
Example of System Calls

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



```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

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   function          parameters
 value      name
```

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:

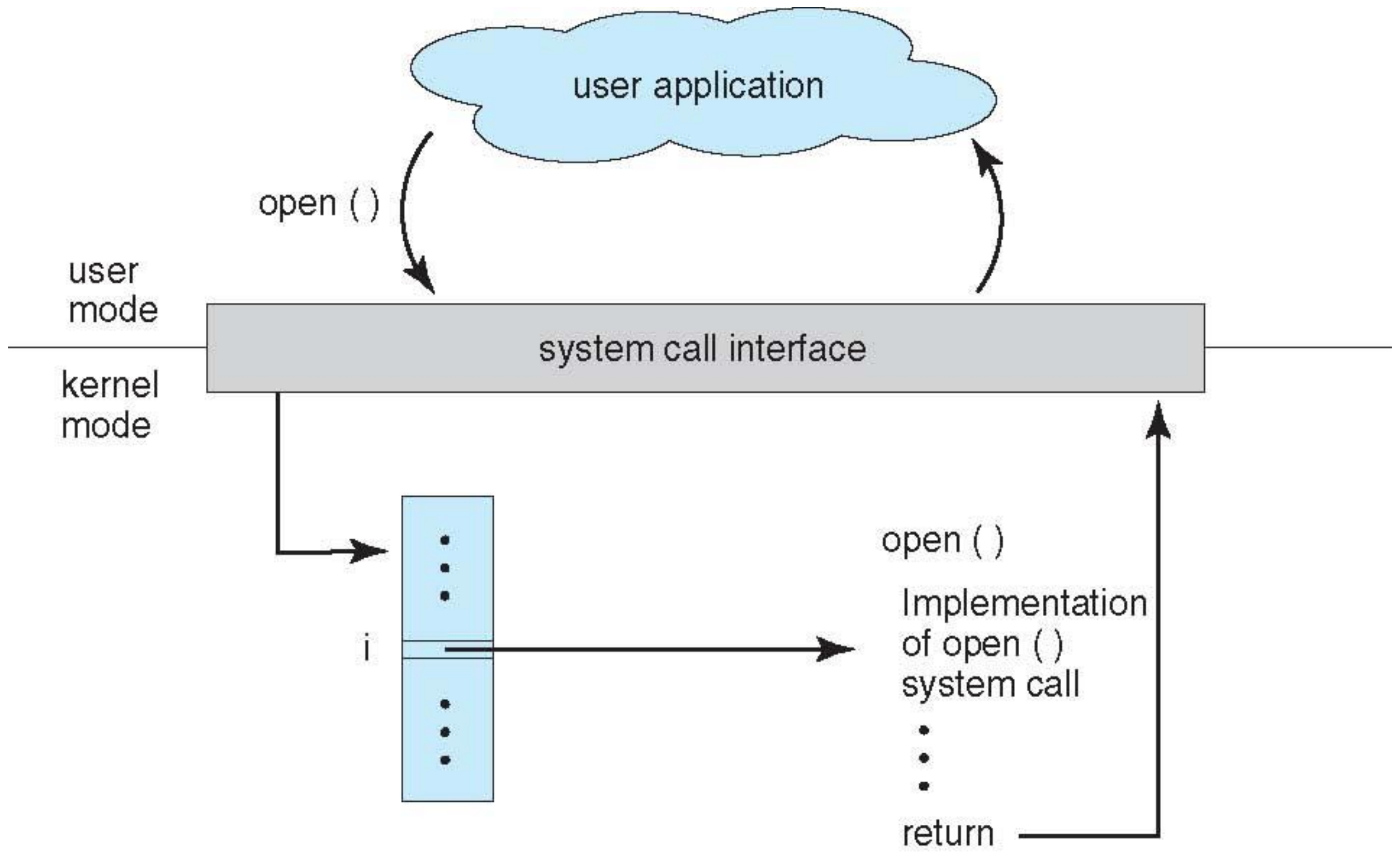
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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

- **System-call interface** is typically, a number 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)

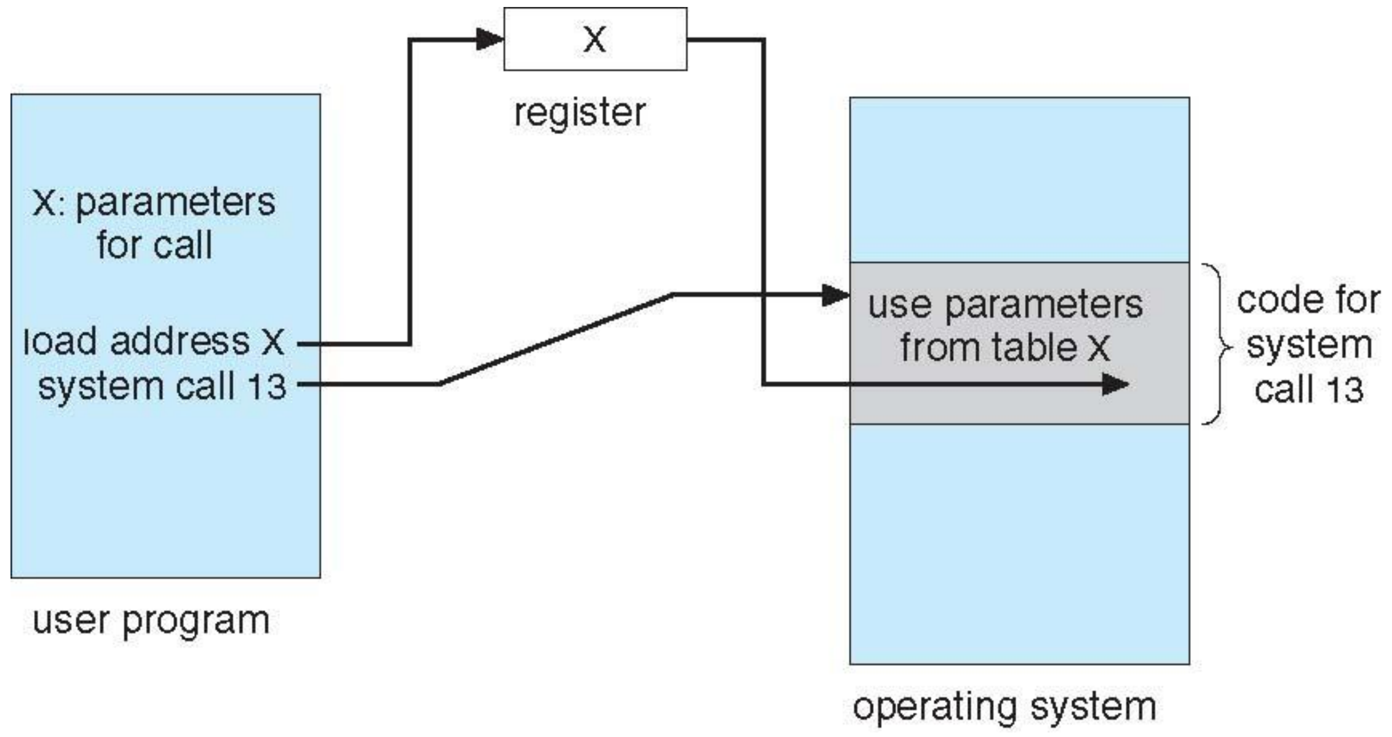
API – System Call – OS Relationship



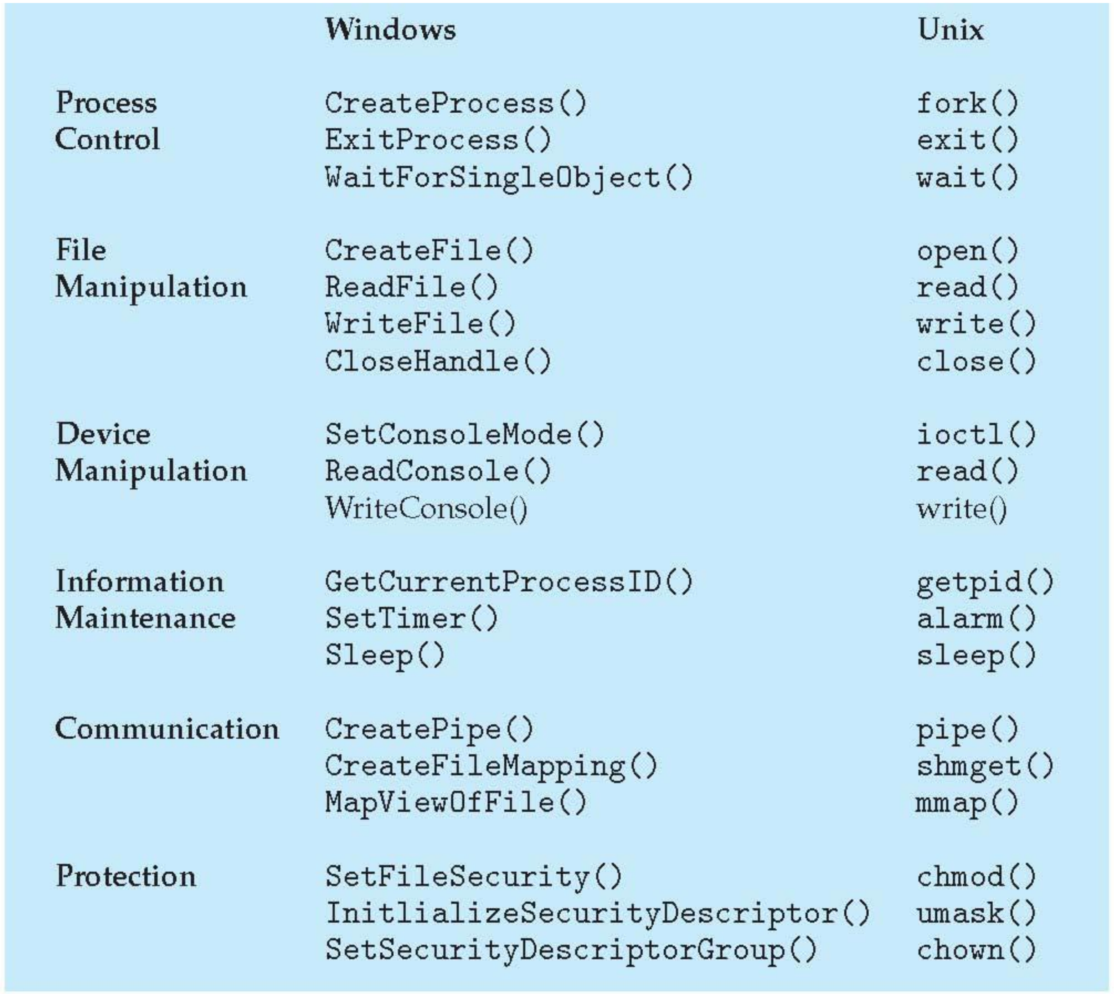
System Call Parameter Passing

- 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



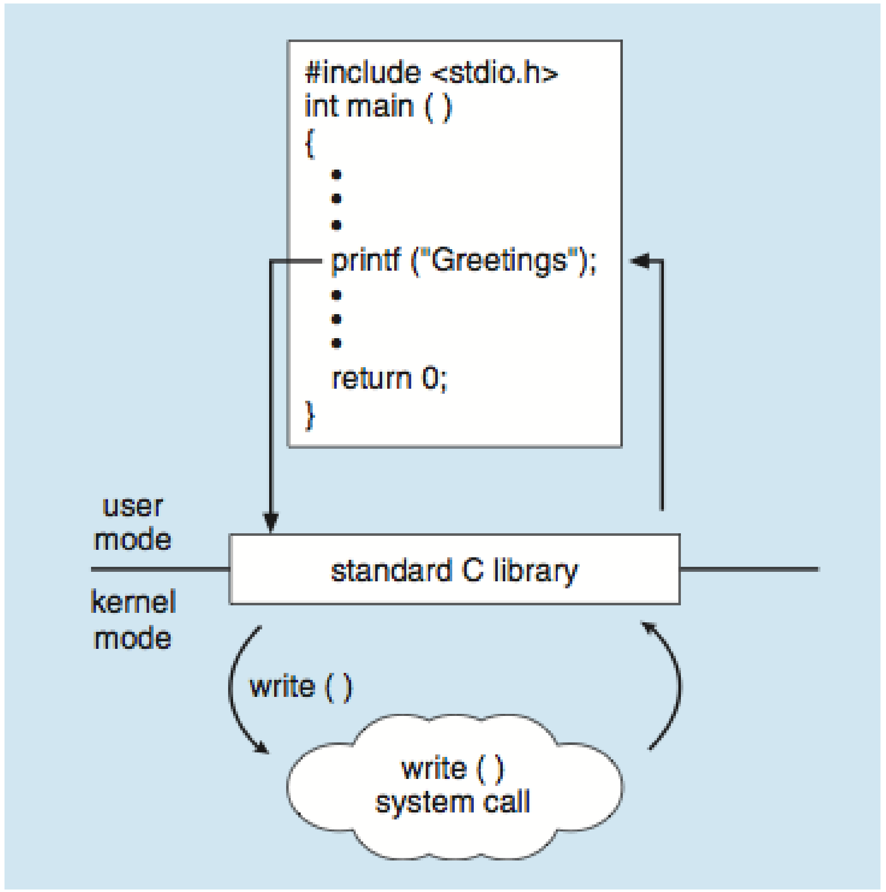
Examples of Windows and Unix System Calls



	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Standard C Library Example

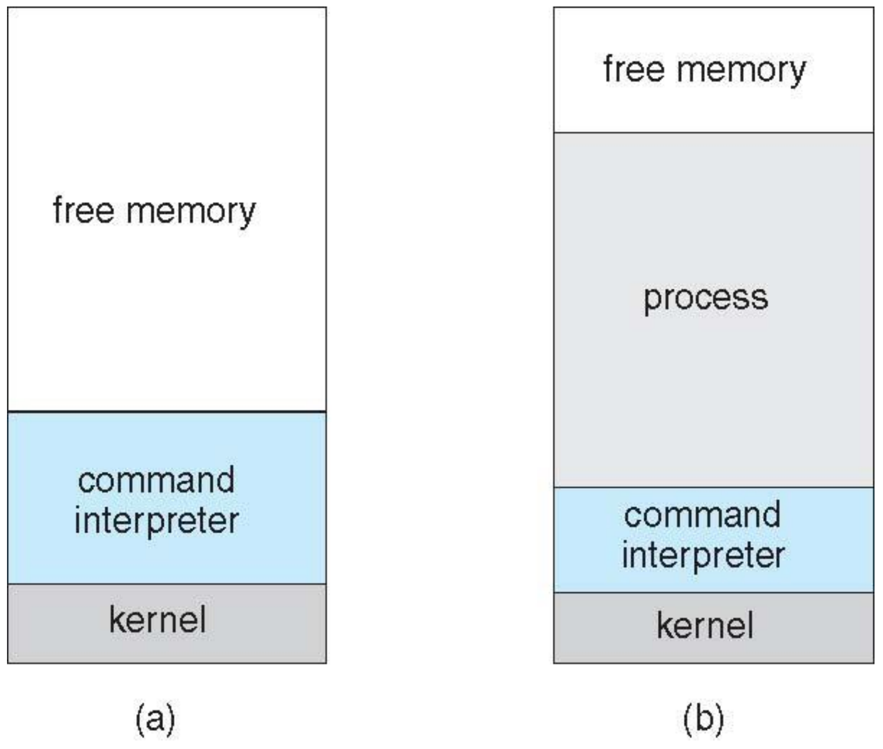
- C program invoking printf() library call, which calls write() system call



```
#include <stdio.h>
int main ( )
{
  •
  •
  •
  printf ("Greetings");
  •
  •
  •
  return 0;
}
```

Example: MS-DOS

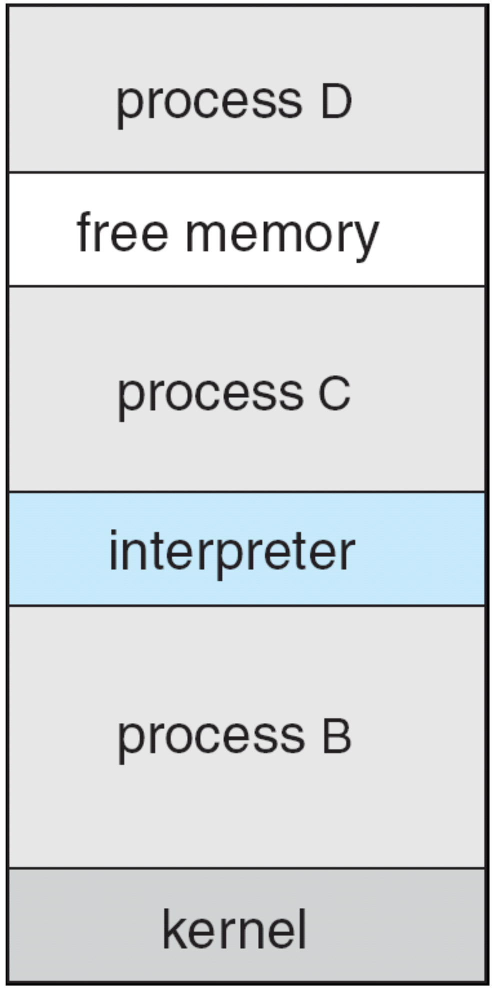
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup 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



Types of System Calls

- System calls can be grouped roughly into six major categories:
  - Process control
  - File manipulation
  - Device manipulation
  - Information maintenance
  - Communications
  - Protection

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

- 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

Operating System Structure

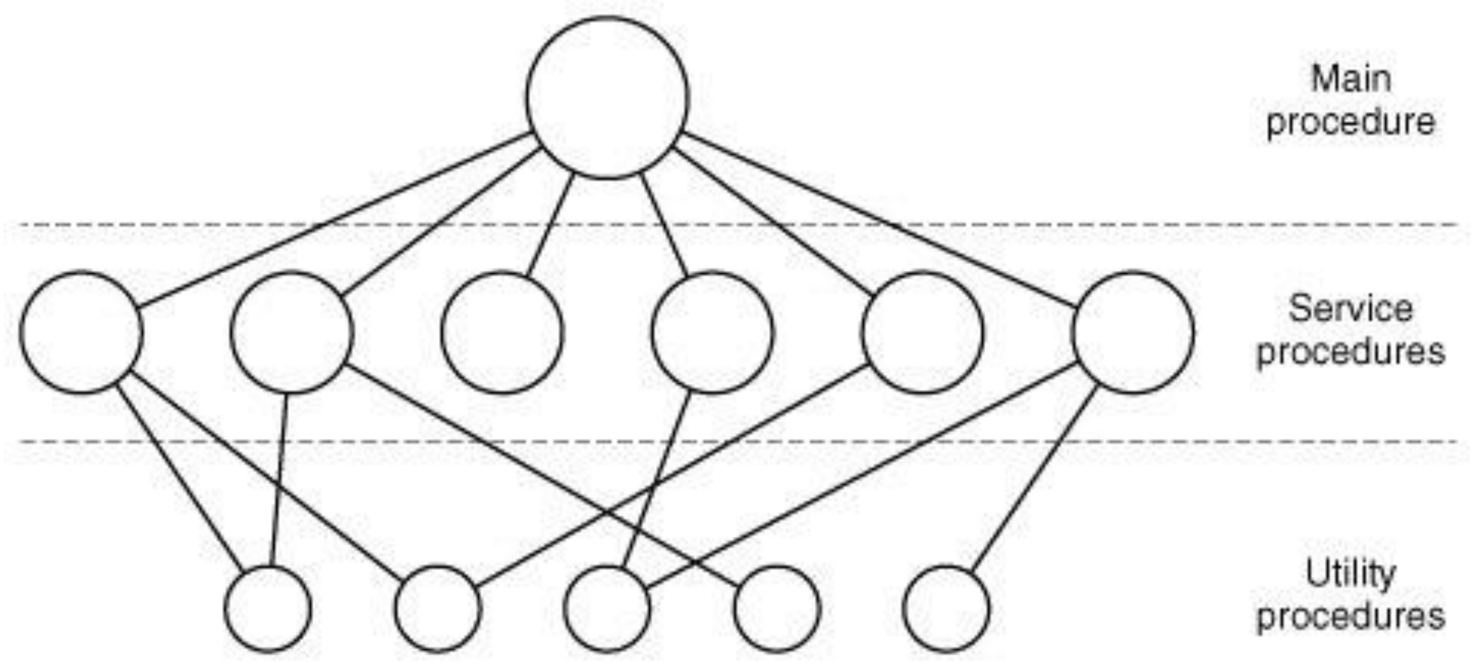
- General-purpose OS is very large program
- A common approach is to partition the task into small components, or modules, rather than have one monolithic system
- Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions
- Various ways to structure ones
  - Simple structure/Monolithic systems – MS-DOS
  - More complex -- UNIX
  - Layered – an abstrcation
  - Microkernel -Mach

Monolithic Systems

- In monolithic systems, there is no communication between layers
- The OS is written as a collection of procedures
- Uses
- Each procedure has a well-defined interface
- Each one is free to call any other one
- Services are provided by executing a special trap instruction called **kernel call** or **supervisor call**
- The basic structure of OS consists of
  1. A main program
  2. A set of service procedures
  3. A set of utility procedures

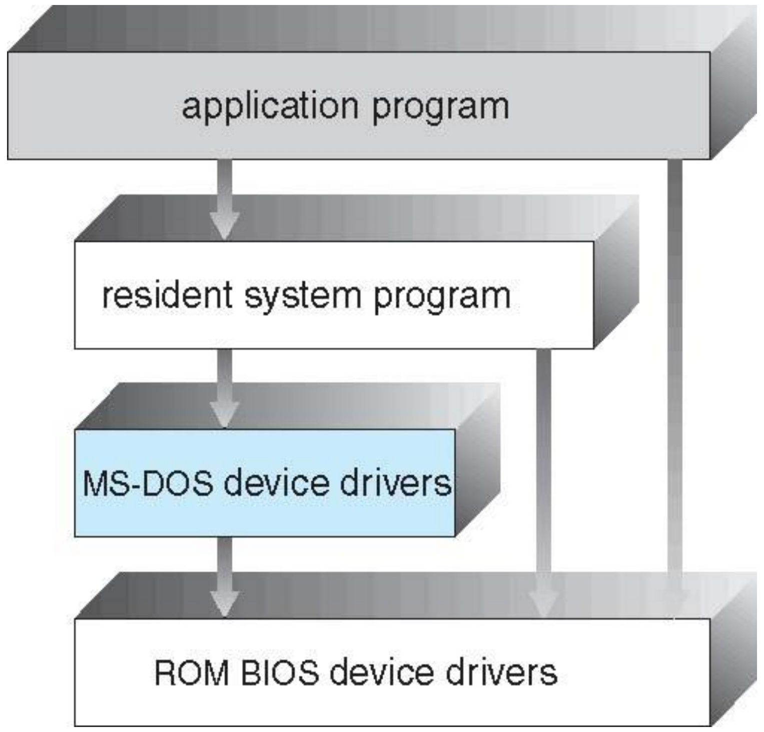
Monolithic Systems…

- A simple structuring model for a monolithic system



Simple Structure -- MS-DOS

- operating systems do not have well-defined structures.
- started as small, simple, and limited systems and then grew beyond their original scope.
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

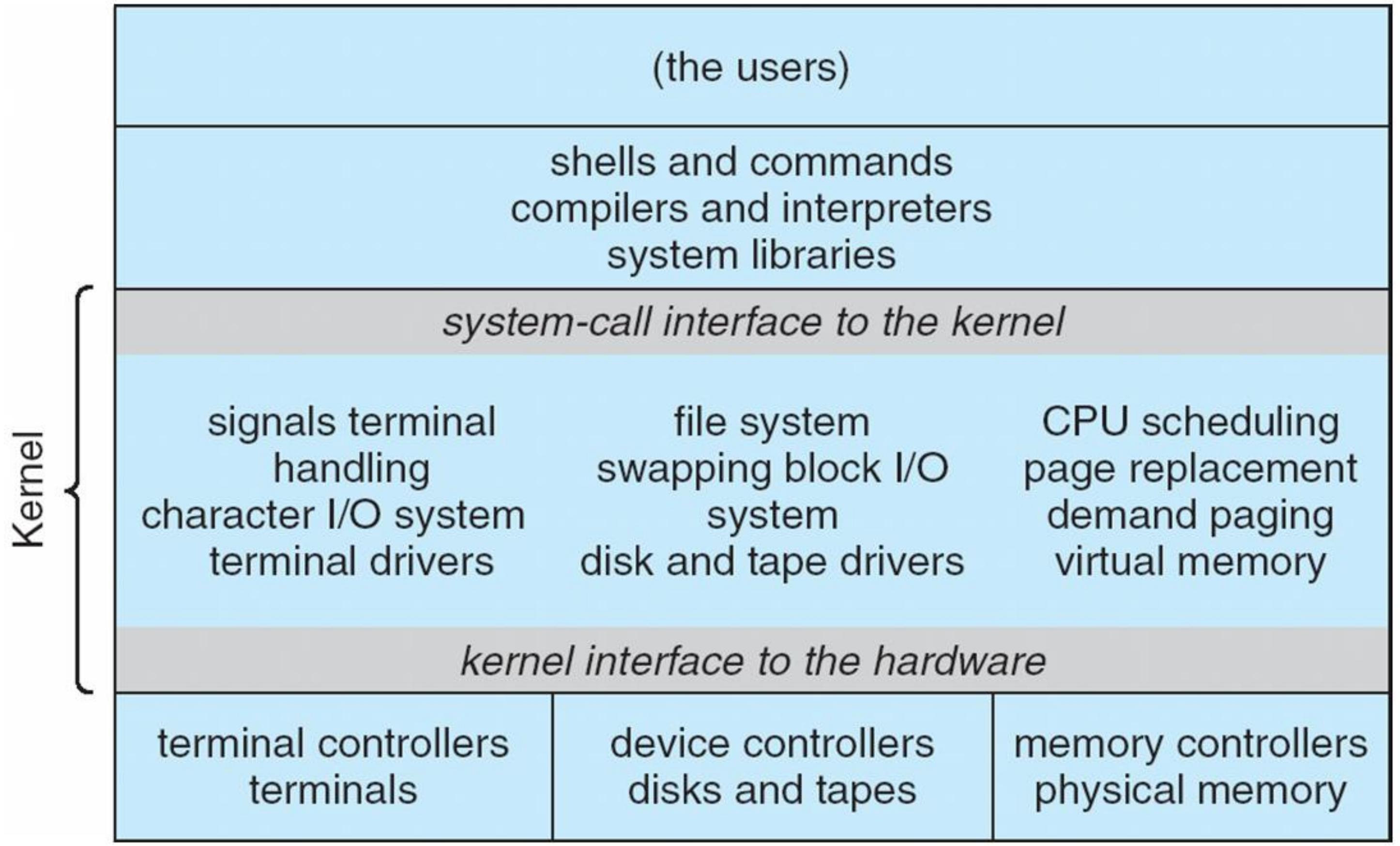


Non Simple Structure -- UNIX

- 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

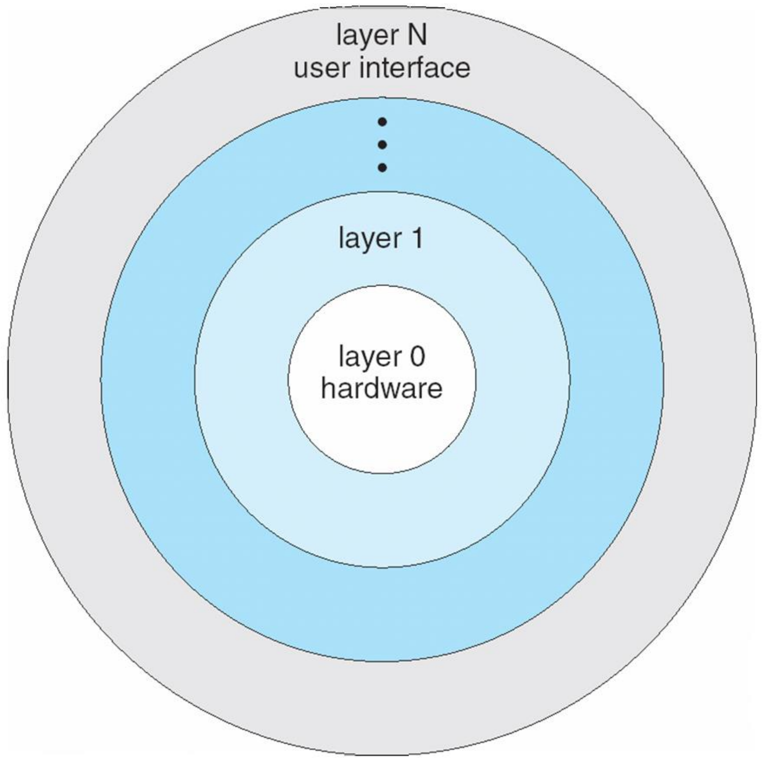
Traditional UNIX System Structure

- Beyond simple but not fully layered



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.
- Advantage of layered approach is **modularity**
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



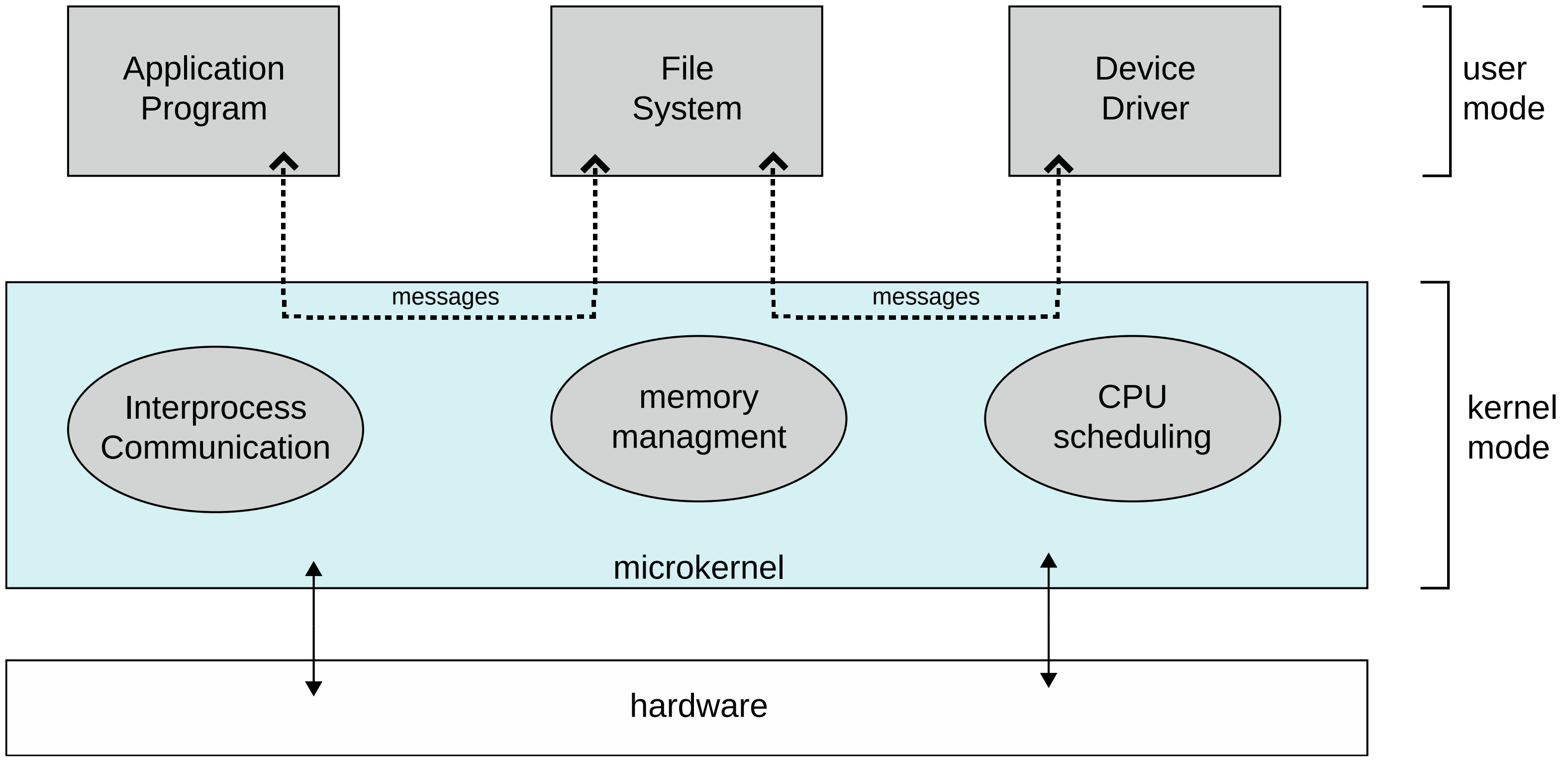
Layered Approach…

- Advantages
  - The main advantage of the layered approach is simplicity of construction and debugging.
  - Provides abstraction
- Limitations
  - Requires appropriate definition of various layers.
  - They tend to be less efficient than other types.

Microkernels

- Moves as much from the kernel into user space
- **Mach** 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

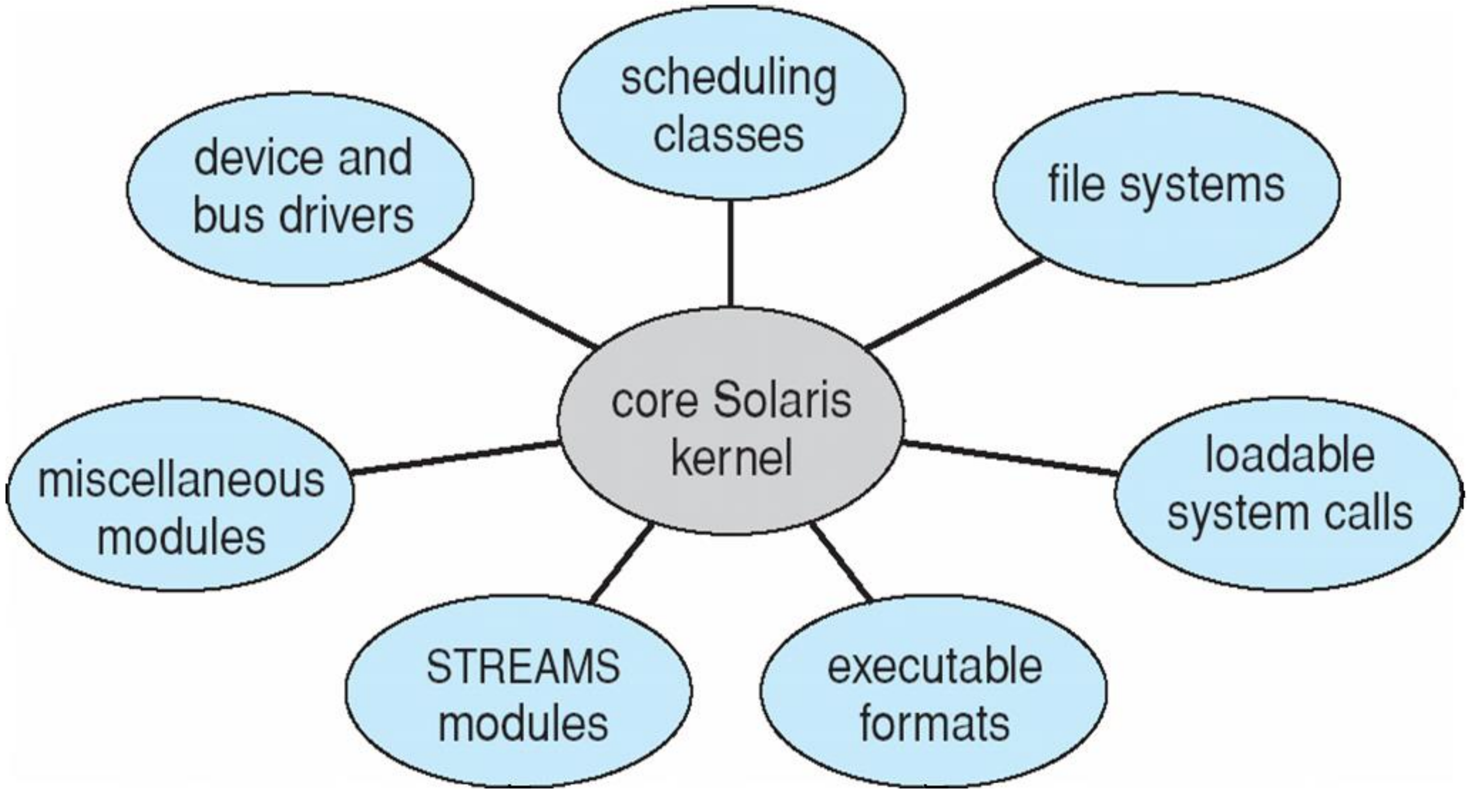
Microkernel System Structure …



Modules

- Many modern operating systems implement **loadable kernel modules**
  - 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

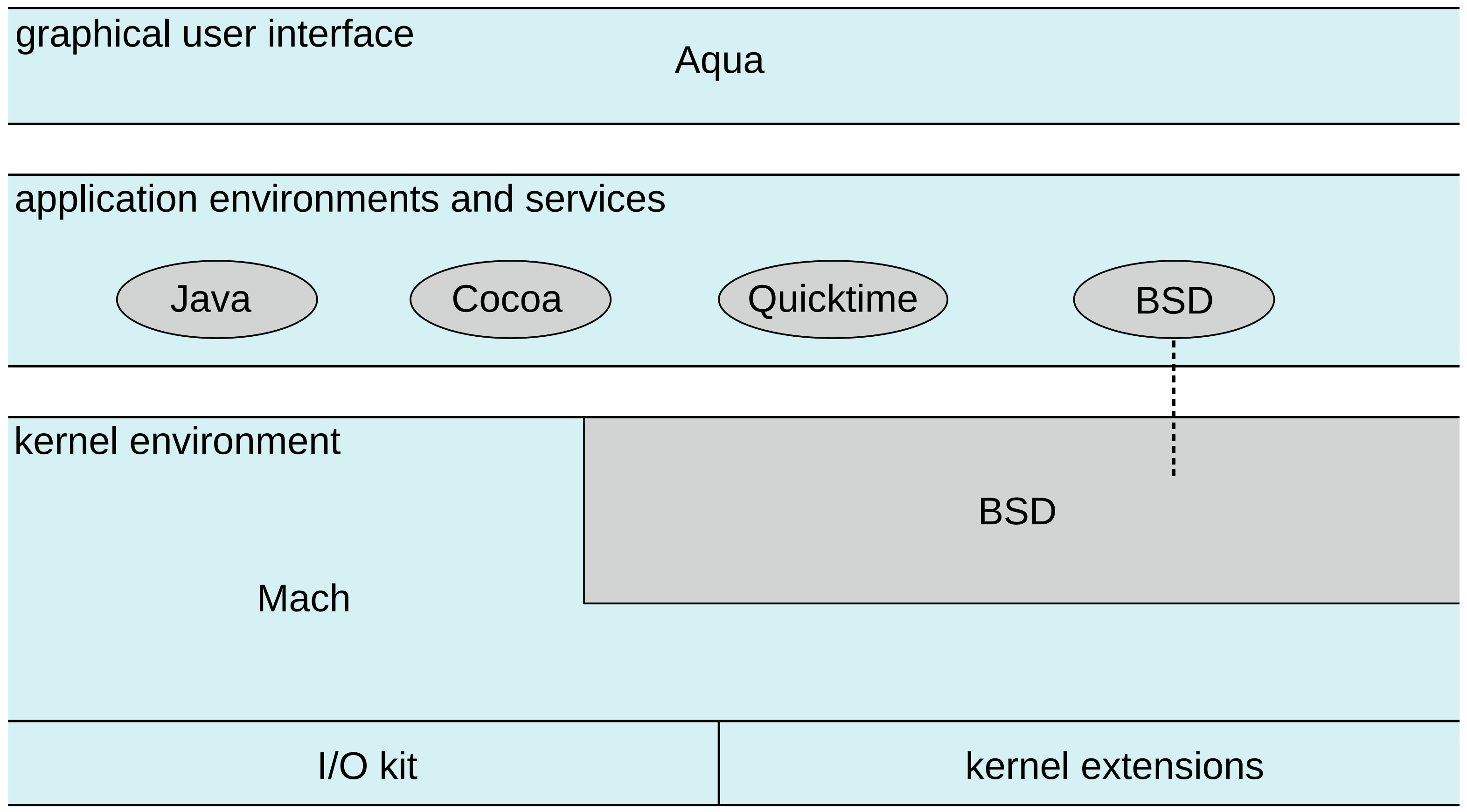
Solaris Modular Approach



Hybrid Systems

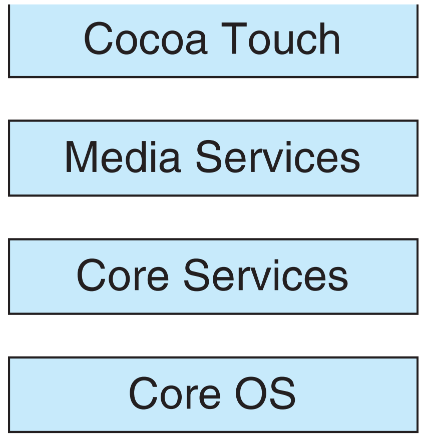
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, 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**)

Mac OS X Structure



iOS

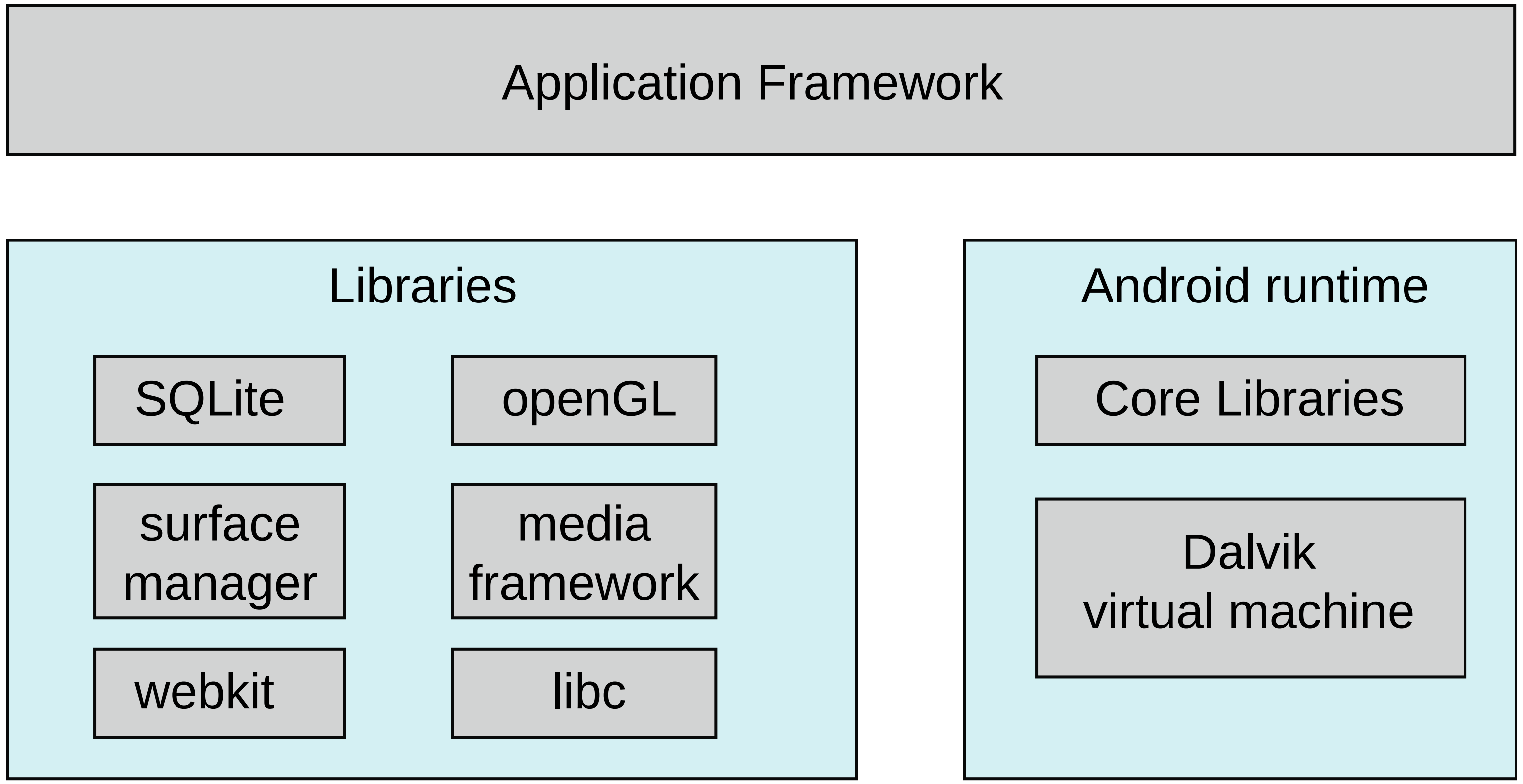
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Android

- 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 than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site→ **System Generation SYSGEN**
- **SYSGEN** program obtains information concerning the specific configuration of the hardware system
  - What CPU is to be used? What options are installed?
  - How will the boot disk be formatted? How many sections?
  - How much memory is available?
  - What devices are available?
  - What operating-system options are desired, or what parameter values are to be used?
- Used to build system-specific compiled kernel or system-tuned
- Can general more efficient code than one general kernel

Operating System Generation…

- This information can be used in several ways:

1. a system administrator can use it to modify a copy of the source code of the operating system. The operating system then is completely compiled to produce an output-object version of the operating system
2. the system description can lead to the creation of tables and the selection of modules from a precompiled library. These modules are linked together to form the generated operating system.
3. construct a system that is completely table driven. All the code is always part of the system, and selection occurs at execution time, rather than at compile or link time. System generation involves simply creating the appropriate tables to describe the system.

System Boot

- Operating system must be made available to hardware so hardware can start it
- The procedure of starting a computer by loading the kernel is known as **booting** the system.
  - Small piece of code – **bootstrap loader**, 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
- When power is initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- The bootstrap program can perform a variety of tasks
  - run diagnostics to determine the state of the machine
  - initialize all aspects of the system

System Boot…

- Storing the operating system in ROM is suitable for small operating systems
    - Problem: changing the bootstrap code requires changing the ROM hardware chips.
- The problem can be resolved by using erasable programmable read-only memory (EPROM)
- All forms of ROM are also known as **firmware**, since their characteristics fall somewhere between those of hardware and those of software.
- Problems with firmware
    - Executing code there is slower than executing code in RAM.
    - It is relatively expensive, so usually only small amounts are available.

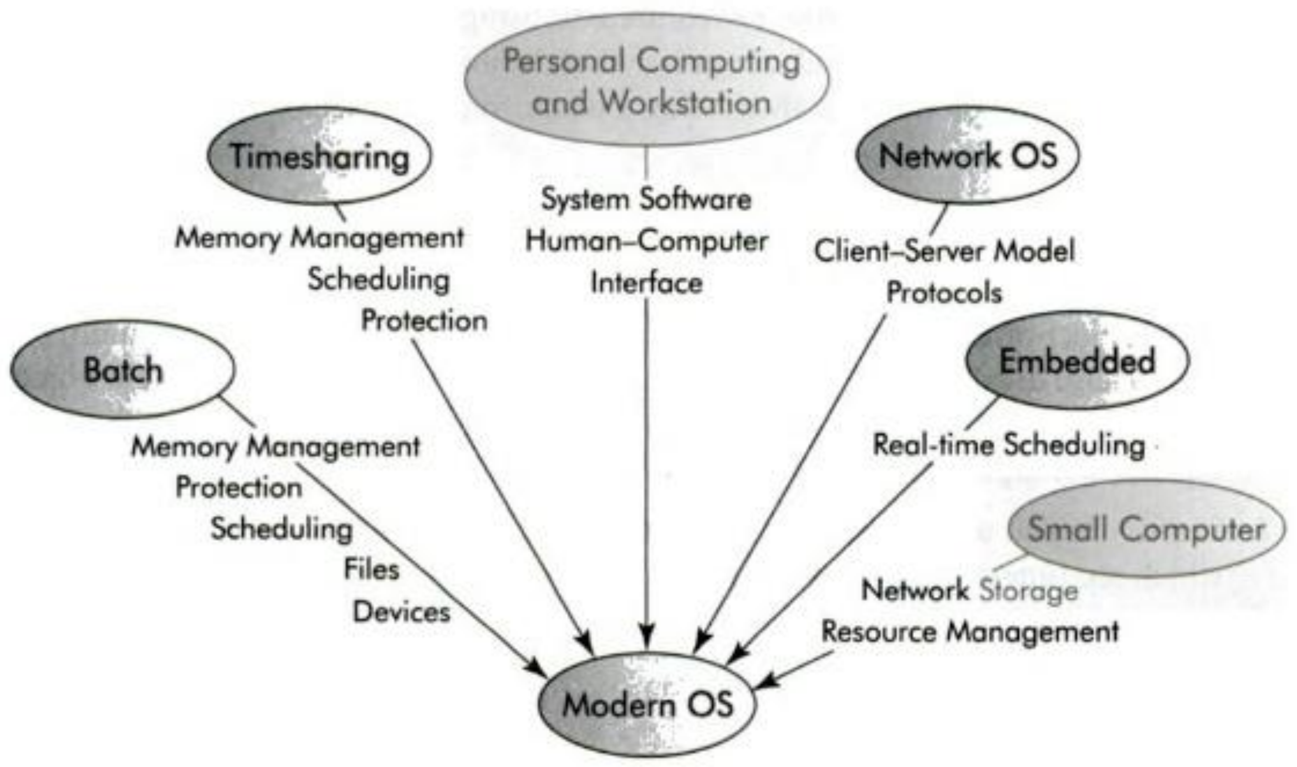
System Boot…

- For large operating systems or for systems that change frequently, the bootstrap loader is stored in firmware, and the operating system is on disk.
  - Boot Block(a single disk block) is loaded from disk to memory
  - Program in boot block load the entire operating system into memory and begin its execution.
- Kernel loads and system is then **running**
- A disk that has a boot partition is called a **boot disk** or **system disk**
- **GRUB** is an example of an open-source bootstrap program for Linux systems.

Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source**
- Counter to the **copy protection** and **Digital Rights Management (DRM)** movement
- Started by **Free Software Foundation (FSF)**, which has "copyleft" **GNU Public License (GPL)**
- 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 Evolution of Modern Operating Systems



Review Questions

- What are the three main purposes of an operating system?
- Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?
- Identify several advantages and several disadvantages of open-source operating systems.
- What is the purpose of interrupts? How does an interrupt differ from a trap?
- Describe the differences between symmetric and asymmetric multipro-cessing.

Review Questions

- What is the purpose of the command interpreter?
- What is the purpose of system calls?
- What is the main advantage of the layered approach to system design? What are the disadvantages of the layered approach
- List the services provided by an operating system
- Describe the methods for passing parameters to the operating system.
- What is the main advantage of the microkernel approach to system design?
- What are the advantages of using loadable kernel modules?
- Structure of various Operating Sytems: MS-DOS Windows, UNIX/LINUX, MAC OS, Android, iOS