



**CHRIST**  
(DEEMED TO BE UNIVERSITY)  
BANGALORE, INDIA

# BCA233: Operating System

## Unit – 1(Contd..)

### Chapter 2 – Operating-System Structures

#### MISSION

CHRIST is a nurturing ground for an individual's holistic development to make effective contribution to the society in a dynamic environment

#### VISION

Excellence and Service

#### CORE VALUES

Faith in God | Moral Uprightness  
Love of Fellow Beings  
Social Responsibility | Pursuit of Excellence

## Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Structure
- System Boot

# 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

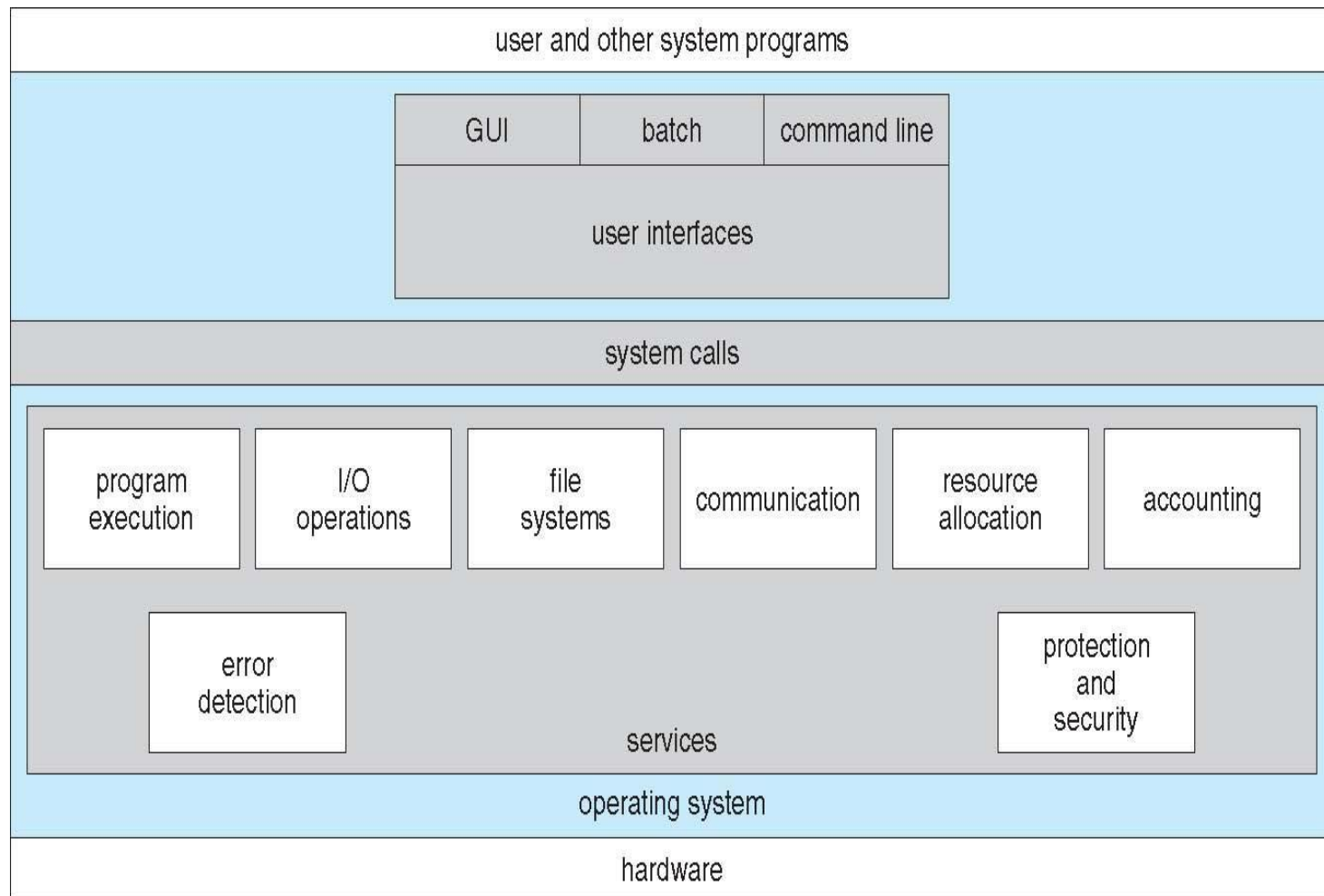
## Operating System Services (Cont.)

- **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

# 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 Operating System Interface - GUI

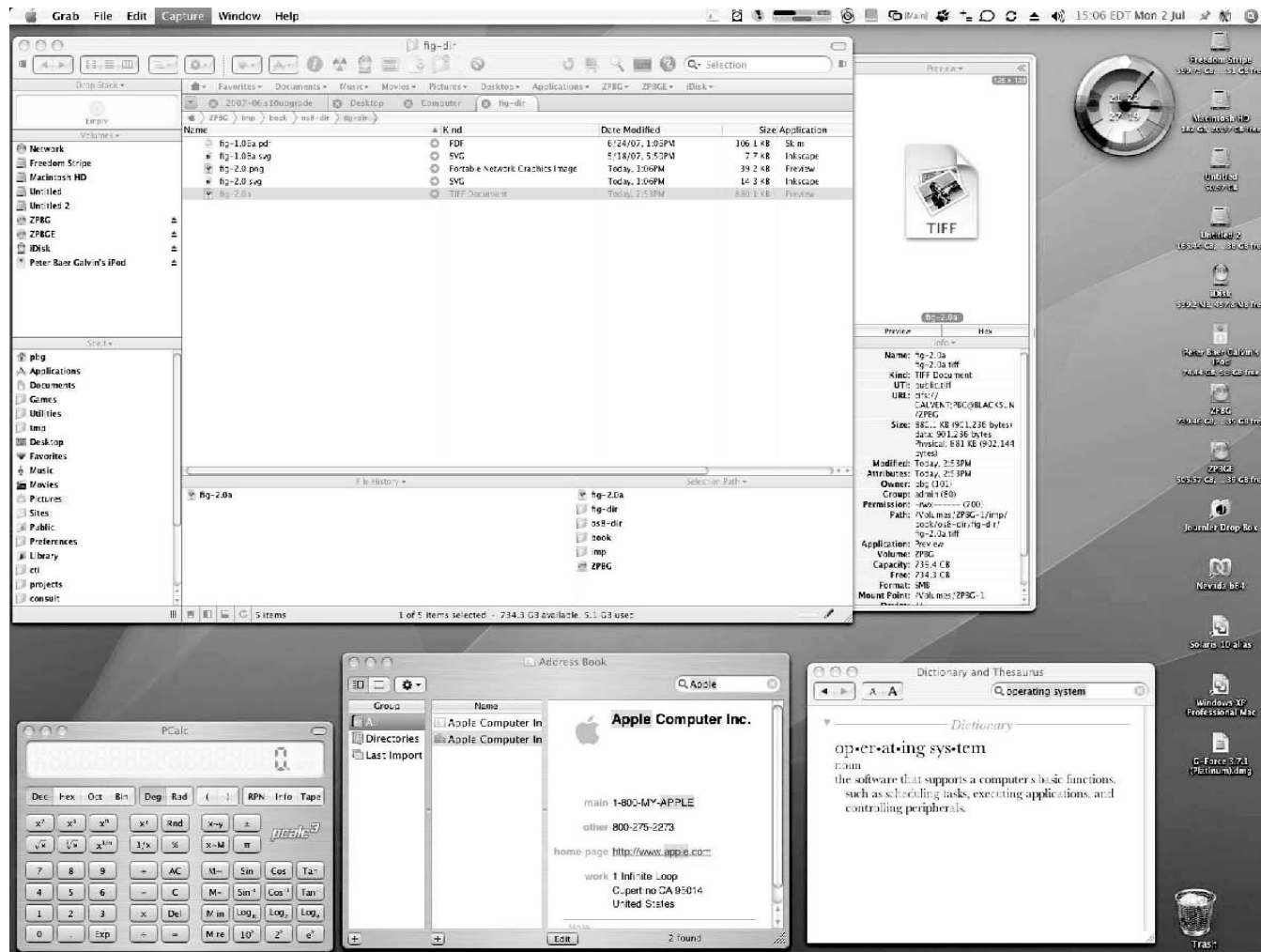
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))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

# Touchscreen Interfaces

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



# The Mac OS X GUI

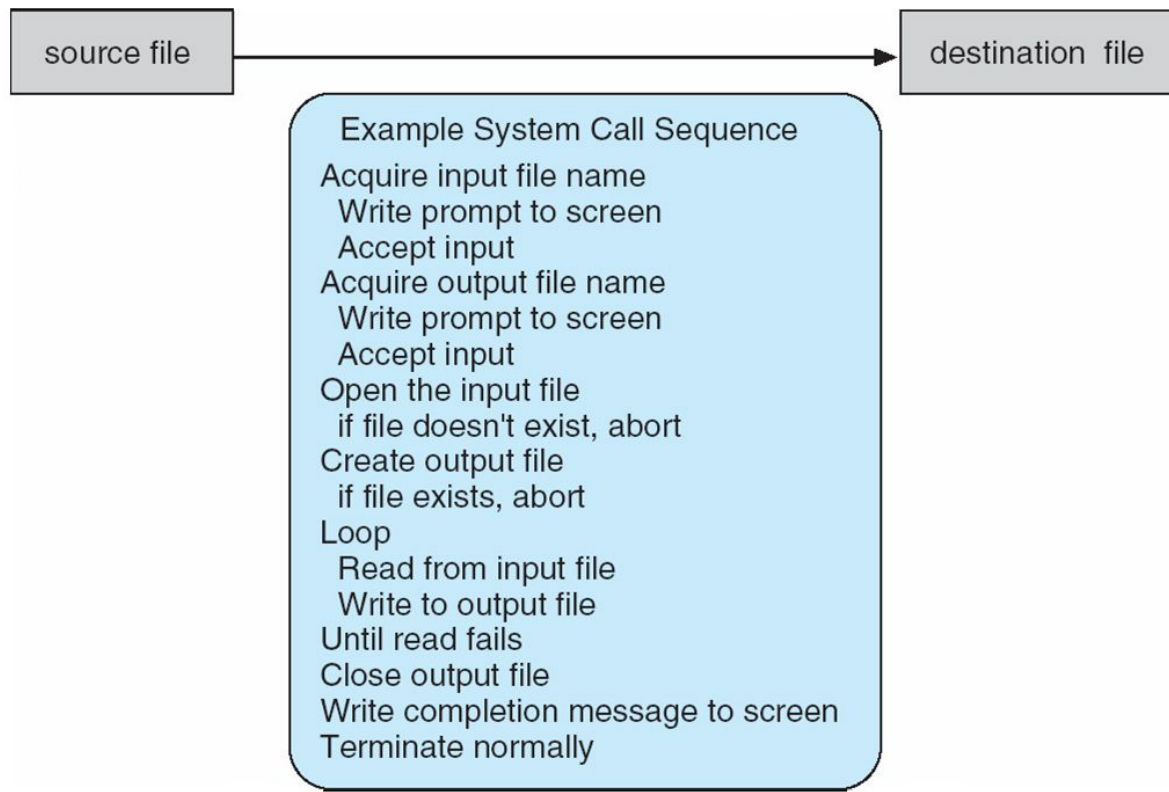


## System Calls

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

## Example of System Calls

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



# Example of Standard API

## EXAMPLE OF STANDARD API

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

```
man read
```

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

<pre>#include &lt;unistd.h&gt;</pre>		
<pre>ssize_t</pre>	<pre>read(int fd, void *buf, size_t count)</pre>	
<div></div>	<div></div>	<div></div>
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

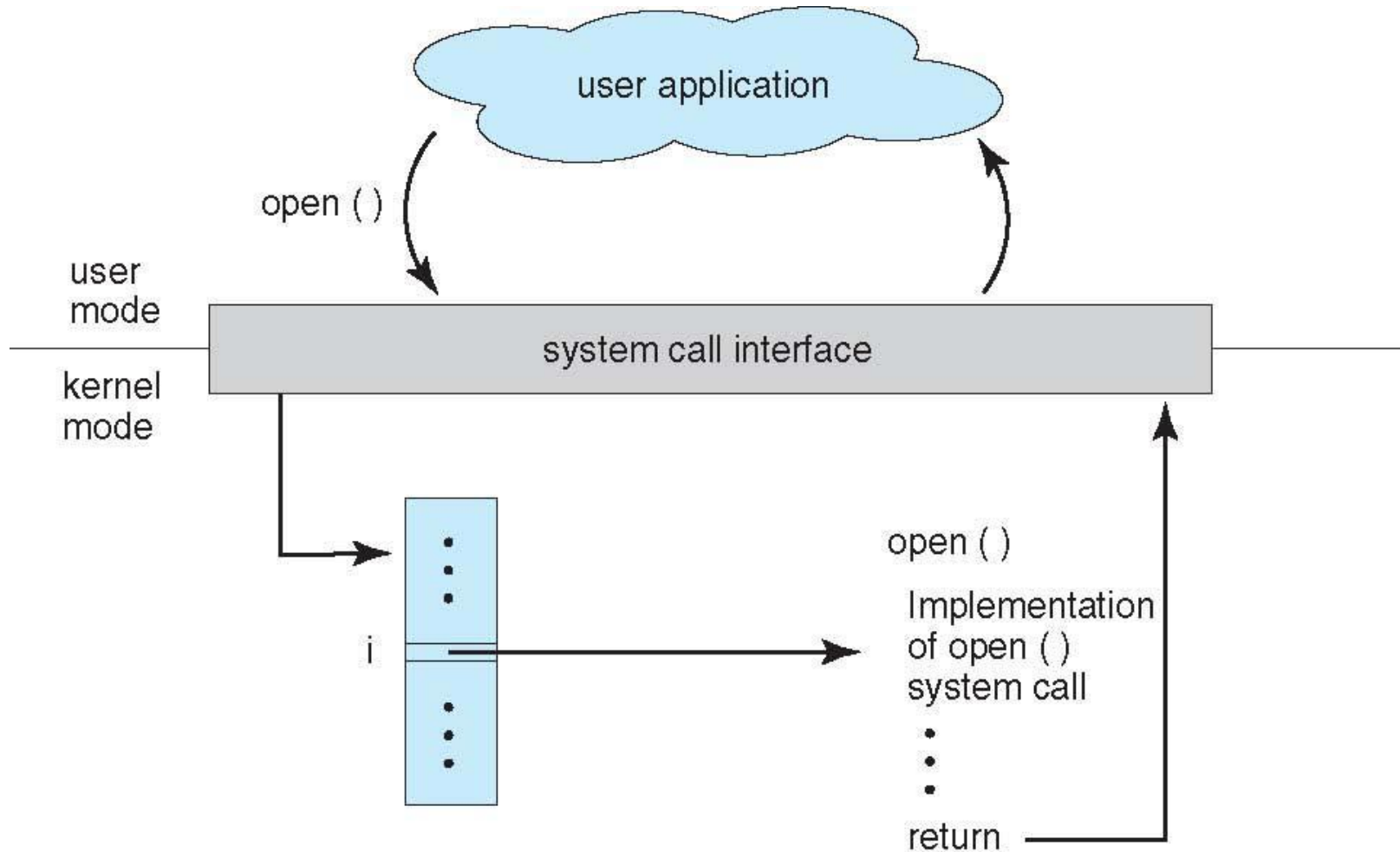
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

## System Call Implementation

- Typically, a number 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

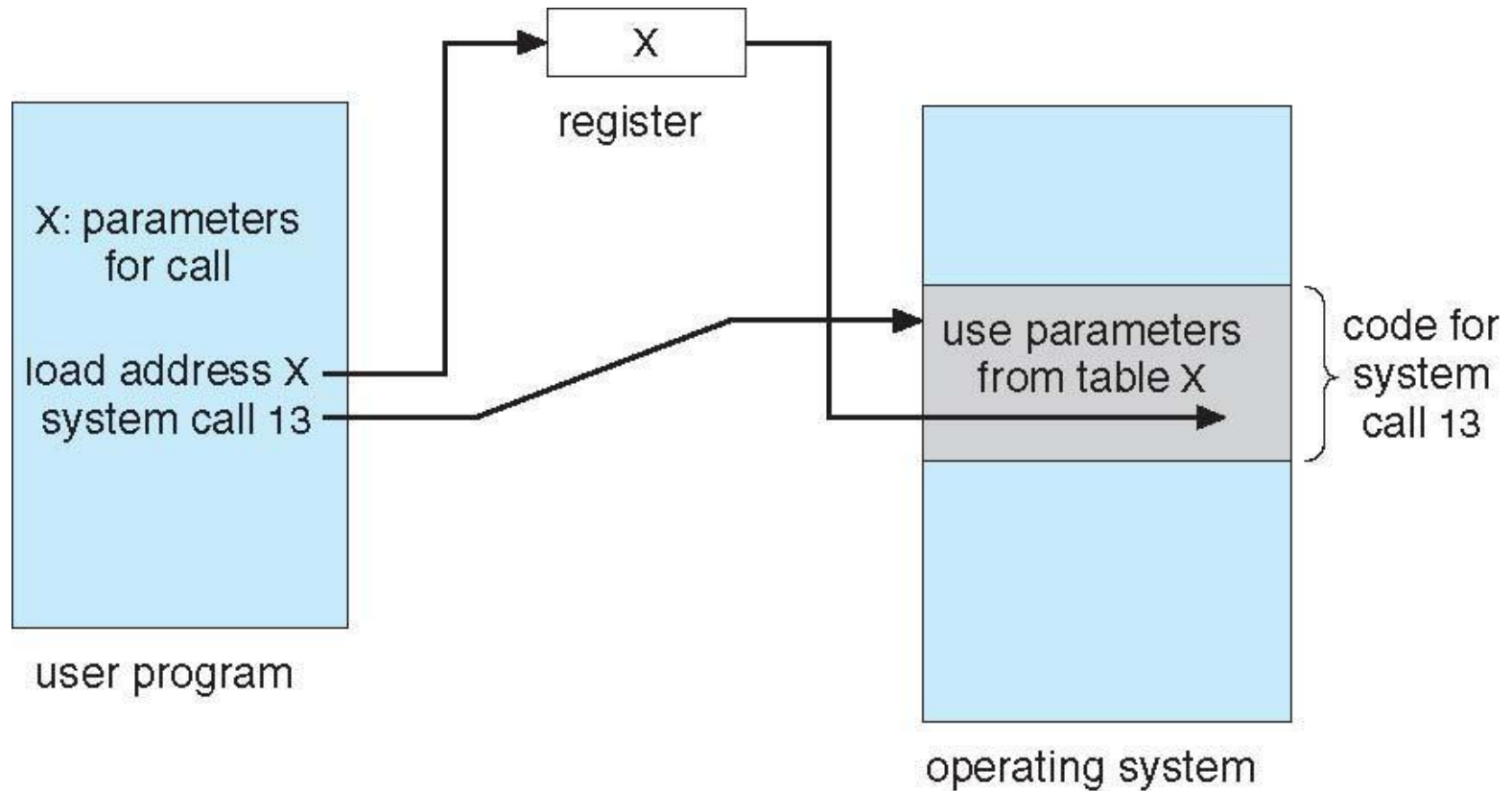




## System Call Parameter Passing

- 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

# Examples of Windows and Unix System Calls

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

## System Programs

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



## System Programs (Cont.)

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

## System Programs (Cont.)

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

## System Programs (Cont.)

- **Background Services**

- Launch at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as **services**, **subsystems**, **daemons**

- **Application programs**

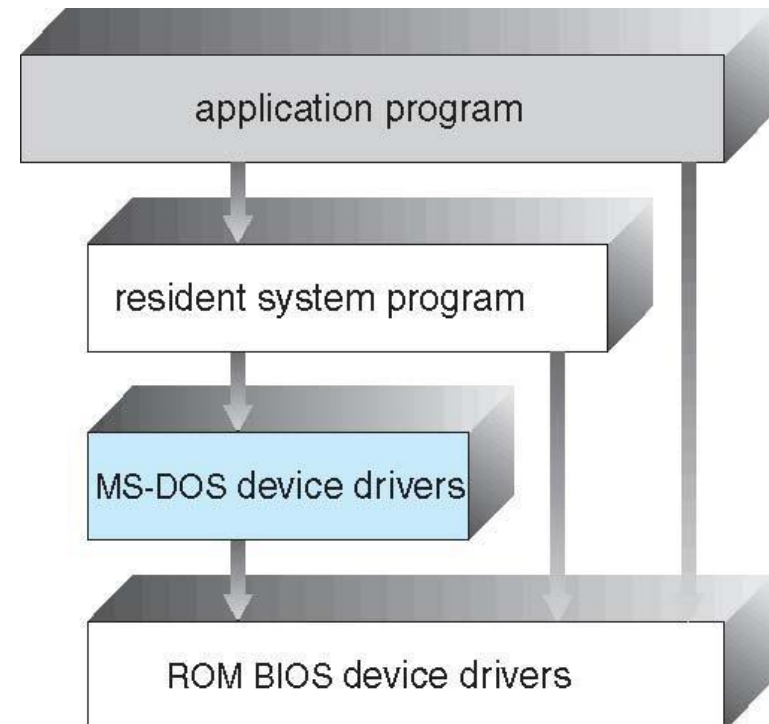
- Don't pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke

# 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



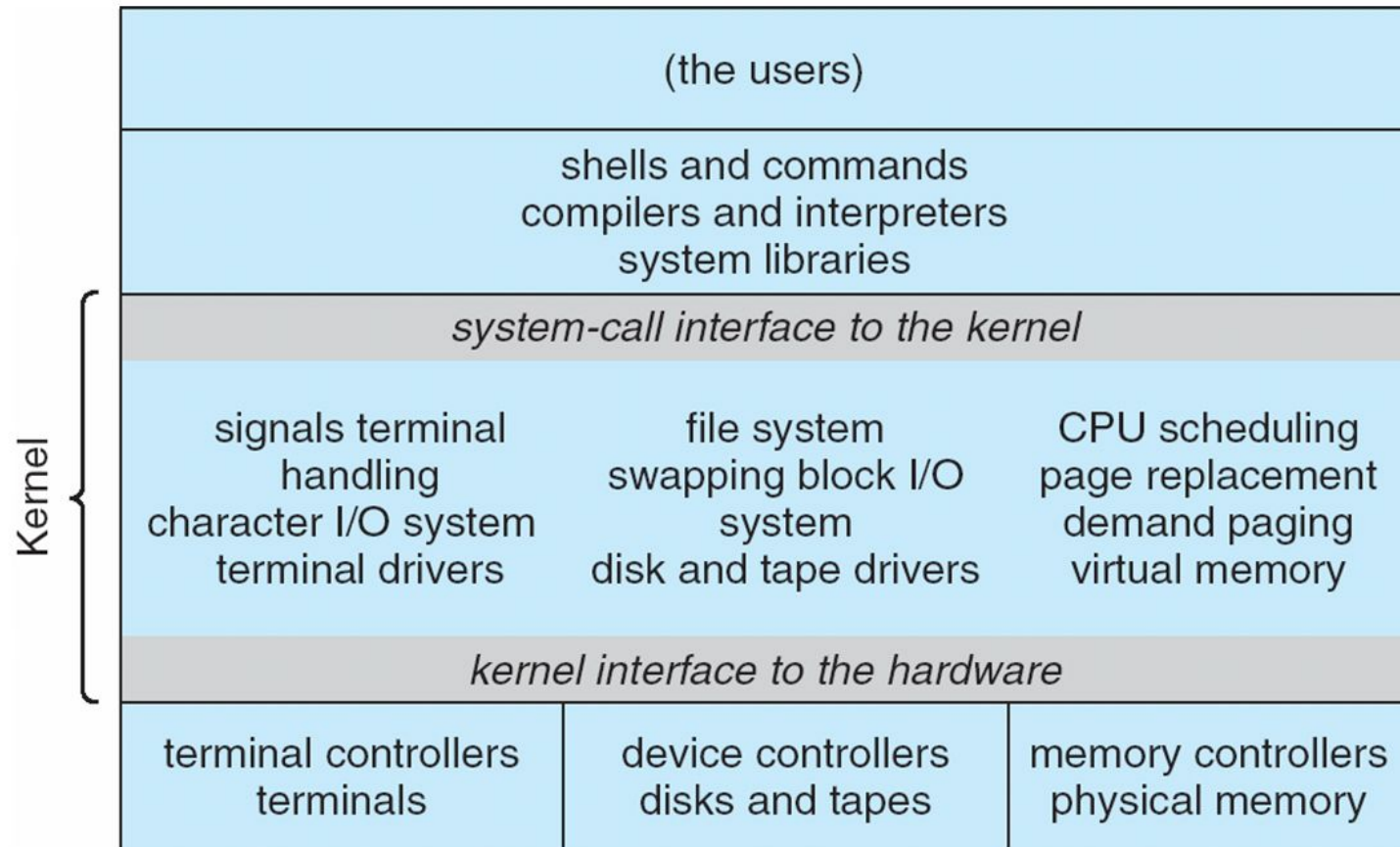
## Non Simple Structure - UNIX (Monolithic structure)

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

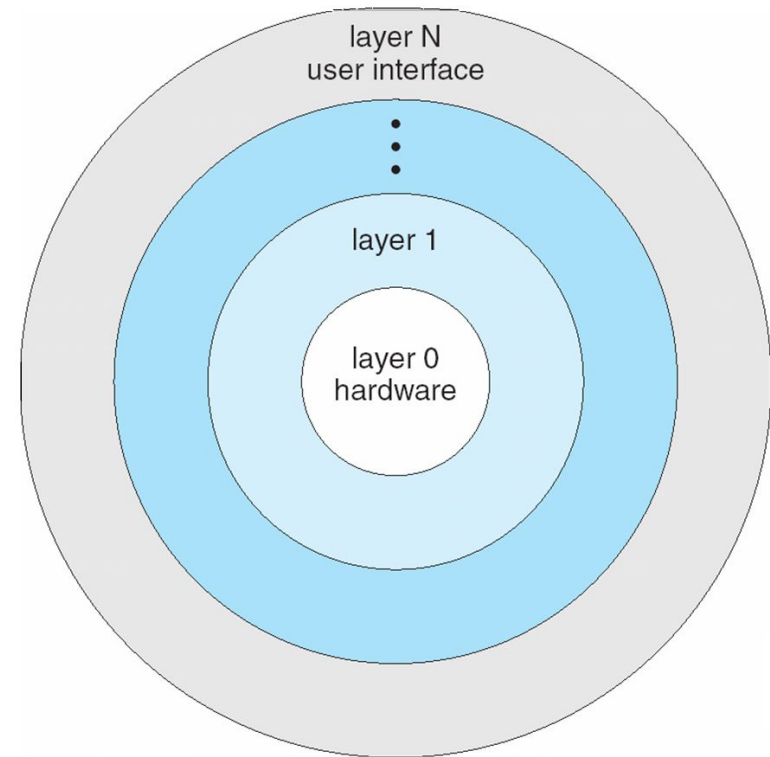
# Traditional UNIX System Structure

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

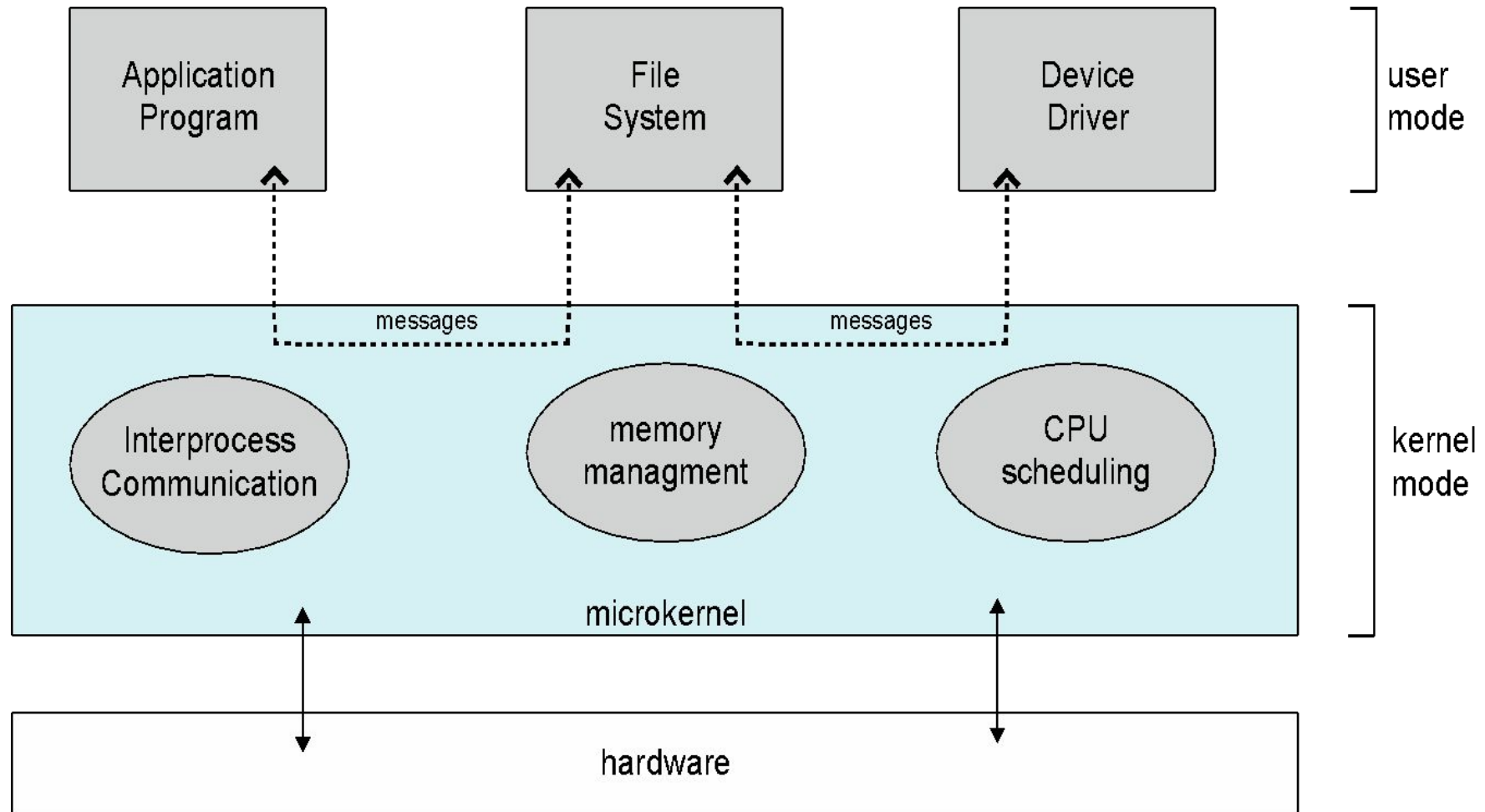




## Microkernel System Structure

- 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

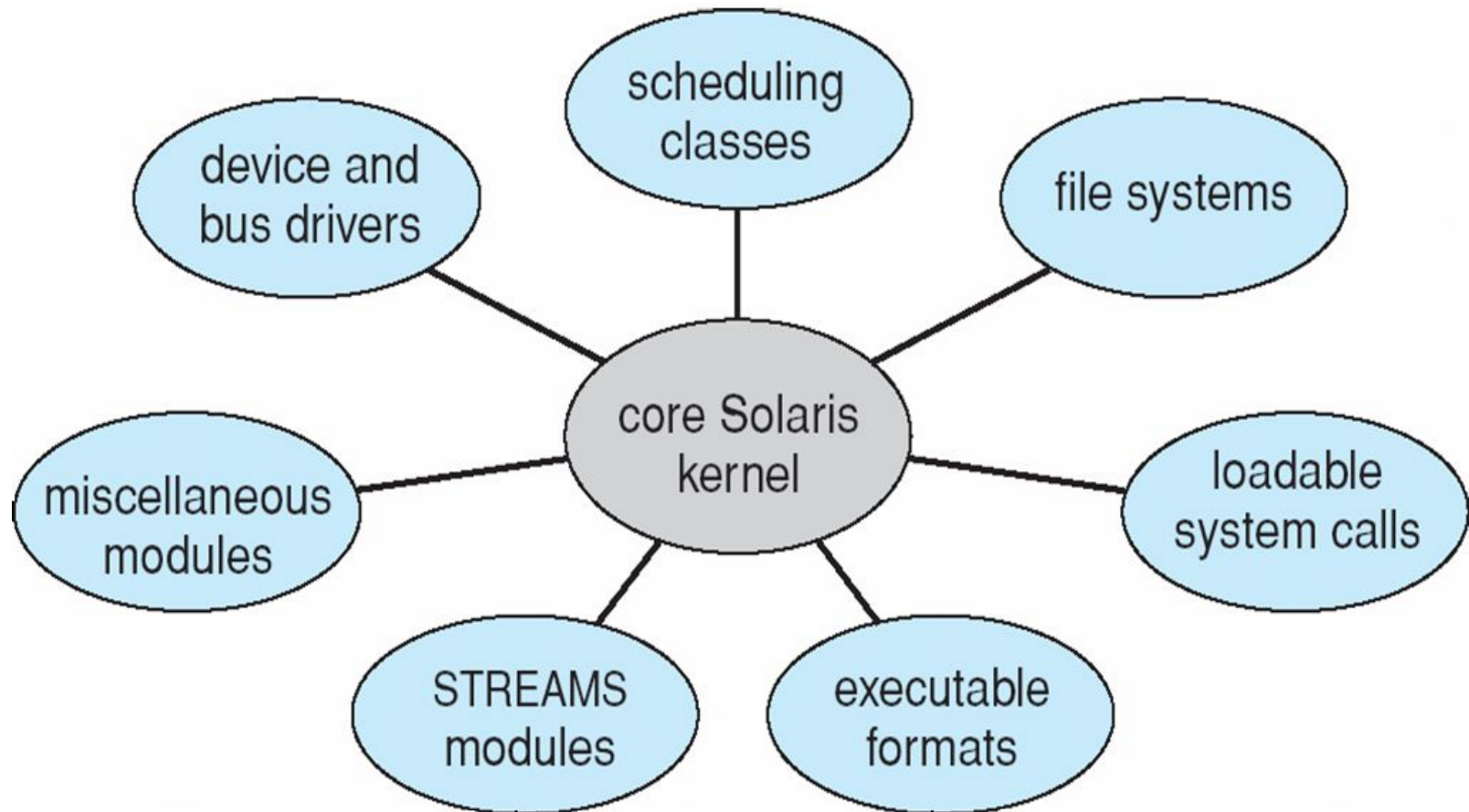
# Microkernel System Structure



## Modules (Modular approach)

- 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

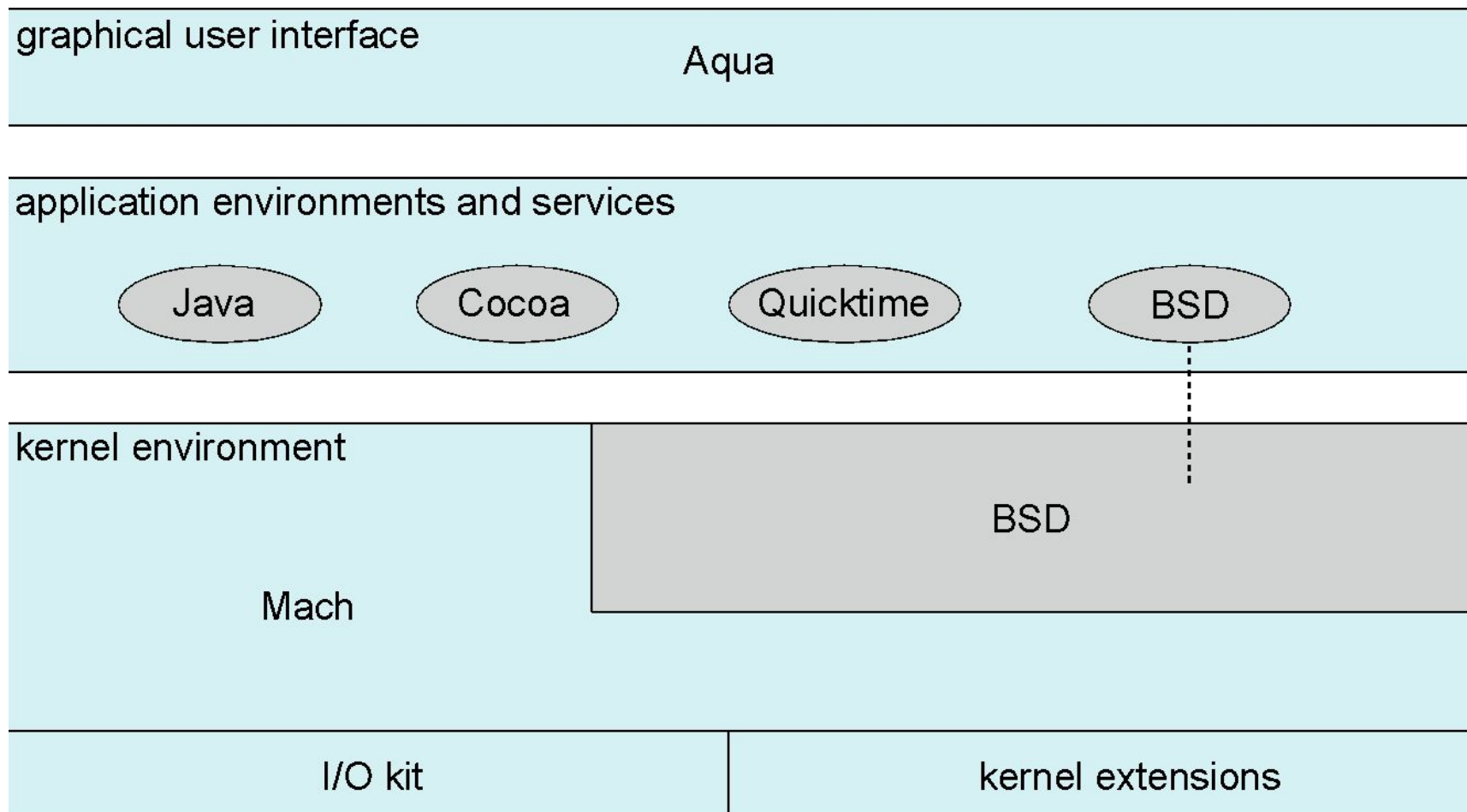
## Solaris Modular Approach



## 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

Cocoa Touch

Media Services

Core Services

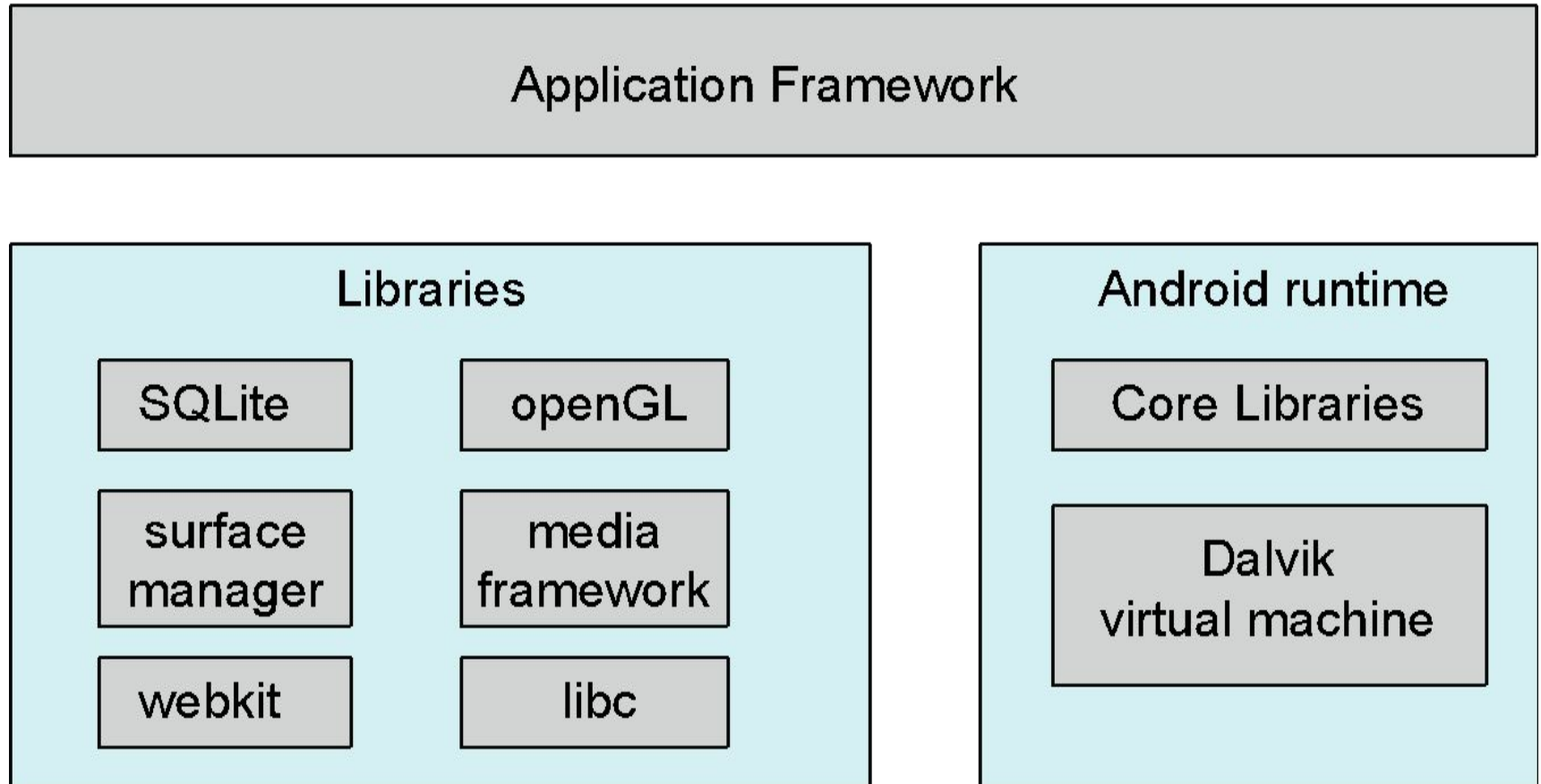
Core OS

## 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



## System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - 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
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

# End of Chapter 2