

# Operating System Structures

Lecturer: Kor Sokchea

**Royal University of Phnom Penh**

**Faculty of Engineering**

kor.sokchea@gmail.com



- **Textbook:** *Operating System Concepts*, 9<sup>th</sup> Edition, by Silberschatz, Galvin, and Gagne.
- Class
  - IT Engineering: Room **T002**
    - Monday Session: 13:00 - 14:30
    - Tuesday Session: 16:00 - 17:30
  - TEED: Room **T104**
    - Monday Session: 16:00 - 17:30
    - Tuesday Session: 13:00 - 14:30
- Exams
  - Final Exam: 60%
  - Homework: 10%
  - Assignment: 20%
  - Attendance: 10%

# Recapitulation

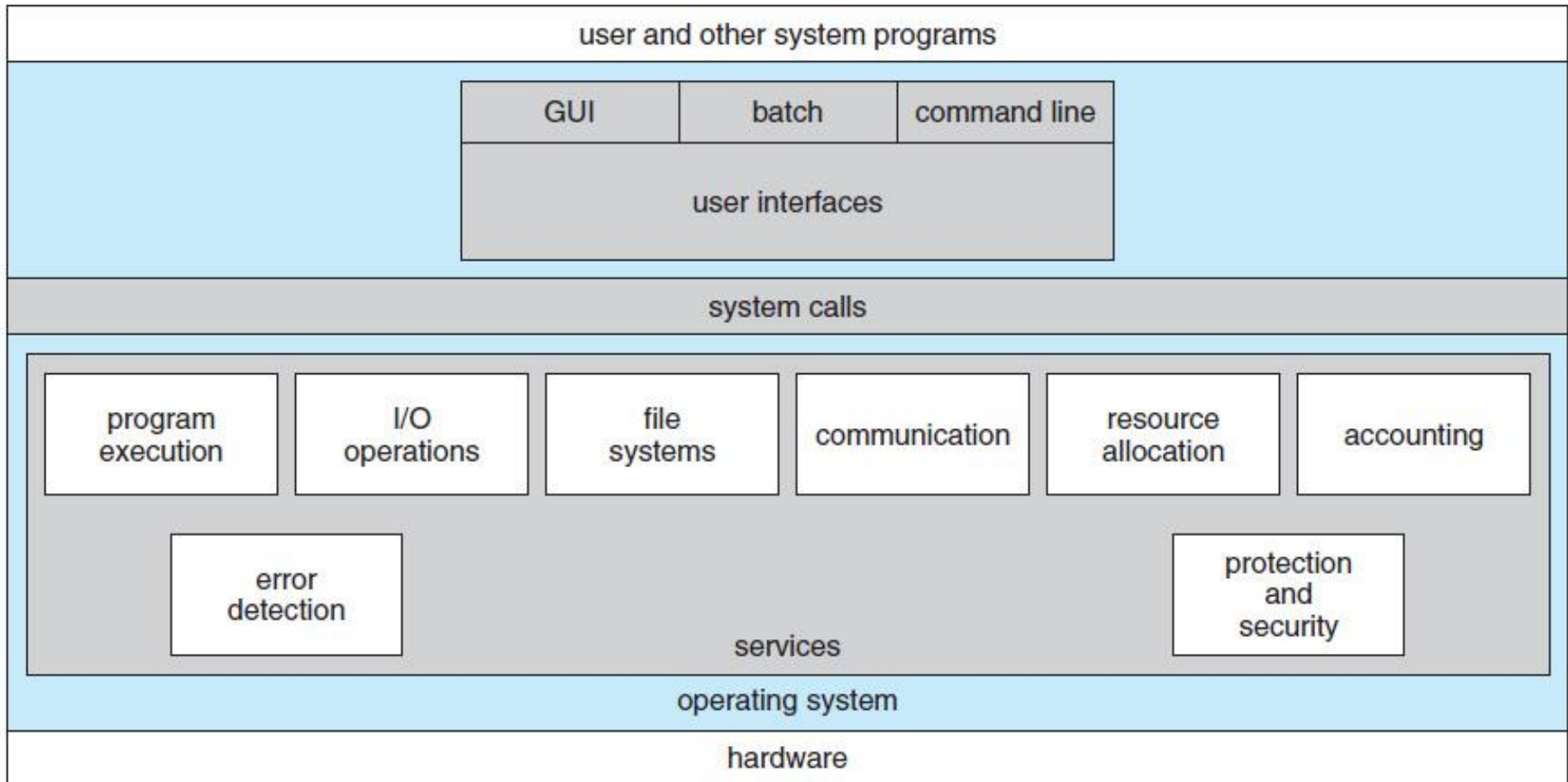
- What is an Operating System?
- Why study Operating Systems?
- Computer System Structure
- What Operating Systems do
- Computer-System Organization
- Computer-System Architecture
- **Operating System Structures**

# Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

# Operating-System Services

- An operating system offers an environment for the execution of programs,
- It provides services to programs and users of executed programs



**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 interface (CLI)**, **graphical 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

- ❑ One set of operating system services provides functions that are helpful to the user
  - **File-system manipulation:** The file system is of particular interest. Program need:
    - Read, write, create, delete, and search files and directories
    - List file information
    - Permissions management
  - **Communications:** Processes needs to exchange information, on the same computer or between computers over a network
    - Communications may be implemented via **shared memory** or through **message passing** (packets moved by the OS)

# Operating-System Services

- ❑ One set of operating system services provides functions that are helpful to the user
  - **Error detection:** OS needs to be constantly identifying and correcting error. Errors may occur in:
    - **CPU and memory hardware**
      - **Ex:** Memory error or power failure
    - **I/O devices**
      - **Ex:** Parity error on disk, a connection failure on a network, or lack of paper in the printer
    - **User Program**
      - **Ex:** Arithmetic overflow, access an illegal memory location, or a too-great use of CPU
    - For each type of error, the OS should take the appropriate action to ensure correct and consistent computing



# Operating-System Services

- ❑ Another set of OS functions exists for ensuring the efficient operation of the system itself through resource sharing
  - **Resource allocation:** When there are multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Various 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:** involve 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 - CLI

- ❑ **Command-line interface (CLI)** or **command interpreter** allows users to directly enter commands
  - Sometimes implemented in kernel, sometimes by systems program
  - The interpreters are known as **shells**
  - Various kinds of shells on UNIX and Linux systems
    - Bourne Shell
    - C shell
    - Bourne-Again shell
    - Korn shell
  - Primarily to get a command from user and executes it

# User and Operating-System Interface - CLI

❑ Commands can be implemented in two general ways

1. The command interpreter itself contains the code to execute command
  - **Ex:** a command to delete a file may cause the command interpreter to jump to a section of its, setting up the parameters and making the appropriate system call
  - The number of commands that can be given determines the size of the command interpreter
2. Alternative approach used by UNIX implements most commands through system programs
  - The command interpreter does not understand the command in any way
  - It uses the command to identify a file to be loaded into memory and executed

# Shell Example

```
Terminal
File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.0    0.2    0.0    0.2  0.0  0.0    0.4  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
          extended device statistics
device   r/s    w/s    kr/s    kw/s wait actv  svc_t  %w  %b
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.6    0.0   38.4    0.0  0.0  0.0    8.2  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# w
 4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User      tty          login@ idle   JCPU   PCPU   what
root      console      15Jun0718days    1          /usr/bin/ssh-agent -- /usr/bi
n/d
root      pts/3        15Jun07          18    4   w
root      pts/4        15Jun0718days          w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
- (/var/tmp/system-contents/scripts)#
```

# Shell Example

```
Command Prompt

Connection-specific DNS Suffix  . :
IPv6 Address. . . . . : 2001:0:5ef5:79fd:c9f:9f88:35c5:9c08
Link-local IPv6 Address . . . . . : fe80::c9f:9f88:35c5:9c08%5
Default Gateway . . . . . : ::

Tunnel adapter isatap.{0CF62963-BA5E-4D0C-BA24-6A3ACA75E321}:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

Tunnel adapter isatap.{2208D382-CECE-4A1F-9335-087FDE291B45}:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

C:\Users\dell>ping google.com

Pinging google.com [43.252.16.168] with 32 bytes of data:
Reply from 43.252.16.168: bytes=32 time=188ms TTL=58
Reply from 43.252.16.168: bytes=32 time=34ms TTL=58
Reply from 43.252.16.168: bytes=32 time=35ms TTL=58
Reply from 43.252.16.168: bytes=32 time=35ms TTL=58

Ping statistics for 43.252.16.168:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 34ms, Maximum = 188ms, Average = 73ms

C:\Users\dell>rm hello.c_
```

# User and Operating System Interface - GUI

- ❑ User friendly graphical user interface (GUI)
  - Mouse-based window and menu system characterized by a desktop metaphor
  - **Icons** represent programs, files, directories and system functions
  - Depending on the mouse pointer's location, clicking a button on the mouse can invoke various actions
    - Select a file or directory (**Folder**)
    - Select program
    - Pull down a menu
    - Provide information and option
  - Invented at Xerox PARC and popularize by Apple

# User and Operating System Interface - GUI

- ❑ Many systems currently contain both CLI and GUI interfaces
  - Microsoft windows is GUI with command line
  - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces
    - KDE
    - GNOME
    - CDE

# Touchscreen Interface

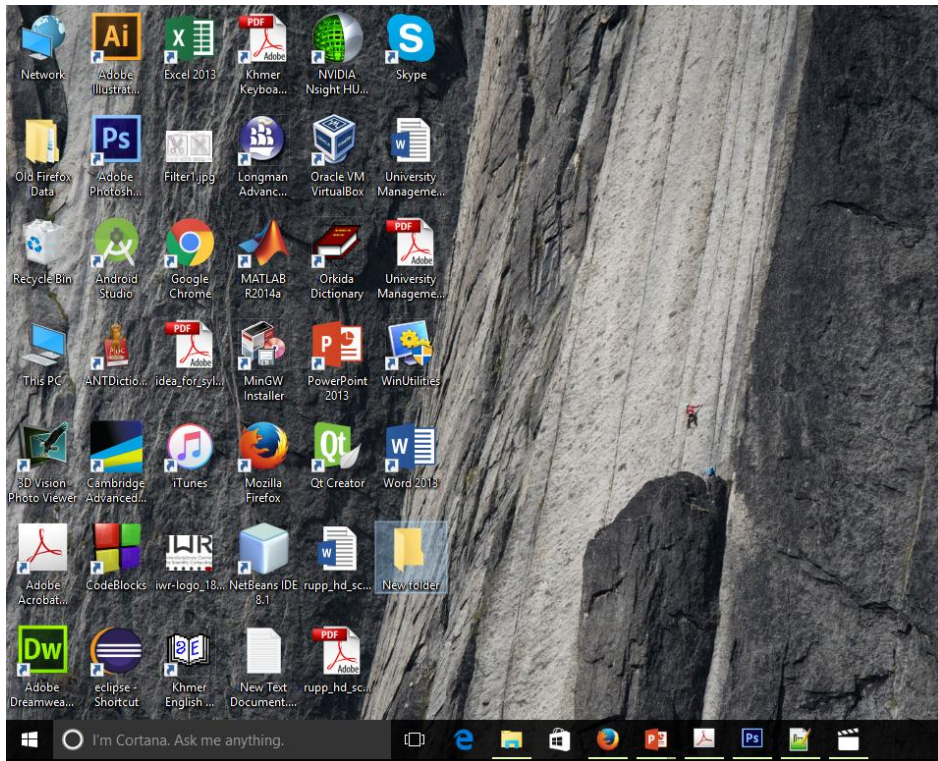
- ❑ Touchscreen interface is a favorite choice for mobile systems
- ❑ Users interact by making gestures on the touchscreen to perform various action





# Choice of Interface

- ❑ System administrators or Unix-like system frequently use the command-line interface
- ❑ Window user prefers the Window GUI environment and almost never use the MS-DOS shell interface



```
Command Prompt

Connection-specific DNS Suffix  . : 
IPv6 Address. . . . . : 2001:0:Sef5:79fd:c9f:9f88:35c5:9c08
Link-local IPv6 Address . . . . : fe80::c9f:9f88:35c5:9c08%5
Default Gateway . . . . . : 

Tunnel adapter {0CF62963-BA5E-4D0C-BA24-6A3ACA75E321}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Tunnel adapter {2208D382-CECE-4A1F-9335-087FDE291B45}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

C:\Users\dell>ping google.com

Pinging google.com [43.252.16.168] with 32 bytes of data:
Reply from 43.252.16.168: bytes=32 time=188ms TTL=58
Reply from 43.252.16.168: bytes=32 time=34ms TTL=58
Reply from 43.252.16.168: bytes=32 time=35ms TTL=58
Reply from 43.252.16.168: bytes=32 time=35ms TTL=58

Ping statistics for 43.252.16.168:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 34ms, Maximum = 188ms, Average = 73ms

C:\Users\dell>rm hello.c
```

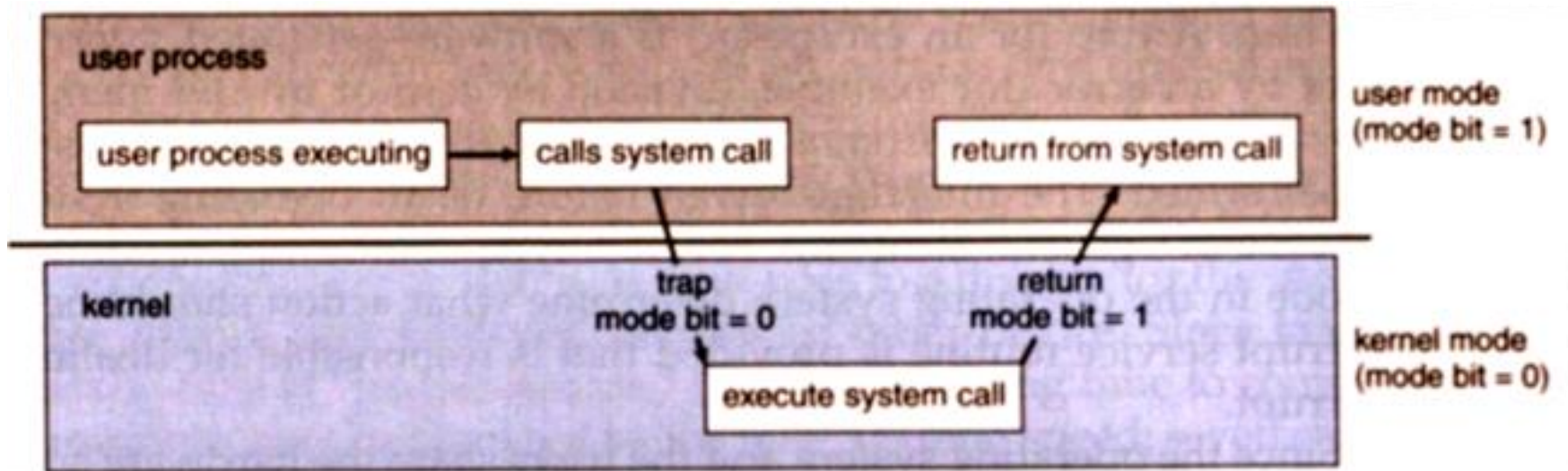
# System Calls

- ❑ System calls provide an interface to the operating system services
- ❑ Application developers often do not have direct access to the system
- ❑ Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- ❑ The functions that are included in the API invoke the actual system calls
- ❑ Three of the most common APIs available to application programmers 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 Machine (JVM)**

# System Calls

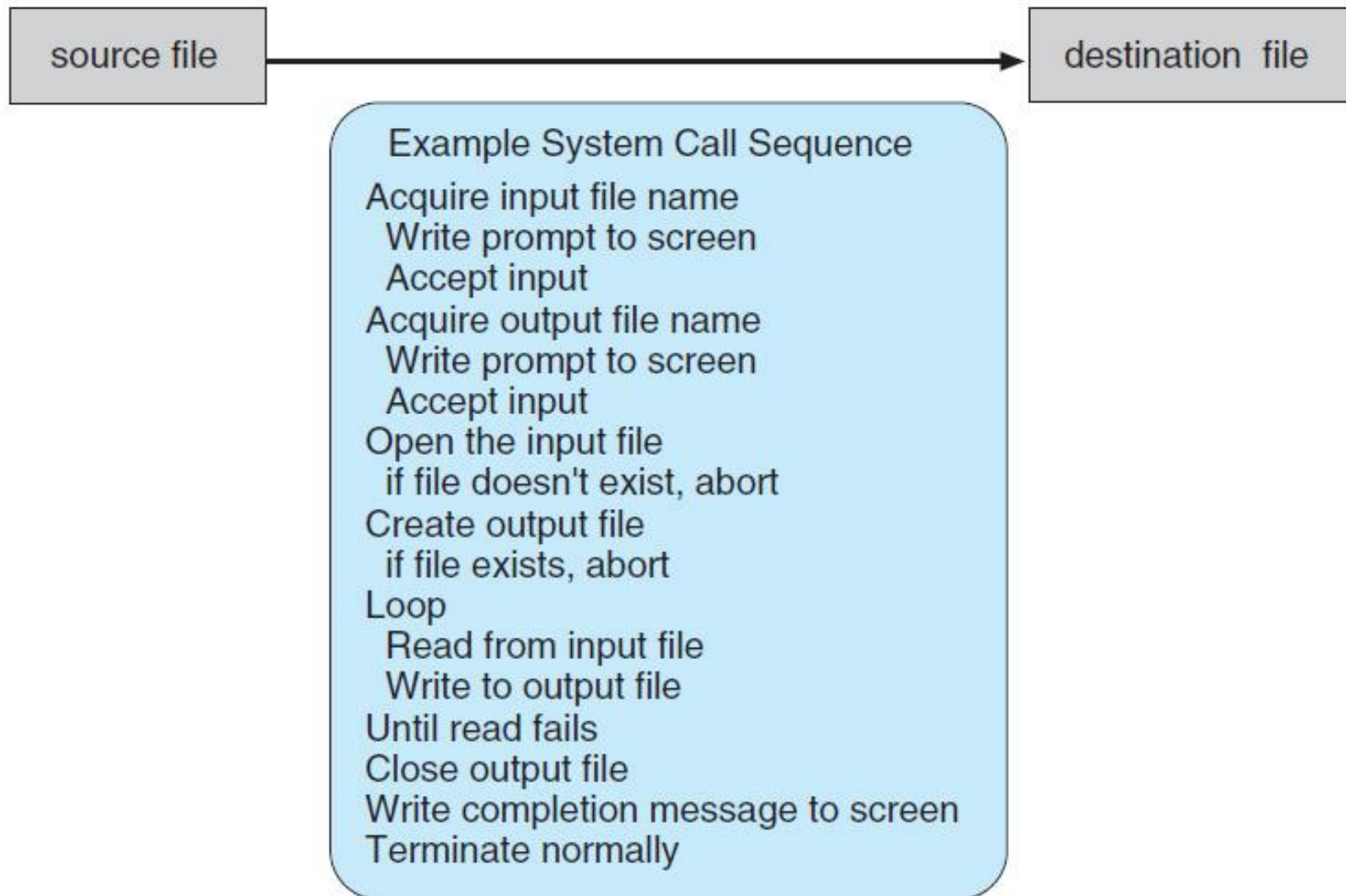
## ❑ Advantage of using API

- ❑ Portability: as long as system supports an API, any program using that API can compile and run
- ❑ Ease of Use: using the API can be significantly easier than using the actual system call



# Example of how system calls are used

□ Note: the system-call names used throughout below text are generic





# Example of Standard API

## EXAMPLE OF STANDARD API

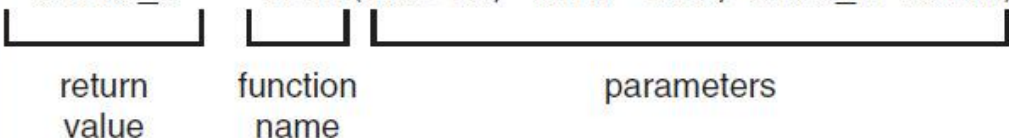
As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
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:

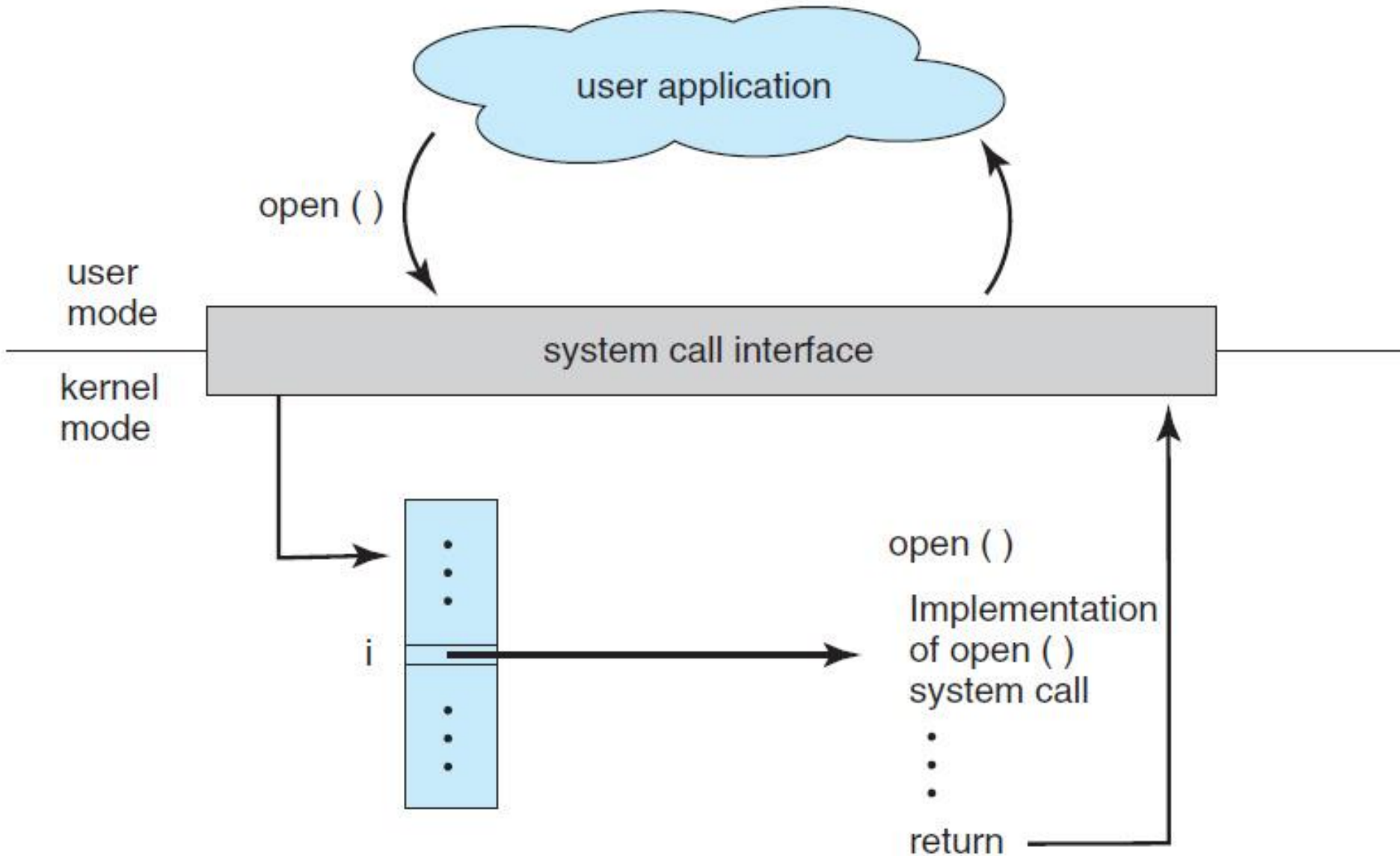
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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

- ❑ Typically, a number is associated with each system call
  - **System call interface** maintains a table indexed according to these numbers
- ❑ System call interface tasks:
  - Invokes the intended system call in the operating system kernel
  - Returns the status of the system call and any return values
- ❑ The caller needs to 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 Interface – OS Relationship

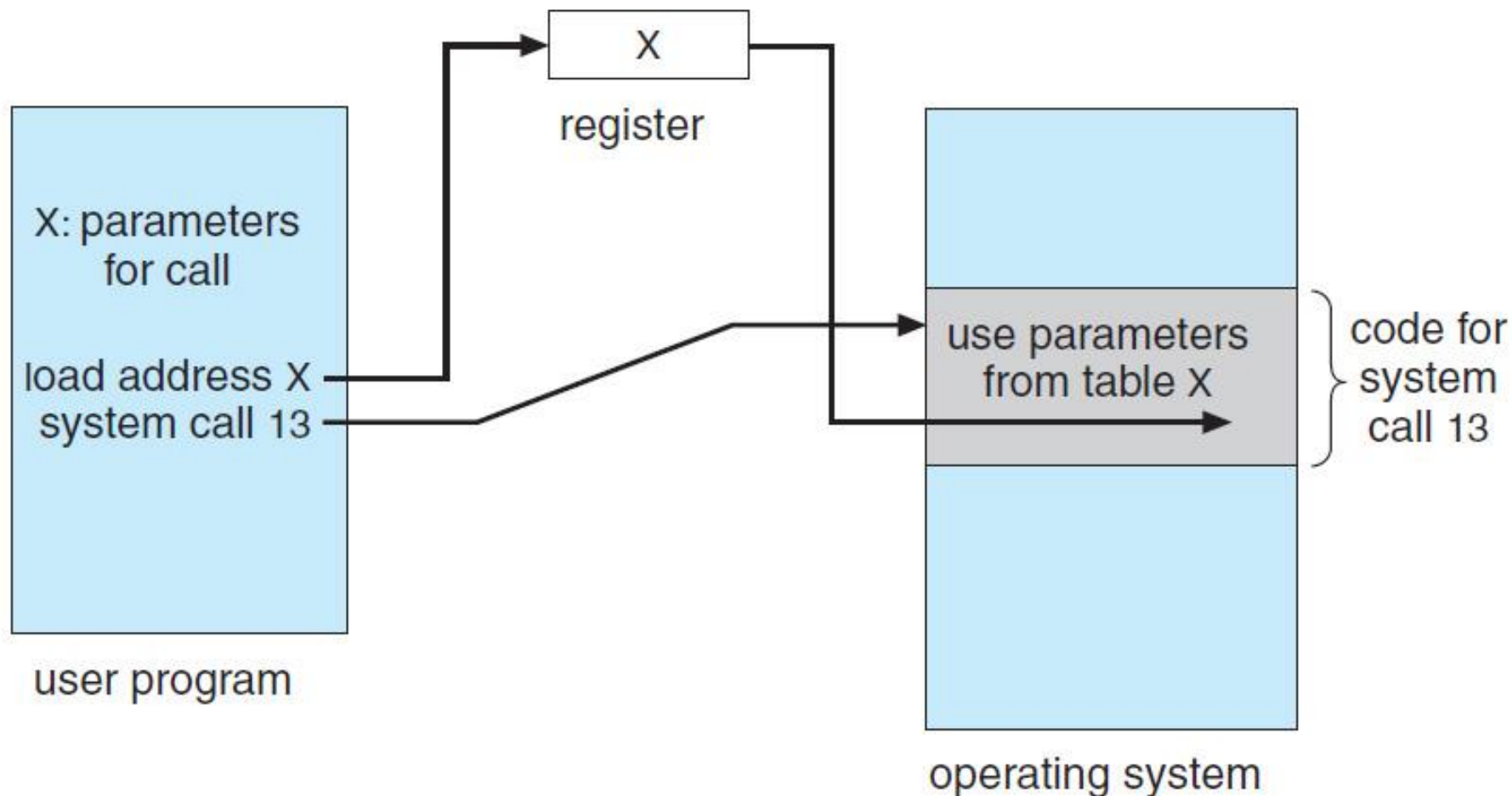


# System Call Interface Parameter Passing

- ❑ 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
  - Parameters can be passed in registers
    - In some cases, may be more parameters than registers
  - Parameters are generally stored in a block or table in memory and the address of the block is passed as a parameter in a register (**Used in Linux & Solaris**)
  - Parameters can be placed or **pushed** onto the **stack** by the program and **popped** off the stack by the OS
  - Block and stack methods do not limit the number or length of parameters being passed



# Parameter Passing via Table



# Types of System Calls

❑ System calls can be grouped into six major categories

- **Process control**
- **File manipulation**
- **Device manipulation**
- **Information maintenance**
- **Communications**
- **Protection**

# Process Control

- ❑ End (normal), abort (abnormal)
- ❑ Load, execute
- ❑ Create Process, terminate process
- ❑ Get process attributes, set process attributes
  - Job's priority
  - Its maximum allowable execution time and so on
- ❑ Wait for time
- ❑ Wait event, signal event
- ❑ Allocate and free memory

# File Management

- ☐ Create file
- ☐ Delete file
- ☐ Open
- ☐ Close
- ☐ Read, write, reposition
- ☐ Get file attributes
- ☐ Set file attributes

# Device management

- ☐ Request device
- ☐ Release Device
- ☐ Read, write, reposition
- ☐ Get file attributes
- ☐ Set file attributes

# Information maintenance

- ☐ Get time or date
- ☐ Set time or date
- ☐ Get system data
- ☐ Set system data
- ☐ Get process, file, or device attributes
- ☐ 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
- ❑ Transfer status information
- ❑ Attach or detach remote devices

# Protection

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



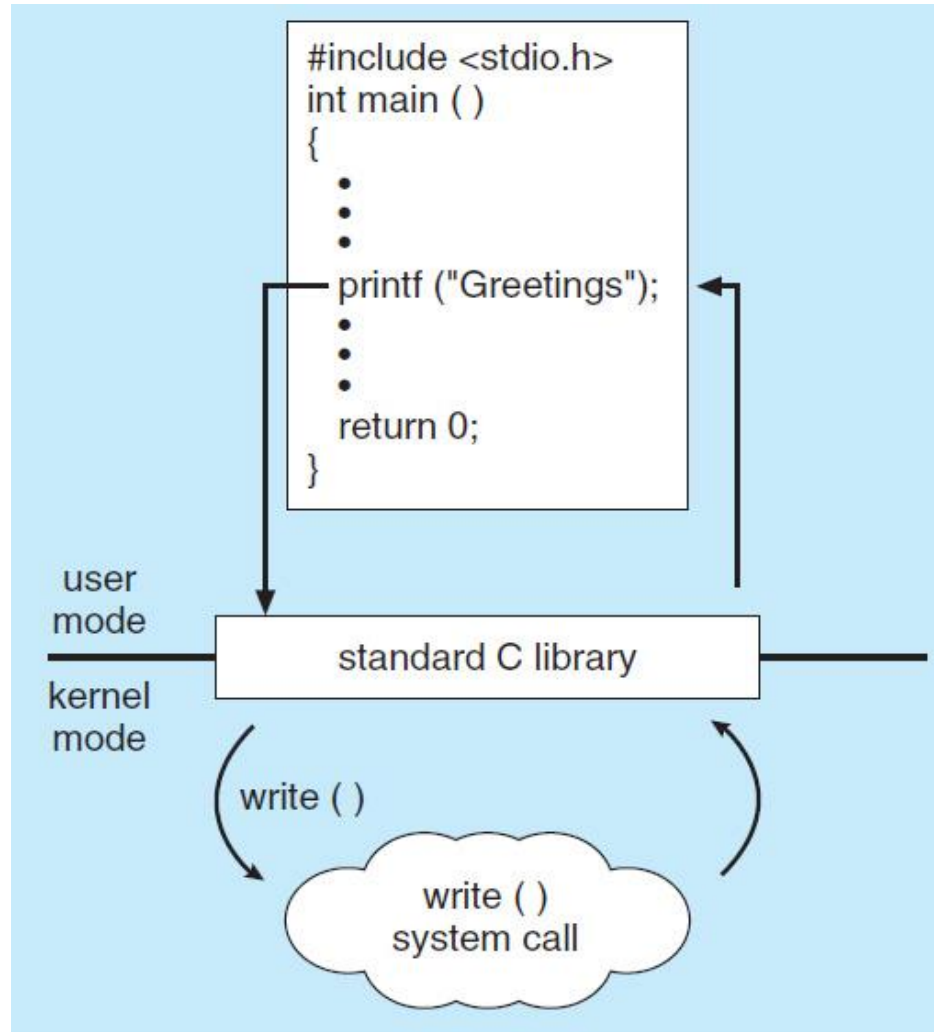
# Examples of Window and Unix System Calls

## EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

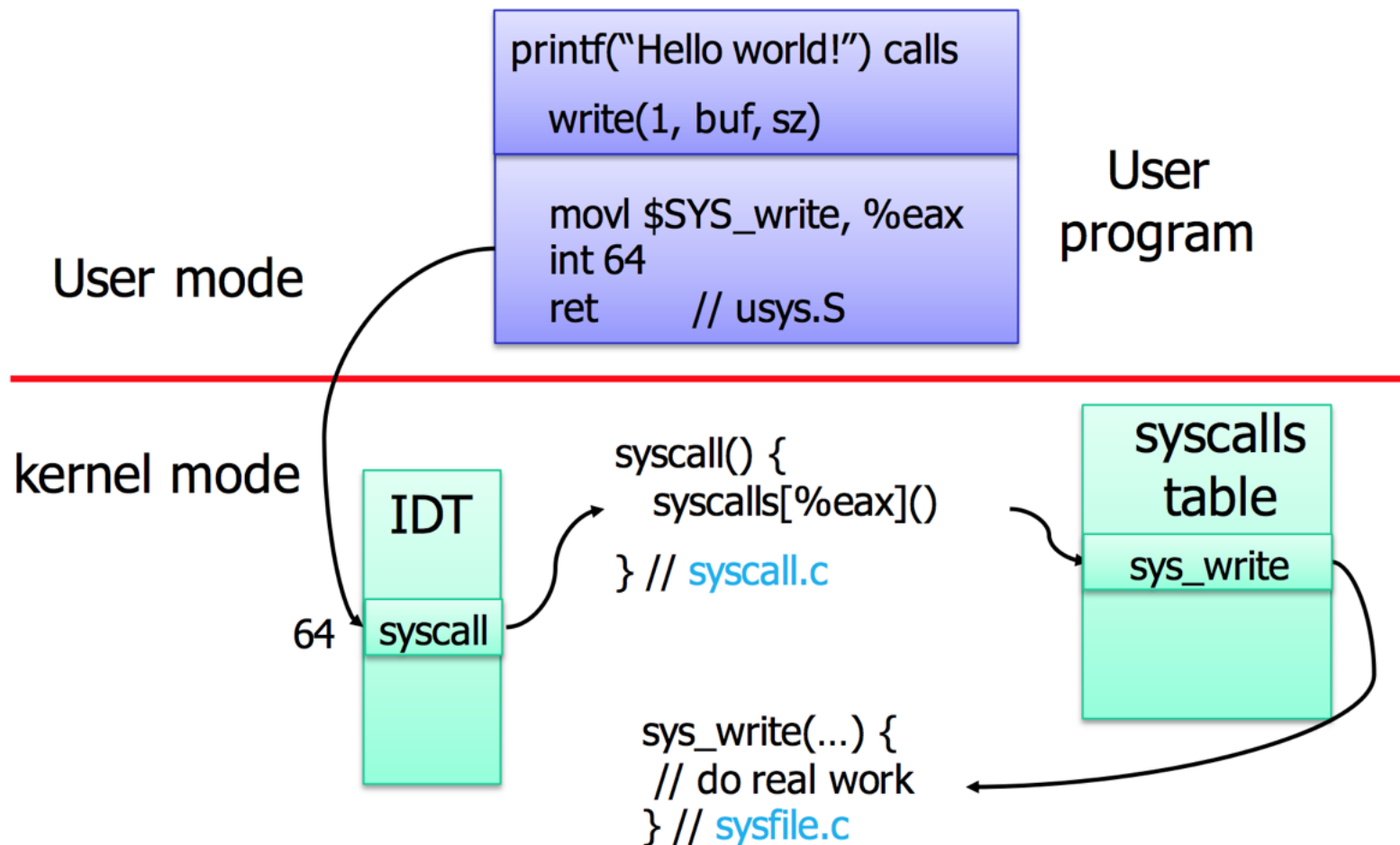
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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

# Examples of Standard C Library

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



# API – System Call Interface – OS Relationship



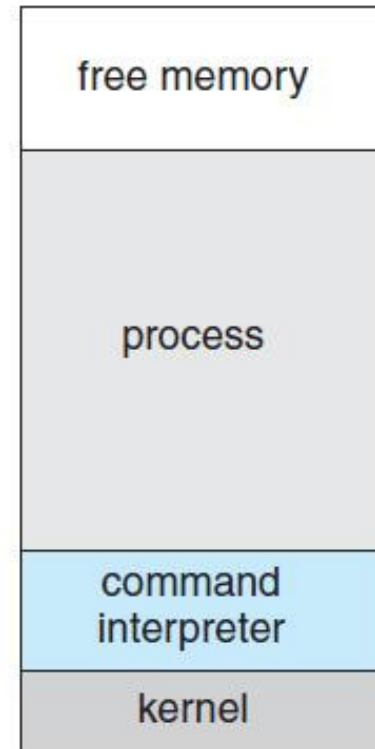
# MS-Dos Execution

- 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



(a)

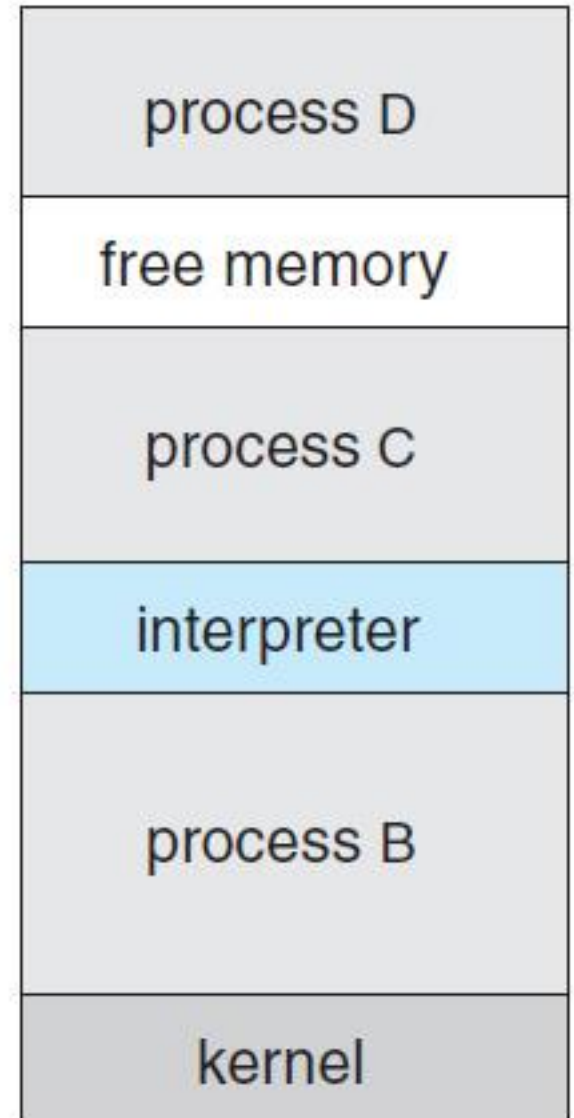
At system startup



(b)

Running a program

- Unix Variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes `fork()` system call to create process
  - Execute `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

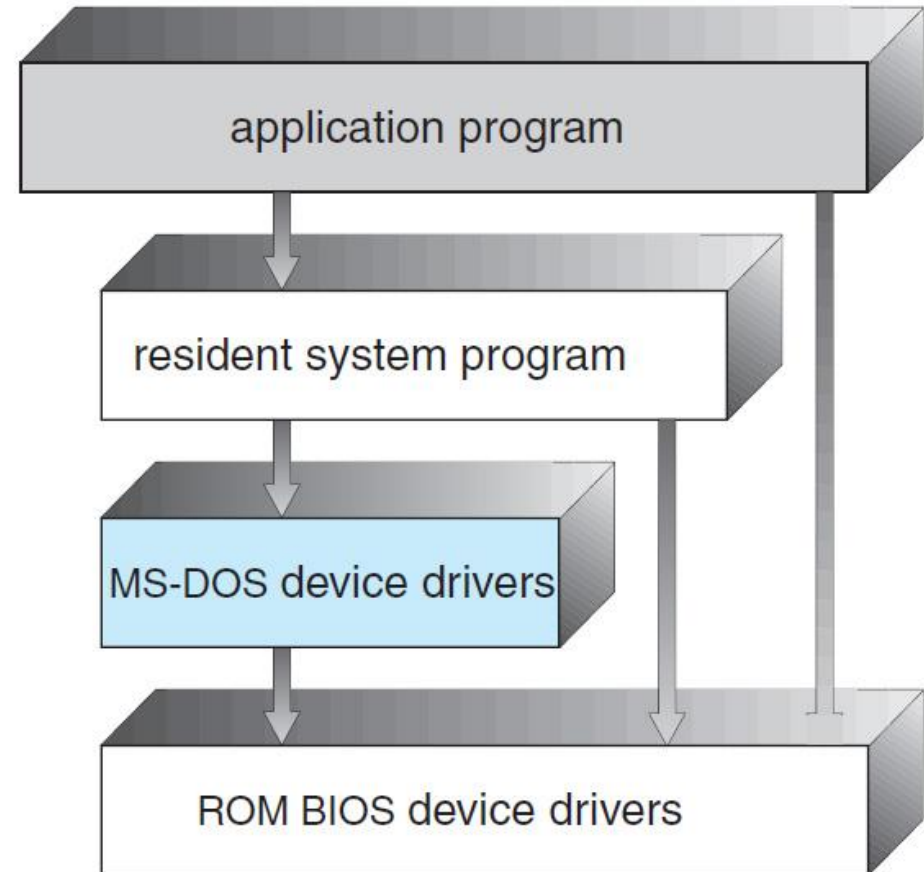


# System Programs

- These programs are not usually part of the OS kernel, but are part of the overall operating system
  - **File management:** these programs create, delete, copy, rename, print, dump, list, generally manipulate files and directories
  - **Status information:**
    - Some programs simply request the date and time, and other simple requests
    - Others provide detailed performance, logging, and debugging information
    - The output of these files is often sent to a terminal window or GUI window
  - **File modification:**
    - Programs such as text editors are used to create, and modify files
  - **Communications:**
    - These programs provide the mechanism for creating a virtual connect among **processes**, **users**, and **other computers**
    - **Ex:** Email and Web browsers

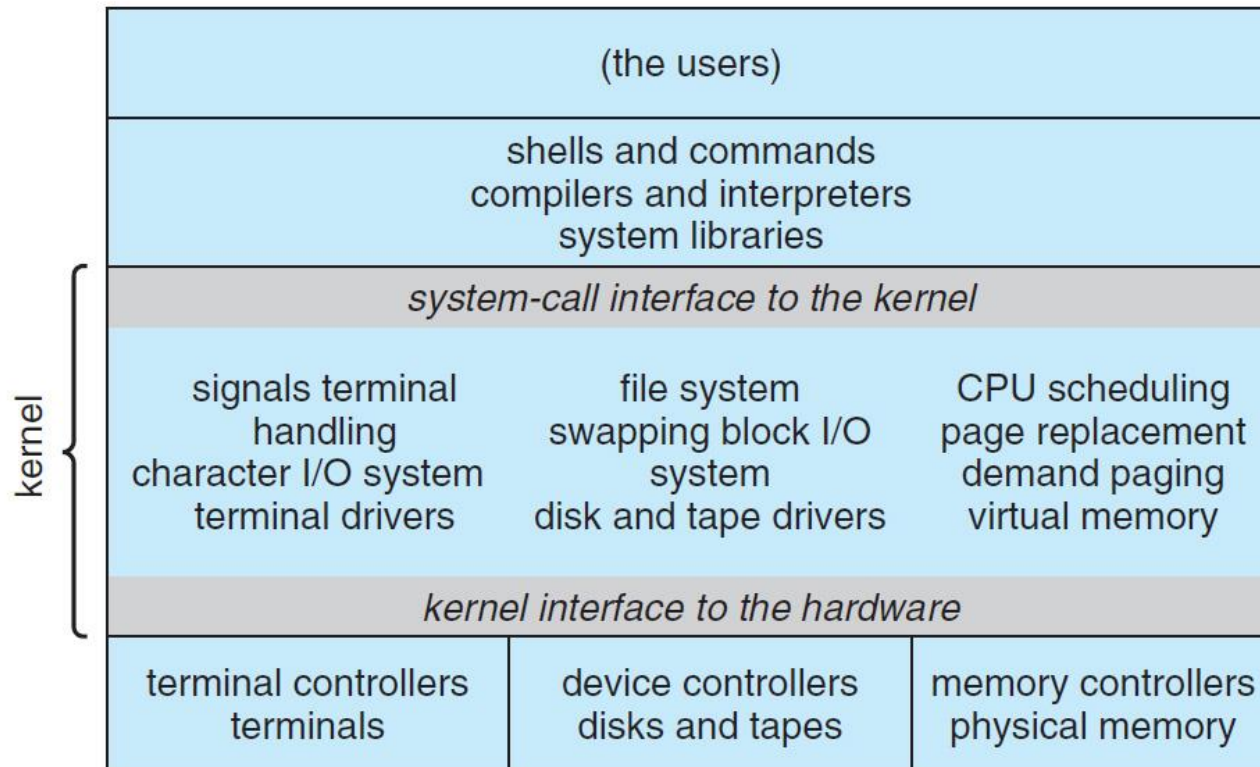
# Operating-System Structure: Simple Structure

- 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
  - There was no [CPU Execution Mode](#) (user and kernel)
  - so errors in applications could cause the whole system to crash



# Operating-System Structure: Simple Structure

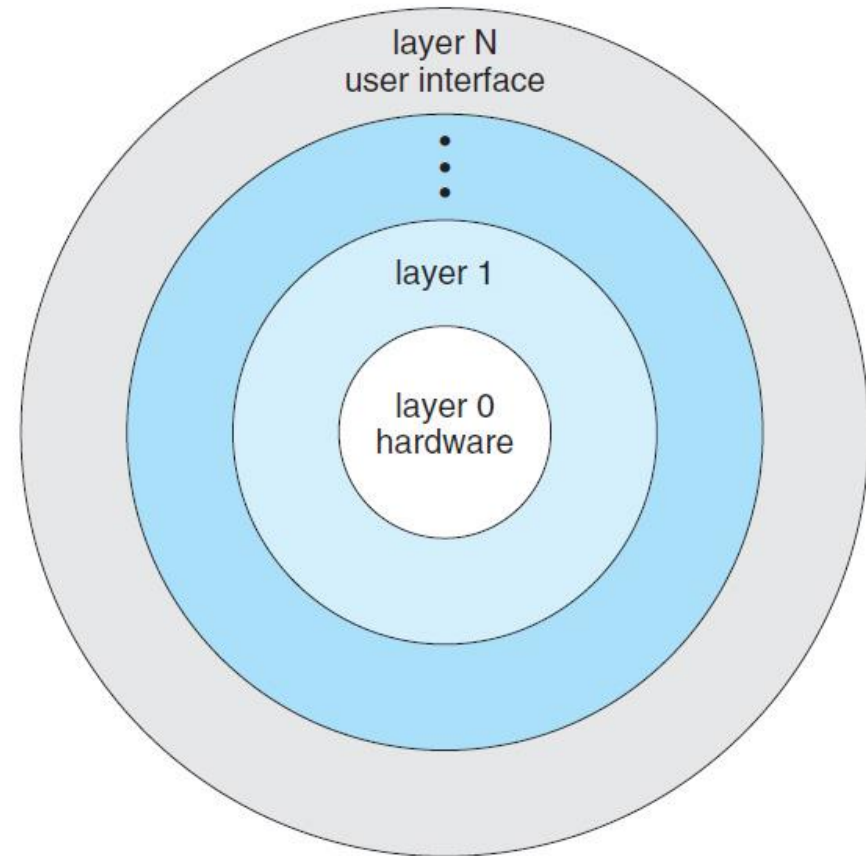
- Traditional Unix consists of two separable parts:
  - The **kernel** is further separated into a series of **interfaces** and **device drivers**
  - The **system programs**





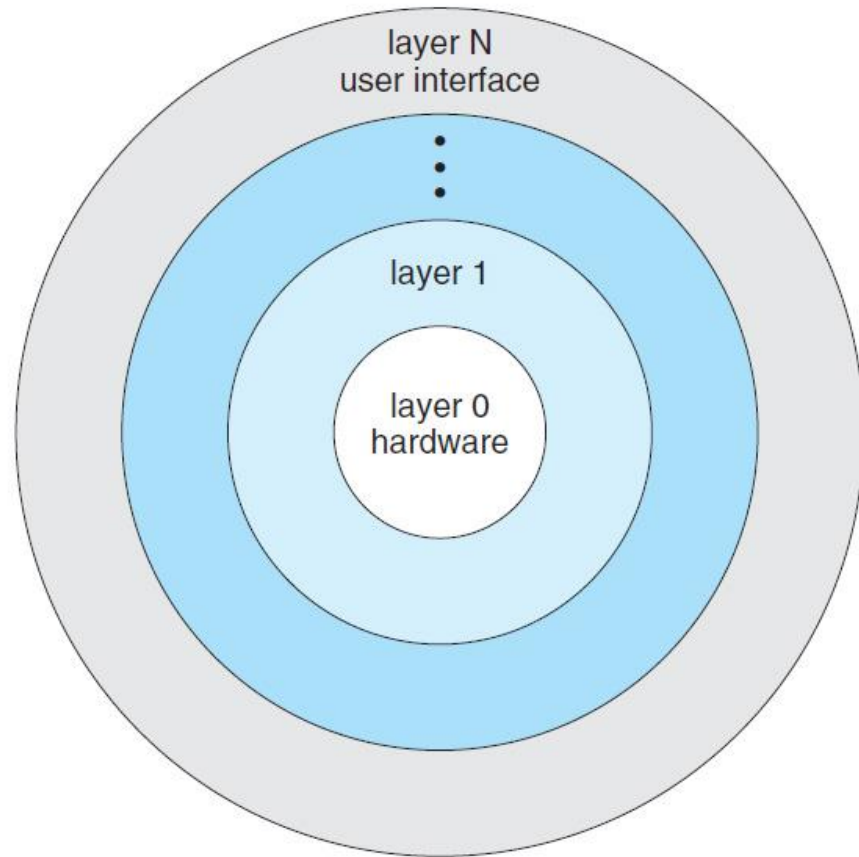
# Operating-System Structure: Layered Approach

- Layered Approach breaks up the operating system into various layers (**Levels**)
  - This allows implementers to change the inner workings, and increase modularity
  - As long as the external interface of the routines don't change, developers have more freedom to change the inner workings of the routines
  - The bottom layer (**level 0**) is the hardware, whereas the highest layer (**layer N**) is the user interface

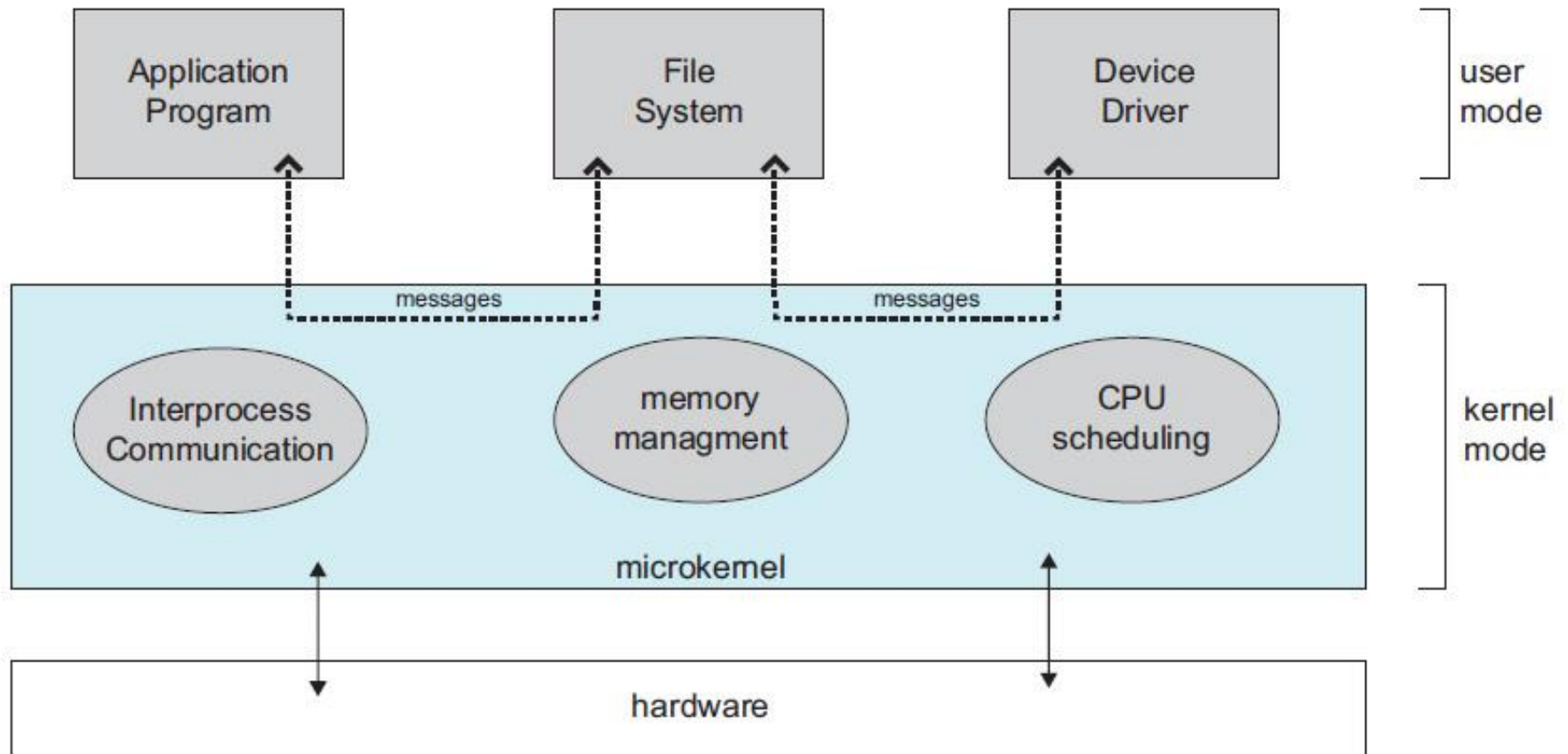


# Operating-System Structure: Layered Approach

- The main **advantage** is simplicity of construction and debugging
- The main **difficulty** is defining the various layers
- The main **disadvantage** is that the OS tends to be less efficient than other implementations



# Operating-System Structure: **Microkernels**



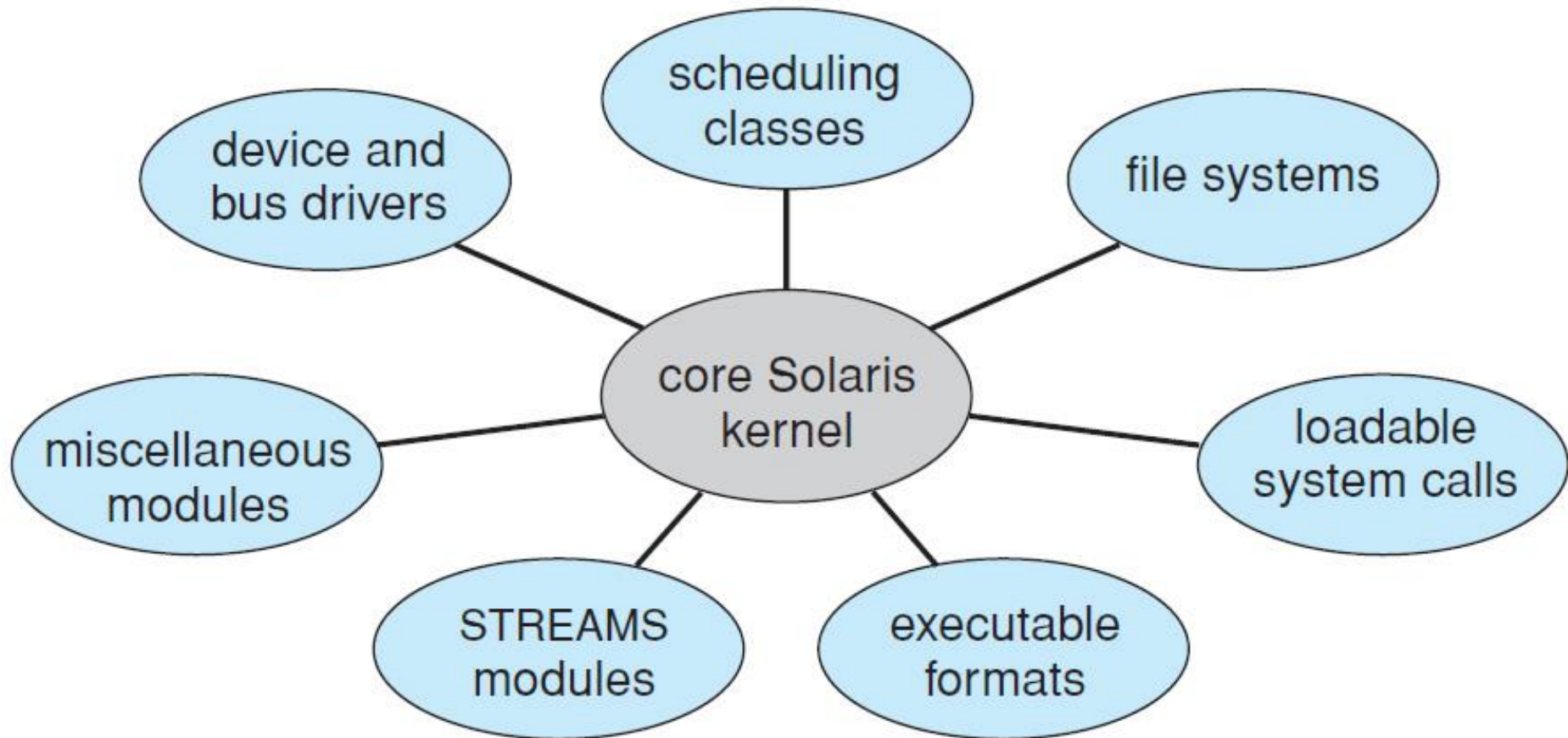
# Operating-System Structure: **Microkernels**

- Microkernel structures the operating system by removing all nonessential portions of the kernel, and implementing them as **system** and **user-level** programs
  - Microkernel provides minimal process and memory management, and a communications facility
  - Communication between the client program and the various services that are also running in user space is provided by **message passing**
- **Advantages:**
  - Extending the operating system becomes much easier
  - Any changes to the kernel tend to be fewer, since the kernel is smaller
  - The microkernel also provides more security and reliability
- **Disadvantages:**
  - Poor performance due to increased system overhead from message passing

# Operating-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

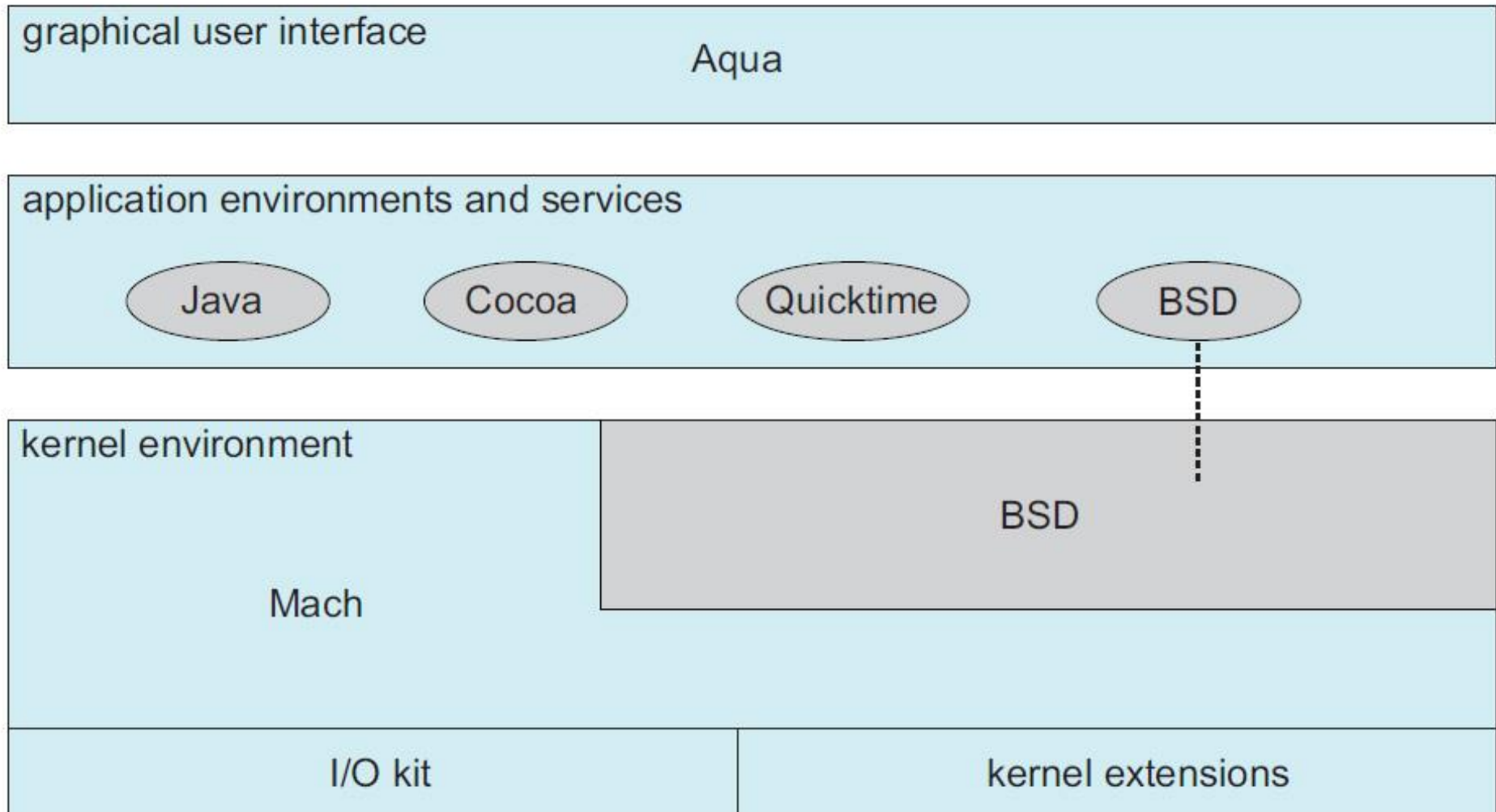
# Operating-System Structure: Modules



# Operating-System Structure: Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Window mostly monolithic, plus microkernel for different subsystem **personalities**
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment

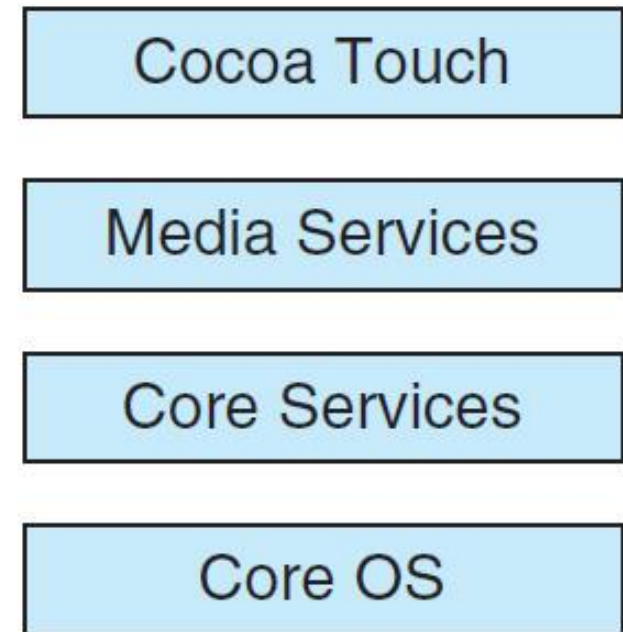
# Operating-System Structure: Hybrid Systems





# Operating-System Structure: iOS

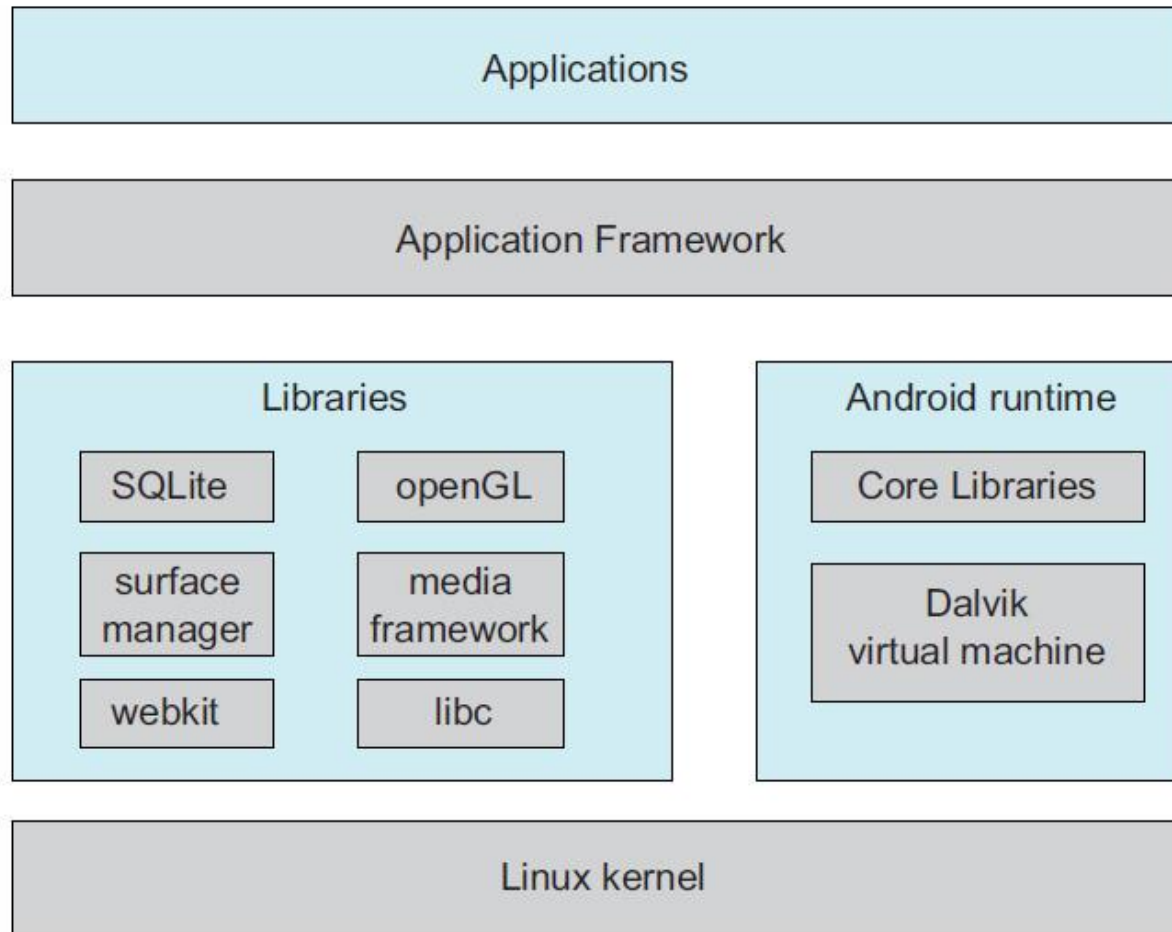
- Apple mobile OS for iPhone, iPad
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs Intel)
  - **Cocoa Touch**: Objective-C API for developing apps
  - **Media services**: layer for graphics, audio, video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



**Architecture of Apple's iOS**

# Operating-System Structure: **Android**

- Developed by Open Handset Alliance (Mostly Google)
  - Open Source



# System Boot

- Booting the system is done by loading the kernel into main memory, and starting its execution
- The CPU is given a reset event, and the instruction register is loaded with a predefined memory location, where execution starts
  - The initial bootstrap program is found in the BIOS read-only memory
  - This program can:
    - Run diagnostics
    - Initialize all components of the system
    - Load and start the OS loader called **boot strapping**
  - The loader program loads and start the OS

# System Boot

- When the OS starts, it:
  - Sets up needed data structures in memory
  - Sets several registers in the CPU
  - Creates and starts the first user level program
- From this point, the OS only runs in response to interrupts