

Operating-System Structures





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)**, **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





Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
 - **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





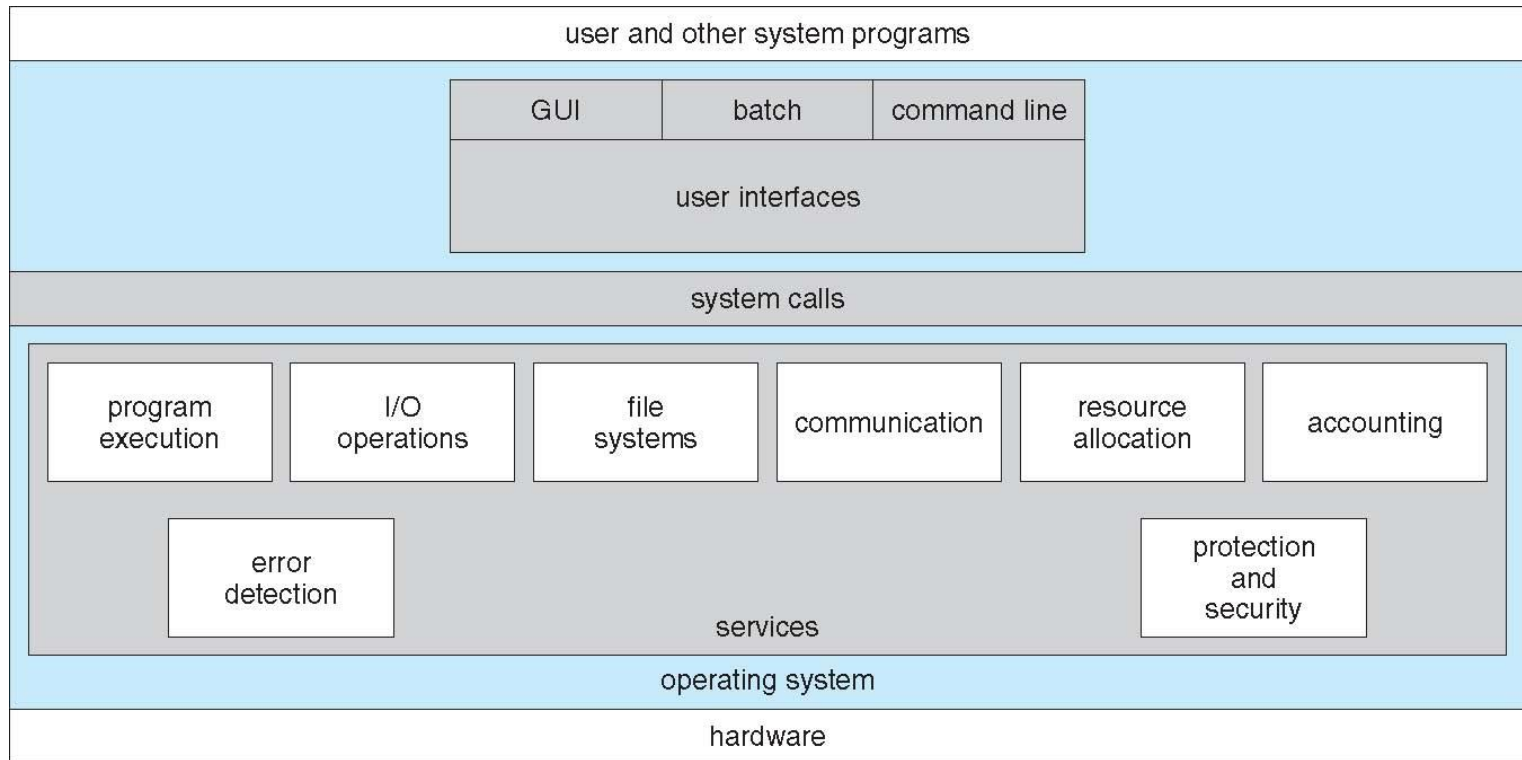
Operating System Services (Cont.)

- 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.
 - **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





A View of Operating System Services





User 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
- ❑ Sometimes commands built-in, sometimes just names of programs
 - ▶ If the latter, adding new features doesn't require shell modification





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)





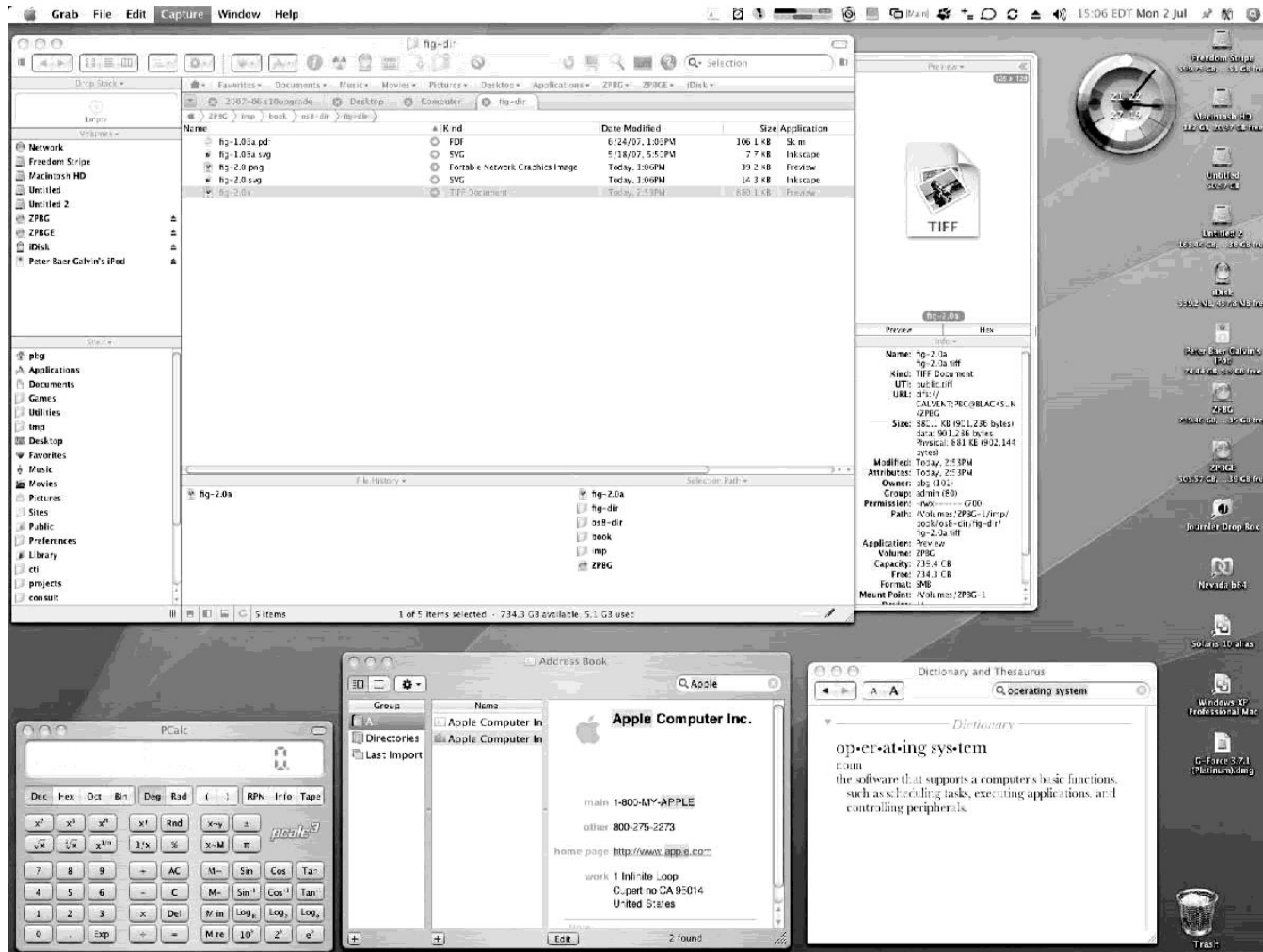
Touchscreen Interfaces

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





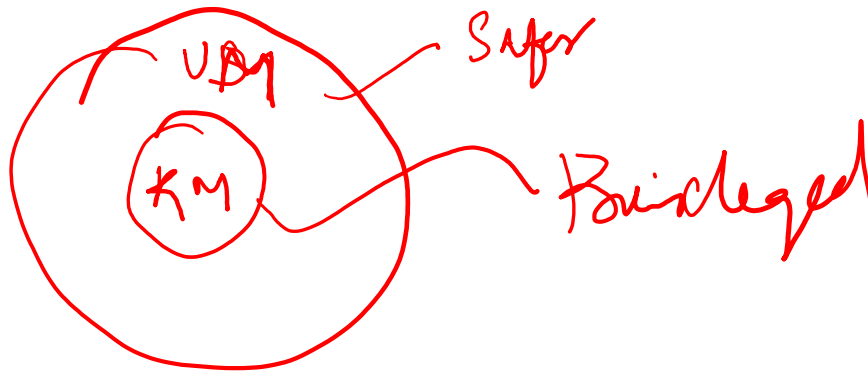
The Mac OS X GUI





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)



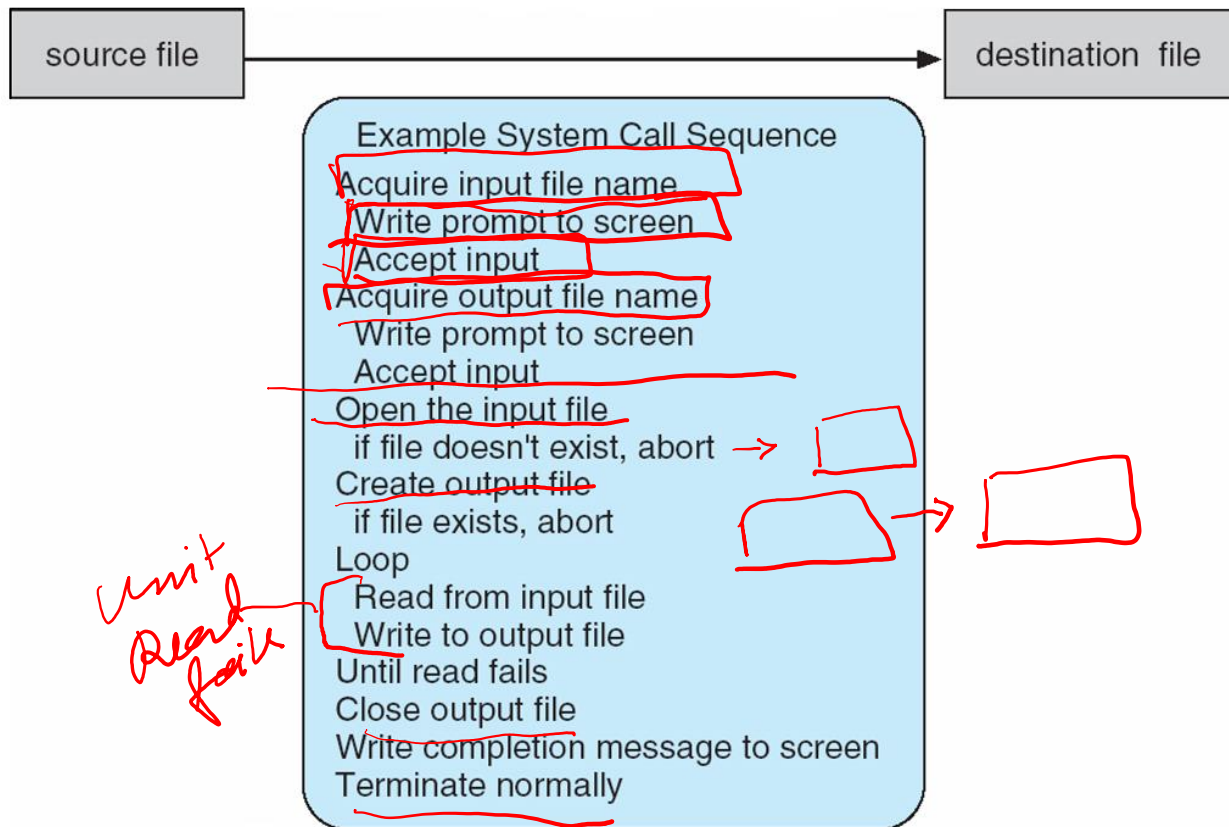
Context Switching





Example of System Calls

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





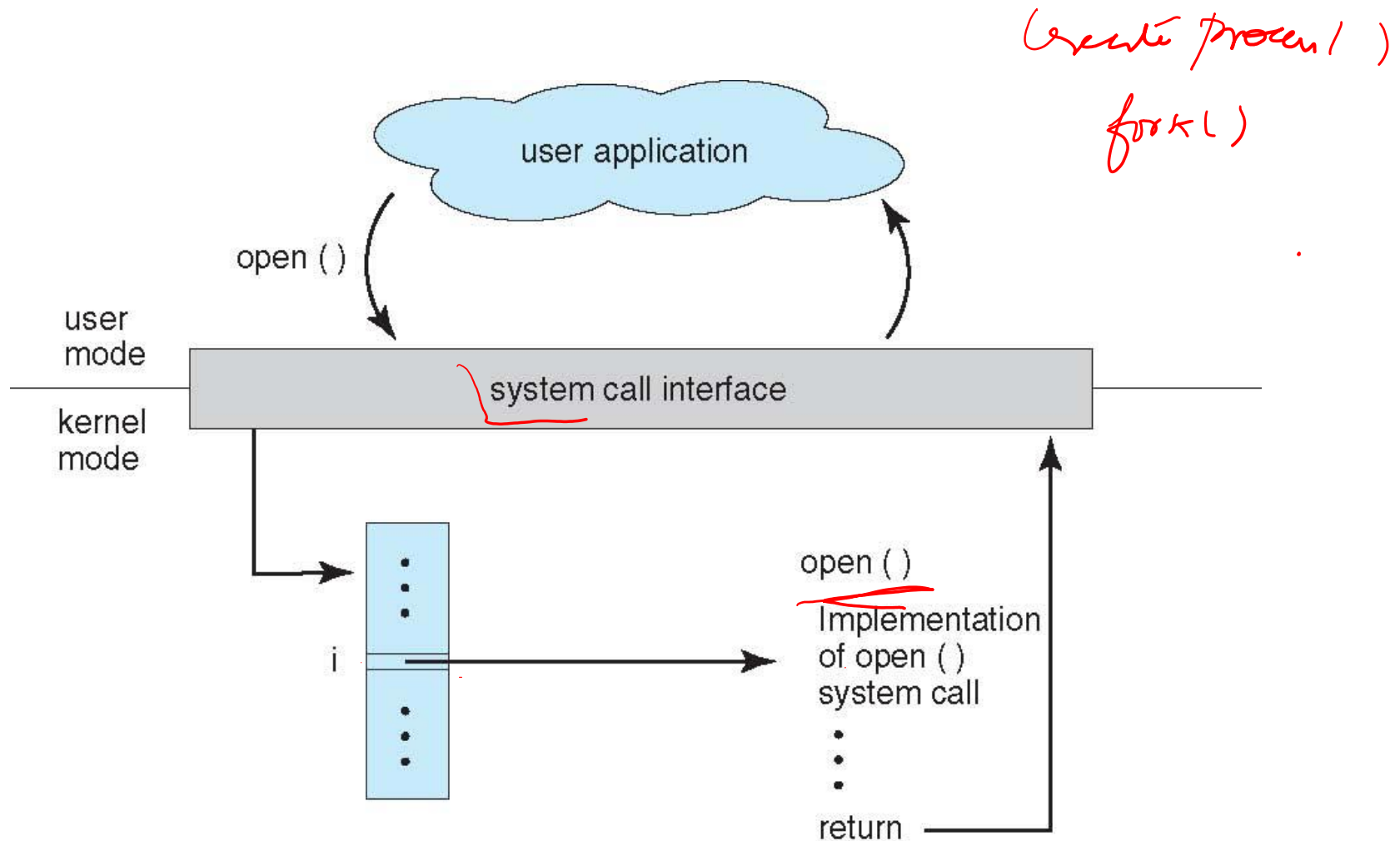
System Call Implementation

- 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





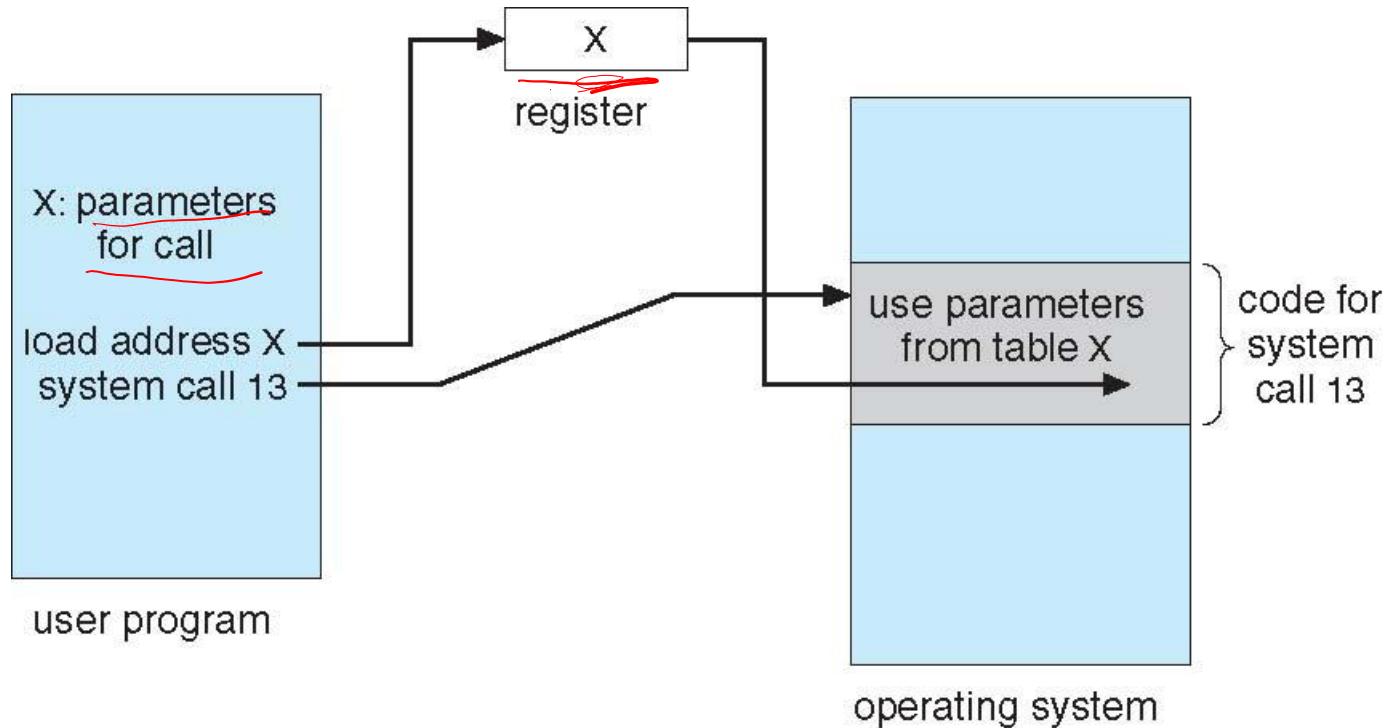
System Call Parameter Passing

- 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





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.)

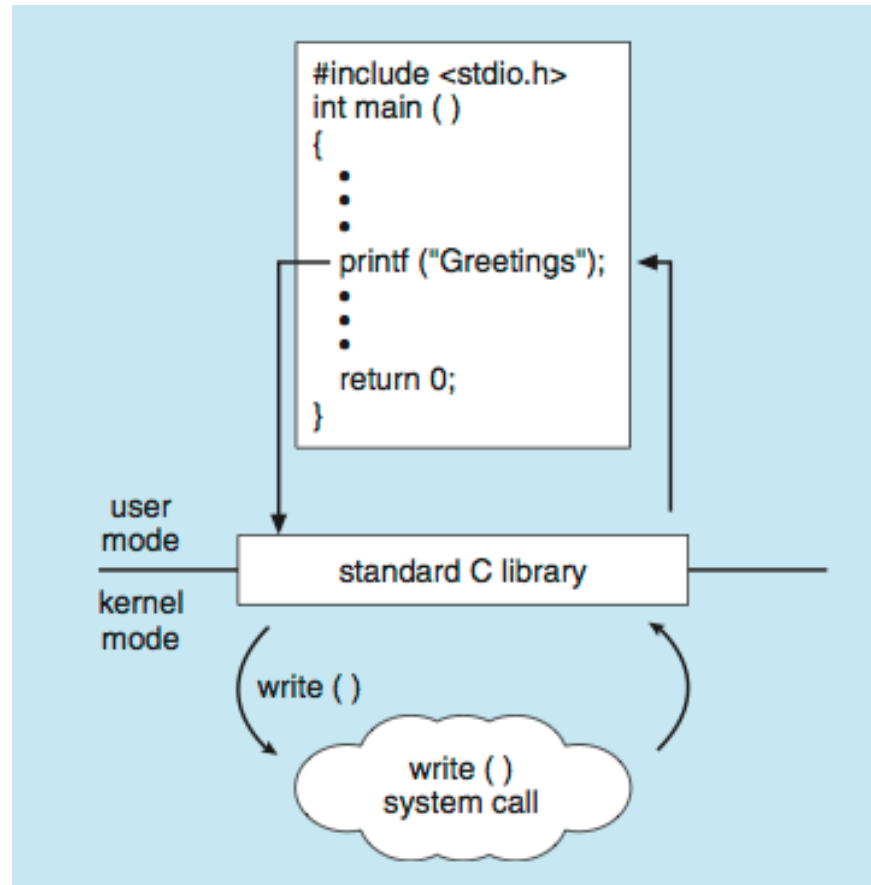
- Protection
 - Control access to resources
 - Get and set permissions
 - Allow and deny user access





Standard C Library Example

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





Operating System Structure

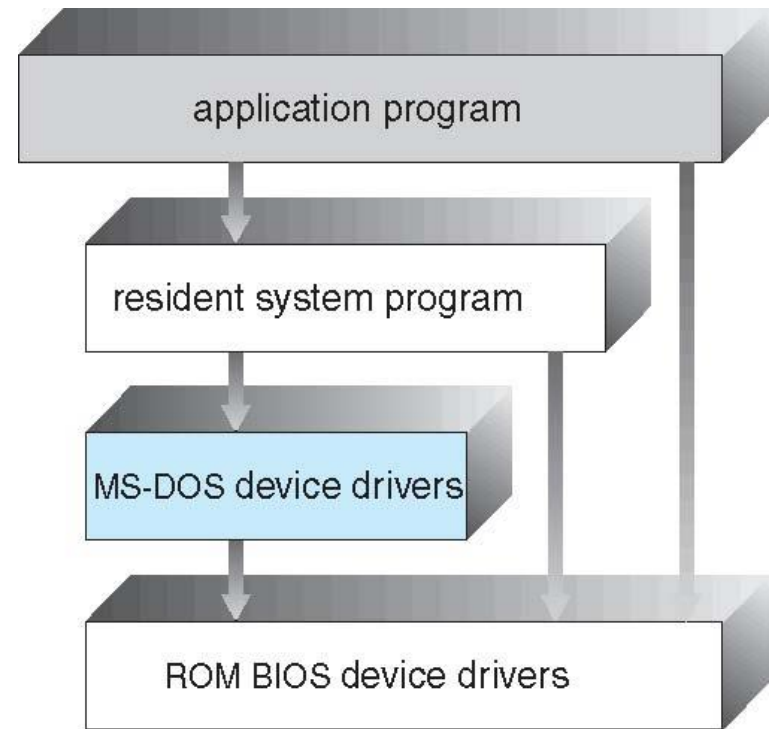
- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure – MS-DOS
 - More complex -- UNIX
 - Layered – an abstraction
 - Microkernel -Mach





Simple Structure -- MS-DOS

- 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

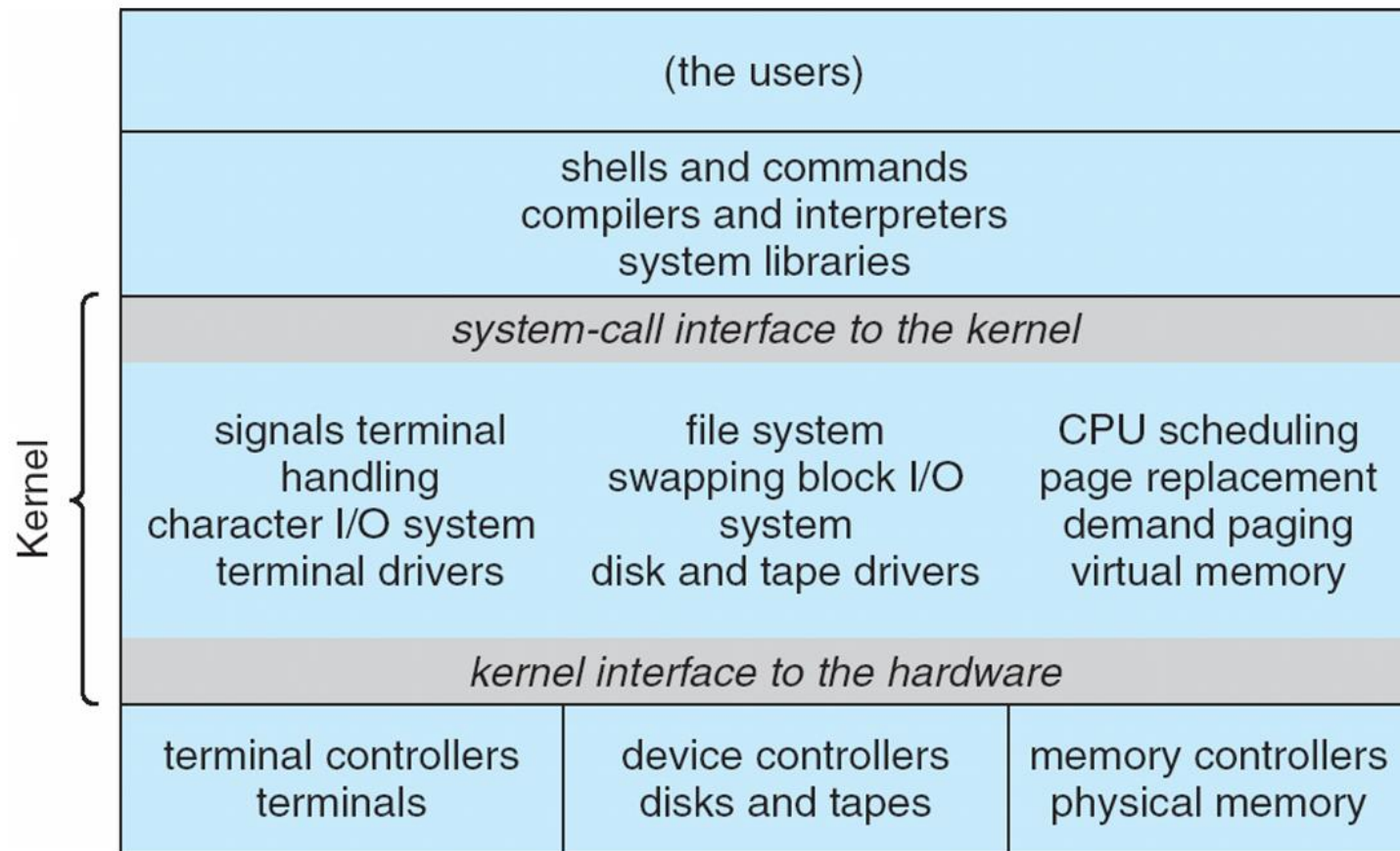
- Systems programs
- The kernel
 - ▶ Consists of everything below the system-call interface and above the physical hardware
 - ▶ Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level





Traditional UNIX System Structure

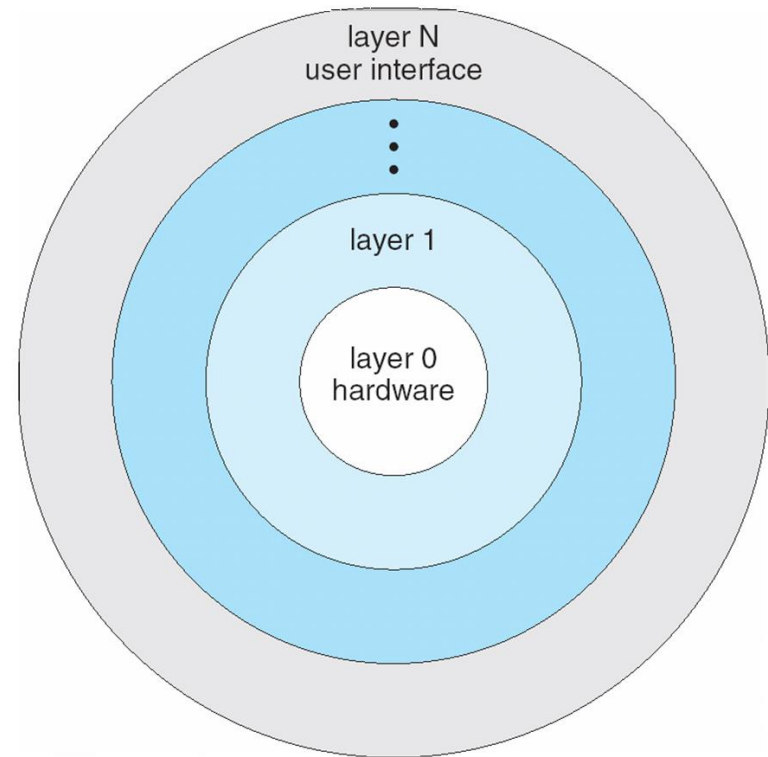
Beyond simple but not fully layered





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





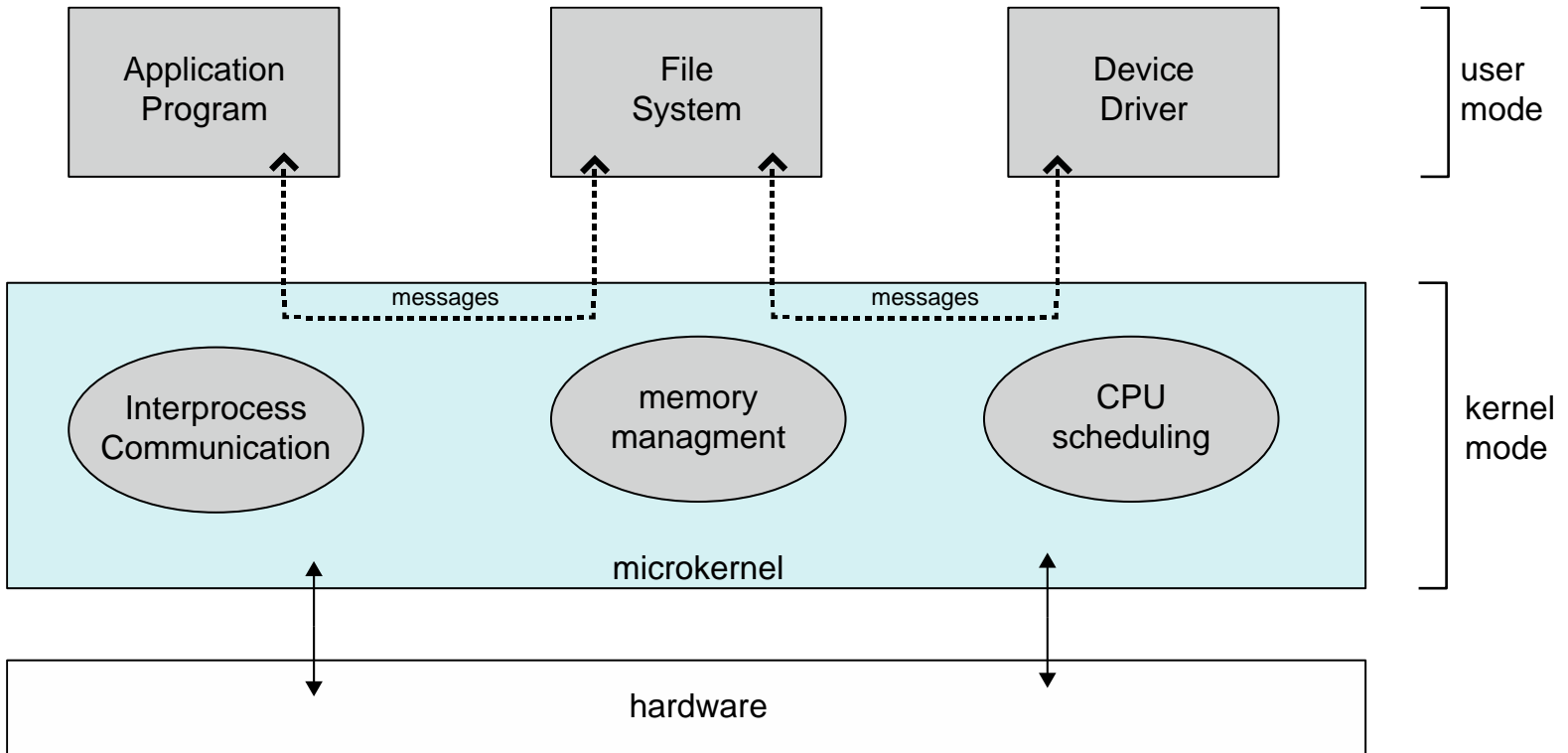
Microkernel System Structure

- ❑ 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





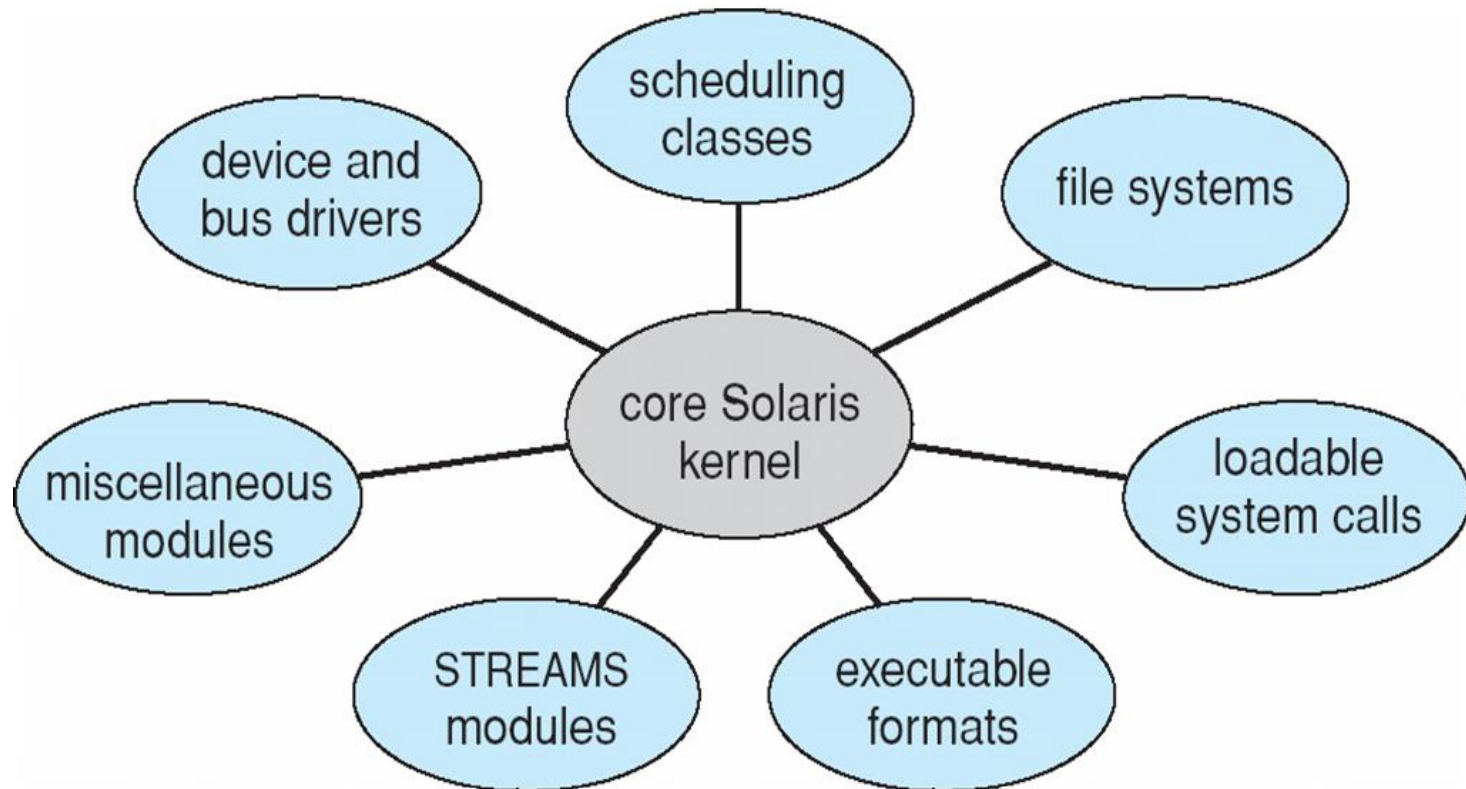
Modules

- 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





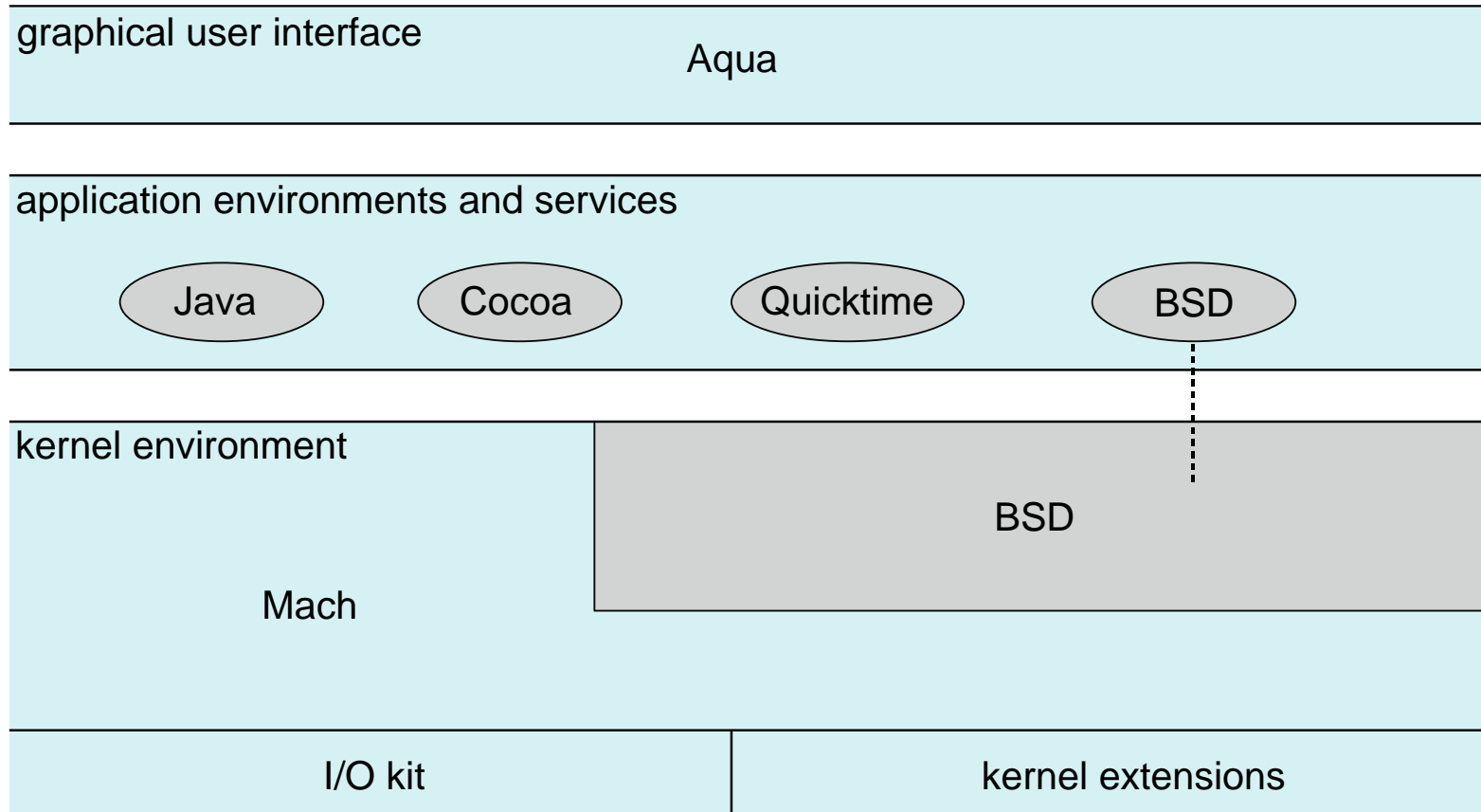
Hybrid Systems

- 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





Types of Operating Systems

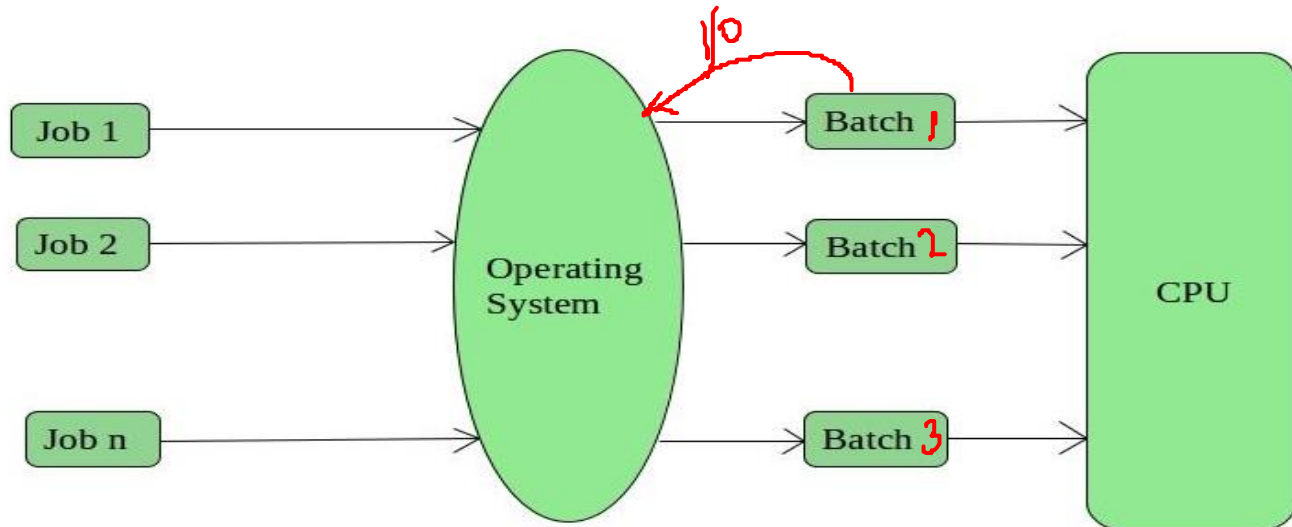
1. Batch Operating System
2. Multiprogramming Operating System
3. Time-Sharing OS
4. Multiprocessing OS
5. Distributed OS
6. Network OS
7. Real Time OS
8. Embedded OS





1. Batch Operating System

- ❑ The users of this type of operating system does not interact with the computer directly.
- ❑ Each user prepares his job on an off-line device like punch cards and submits it to the computer operator
- ❑ There is an operator which takes similar jobs having the same requirement and group them into batches.





1. Batch Operating System (contd...)

Advantages of Batch Operating System:

- Processors of the batch systems know how long the job would be when it is in queue
- Multiple users can share the batch systems
- The idle time for the batch system is very less
- It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails

Examples of Batch based Operating System:

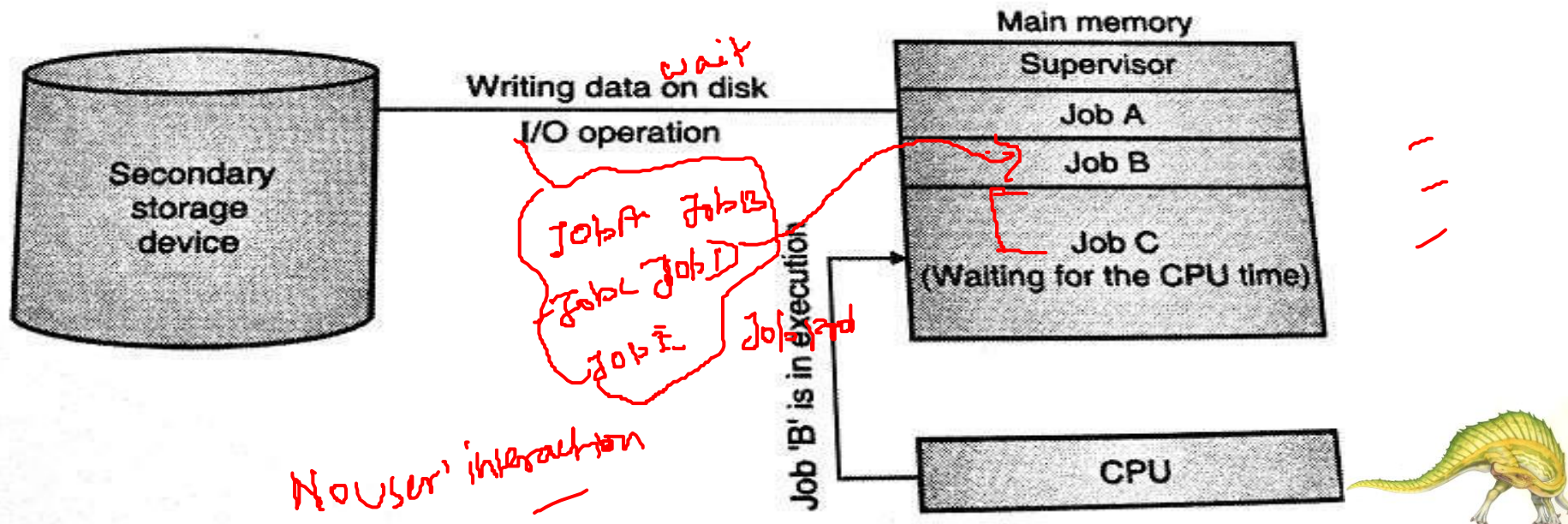
IBM's MVS





2. Multiprogramming Operating System:

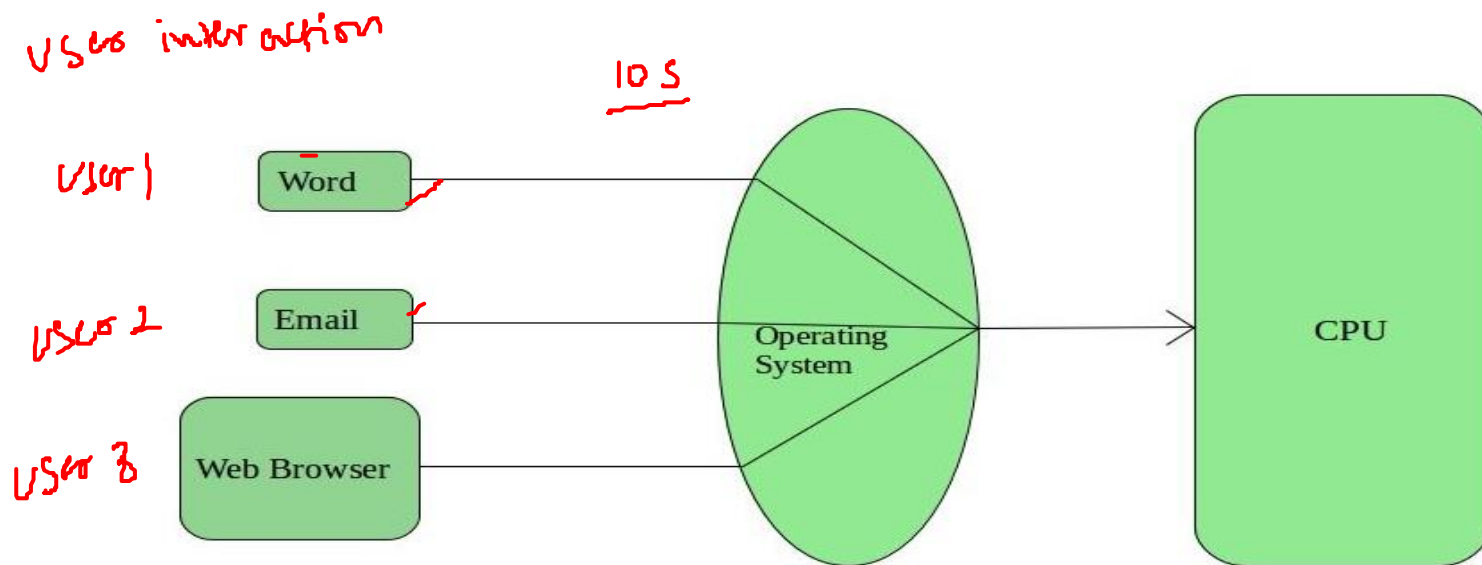
- This type of OS is used to execute more than one jobs simultaneously by a single processor.
- It increases CPU utilization by organizing jobs so that the CPU always has one job to execute.
- Multiprogramming operating systems use the mechanism of job scheduling and CPU scheduling.





3. Time-Sharing Operating Systems

- Each task is given some time to execute so that all the tasks work smoothly.
- These systems are also known as **Multi-tasking Systems**.
- The task can be from a single user or different users also.
- The time that each task gets to execute is called quantum.
- After this time interval is over OS switches over to the next task.





3. Time-Sharing Operating Systems cont..

□ Advantages of Time-Sharing OS:

- Each task gets an equal opportunity
- Fewer chances of duplication of software
- CPU idle time can be reduced

□ Disadvantages of Time-Sharing OS:

- Reliability problem
- One must have to take care of the security and integrity of user programs and data
- Data communication problem

□ Examples of Time-Sharing Oss

Multics, Unix, etc.





4. Multiprocessor operating systems

- Multiprocessor operating systems are also known as parallel OS or tightly coupled OS.
- Such operating systems have more than one processor in close communication that sharing the computer bus, the clock and sometimes memory and peripheral devices.
- It executes multiple jobs at the same time and makes the processing faster.
- It supports large physical address space and larger virtual address space.
- If one processor fails then other processor should retrieve the interrupted process state so execution of process can continue.
- Inter-processes communication mechanism is provided and implemented in hardware.

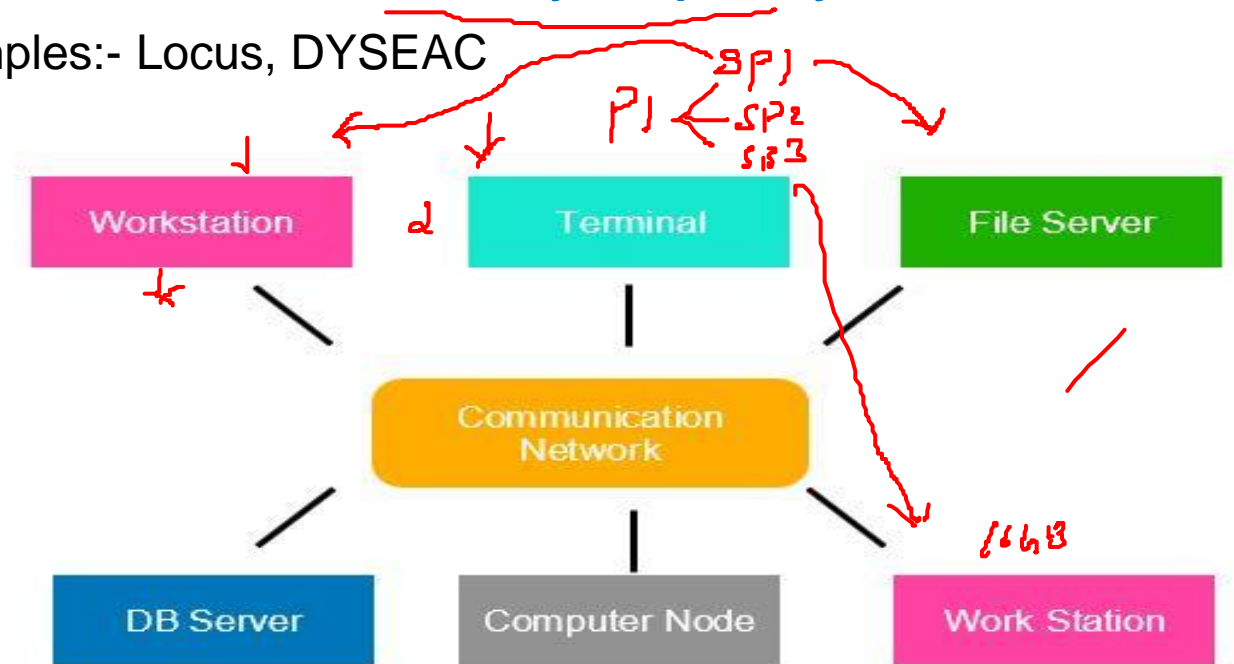




5. Distributed Operating System

- Various autonomous interconnected computers communicate with each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
- These are referred to as **loosely coupled systems**.
- Examples:- Locus, DYSEAC

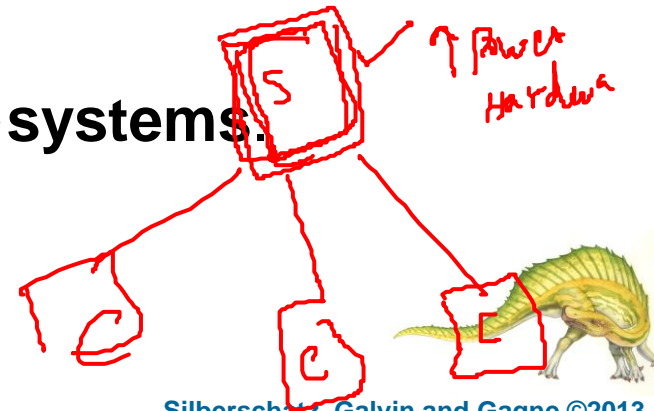
*Load Balance
Computation
Hardware
Data*





6. Network Operating System

- These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions.
- These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network.
- The “other” computers are called client computers, and each computer that connects to a network server must be running client software designed to request a specific service.
- popularly known as ~~peer-to-peer~~ ^{as} ~~coupled~~ ^{as} **tightly coupled systems**.





6. Network Operating System

Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated into the system
- Server access is possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on a central location for most operations
- Maintenance and updates are required regularly

Examples of Network Operating System are:

Microsoft Windows Server 2003/2008/2012, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc.

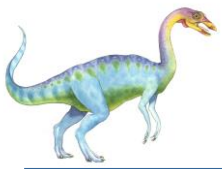




7. Embedded Operating System

- *Low power / Minimum functionality* An embedded operating system is one that is built into the circuitry of an electronic device.
- Embedded operating systems are now found in automobiles, bar-code scanners, cell phones, medical equipment, and personal digital assistants.
- The most popular embedded operating systems for consumer products, such as PDAs, include the following:
 - Windows XP Embedded
 - Windows CE .NET:- it supports wireless communications, multimedia and Web browsing. It also allows for the use of smaller versions of Microsoft Word, Excel, and Outlook.
 - Palm OS:- It is the standard operating system for Palm-brand PDAs as well as other proprietary handheld devices.
 - Symbian:- OS found in “smart” cell phones from Nokia and Sony Ericsson





8. Real-Time Operating System

- These types of OSs serve real-time systems.
- The time interval required to process and respond to inputs is very small. *[time bound]*
- This time interval is called **response time**. *Response Time → output*
- **Real-time systems** are used when there are time requirements that are very strict like
 - missile systems,
 - air traffic control systems,
 - robots, etc.



End of Chapter 2

