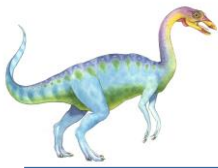


Chapter 2: Operating-System Structures





Operating System Services

- Operating System: Operating System Structure Topics discussed:
 1. **Multiprogramming.**
 2. **Time Sharing (Multitasking)**
- 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





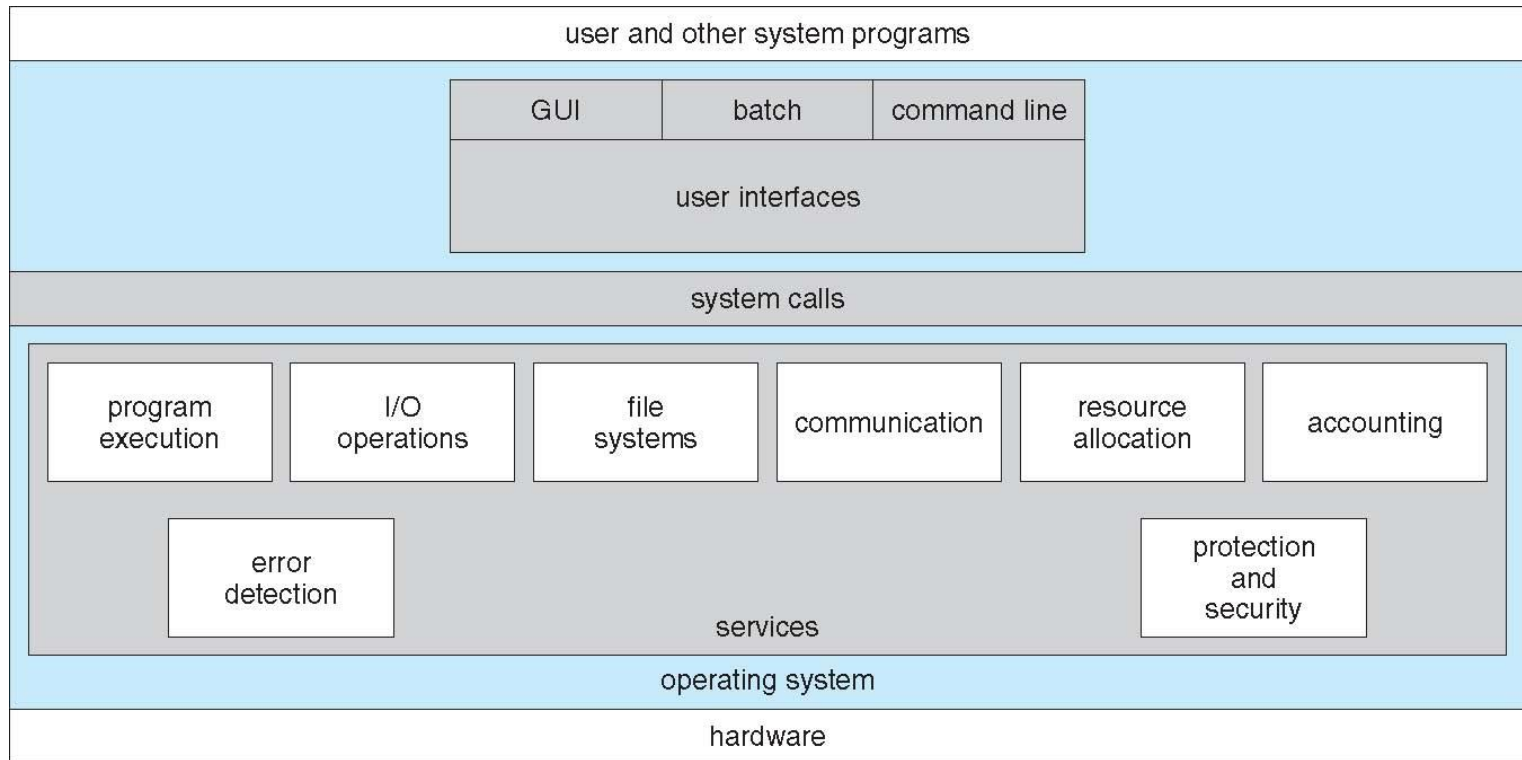
Operating System Services (Cont.)

- 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





Bourne Shell Command Interpreter

```
Default
New Info Close Execute Bookmarks

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$
```





System Calls

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

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





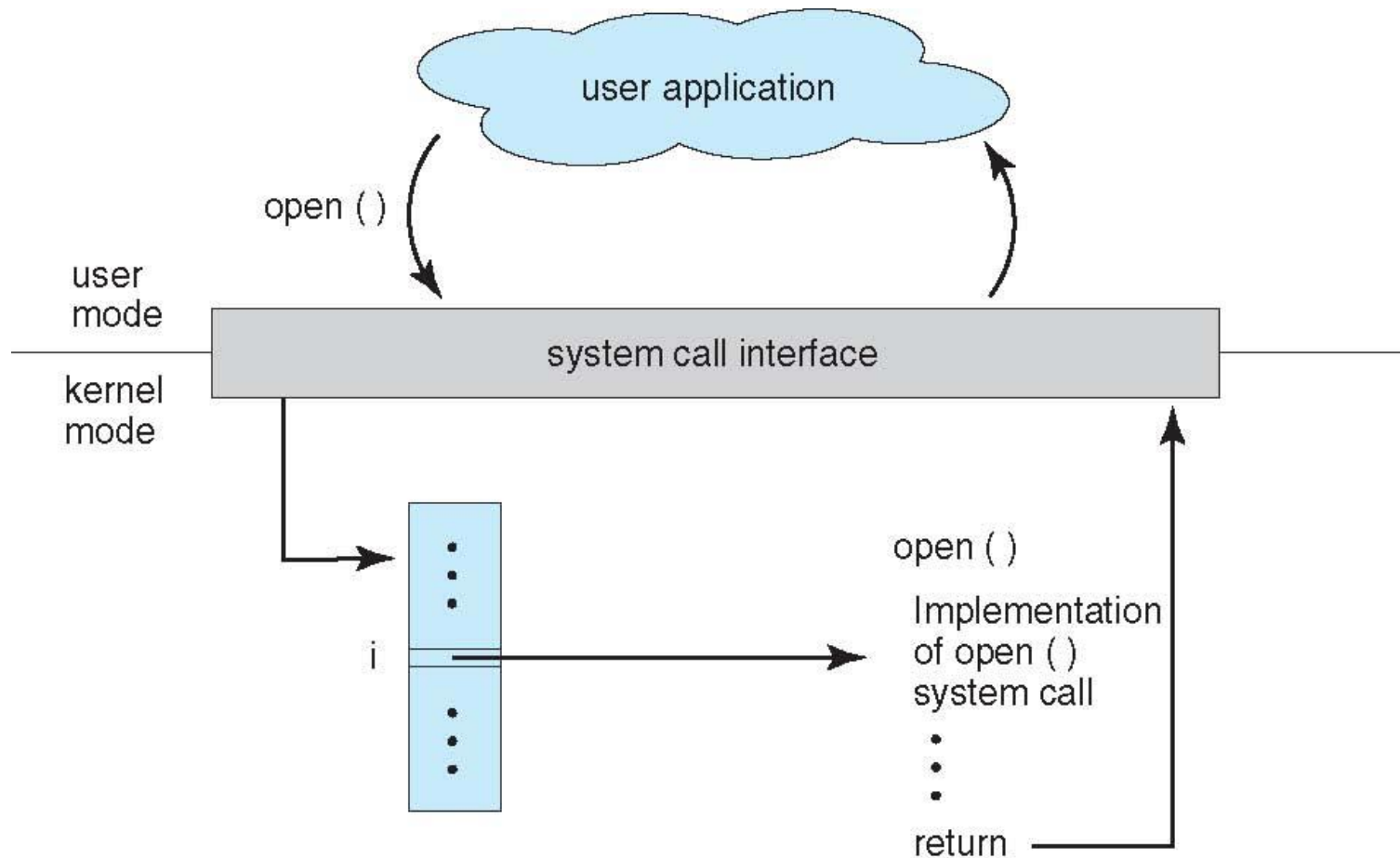
System Call Implementation

- 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
 - 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





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
- **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
- 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





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





Operating System Design and Implementation

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





Operating System Structure

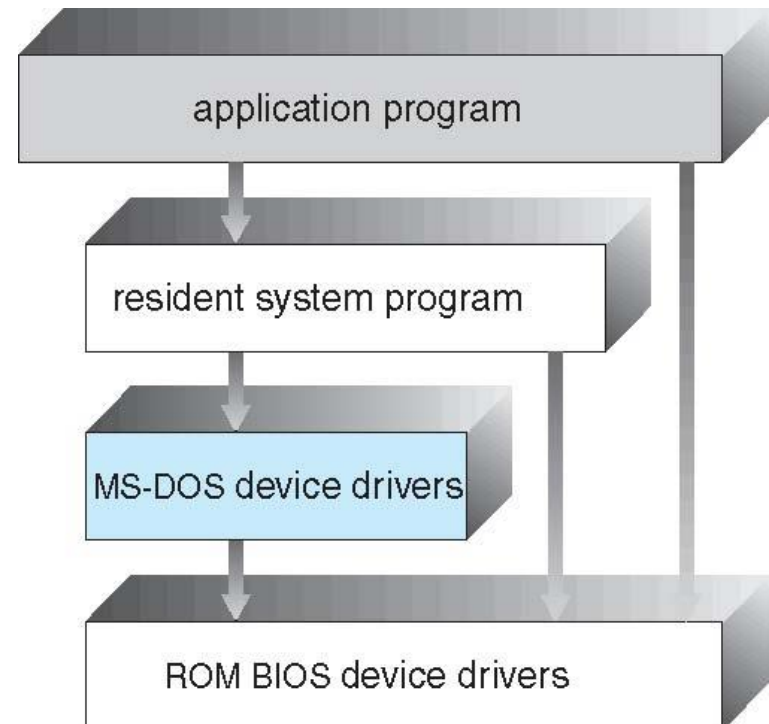
- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure – MS-DOS
 - More complex -- UNIX
 - Layered – an abstraction
 - Microkernel -Mach





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

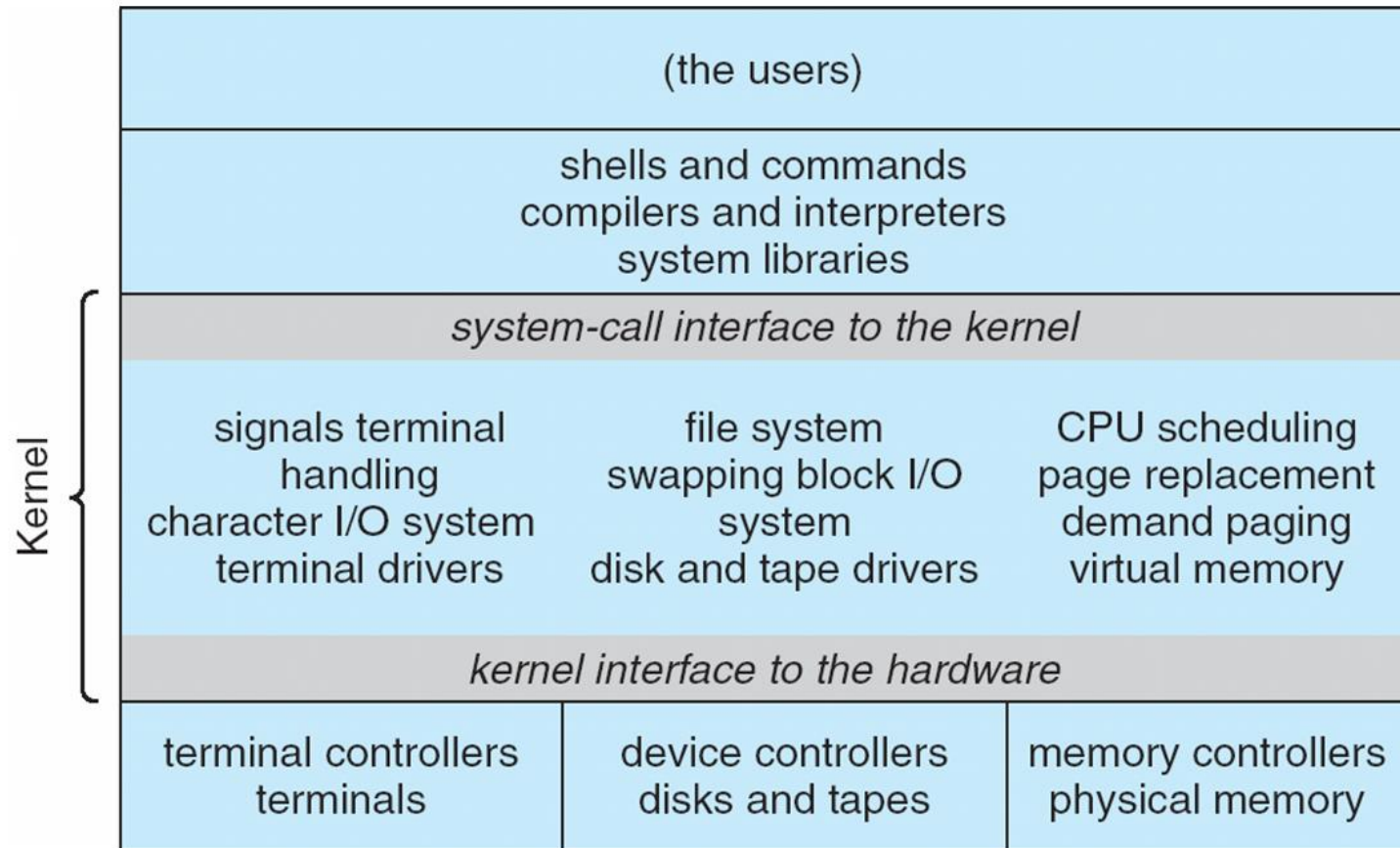
- Systems programs
- The kernel
 - ▶ Consists of everything below the system-call interface and above the physical hardware
 - ▶ Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level





Traditional UNIX System Structure

Beyond simple but not fully layered





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
- Disadvantages:
 - Performance overhead of user space to kernel space communication





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**)





Operating-System Debugging

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

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





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**

