

# Chapter 2: 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
    - ▶ For any error, OS should take 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





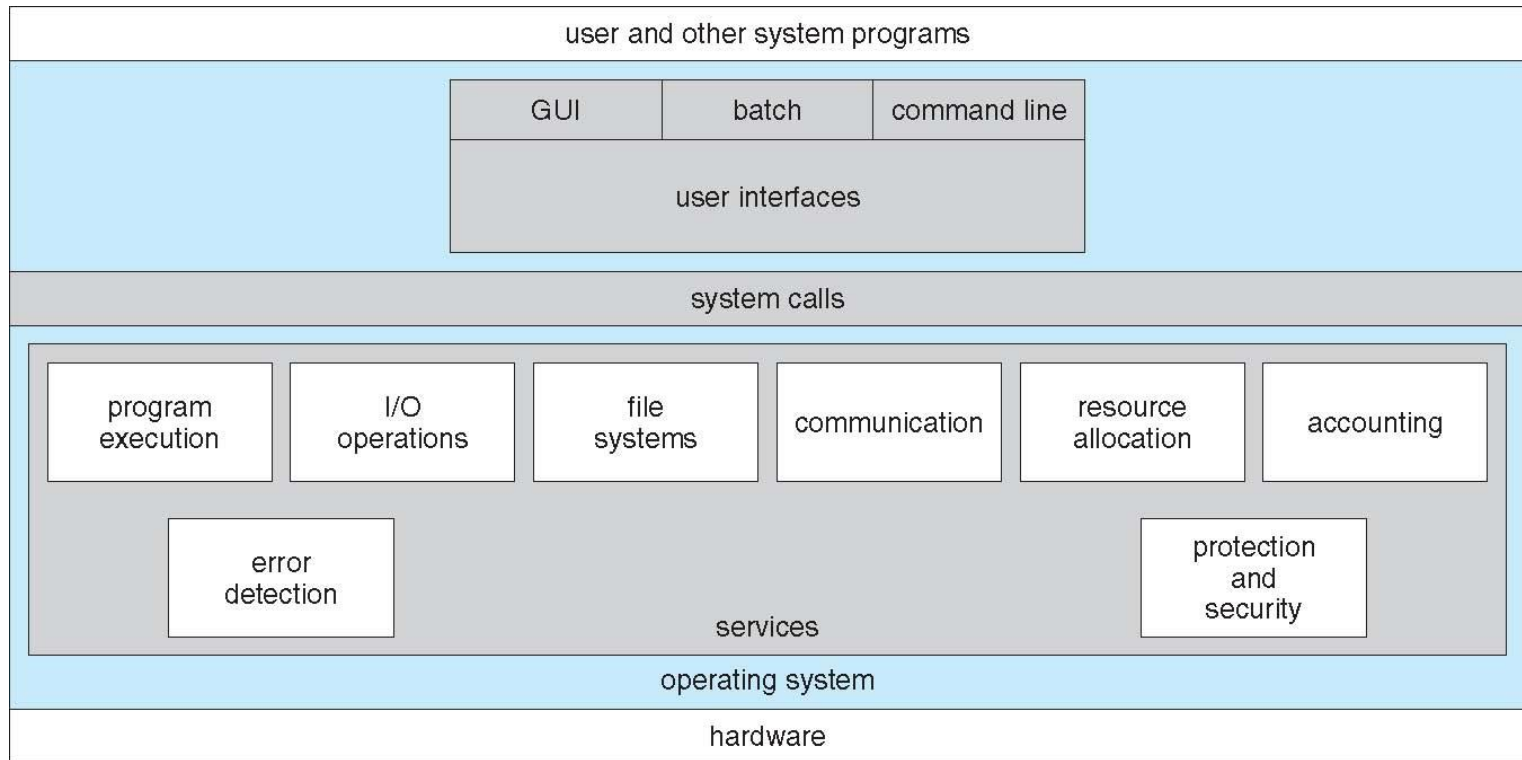
# Operating System Services (Cont.)

- Another set of OS functions exists for ensuring efficient operation of the system itself via resource sharing
  - **Resource allocation** - When multiple users and multiple jobs are running concurrently, resources must be allocated to each of them
    - ▶ 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** -
    - ▶ **Protection** involves ensuring that access to system resources is controlled
    - ▶ **Security** of the system 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

```

Default
New Info Close Execute Bookmarks

Default
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)

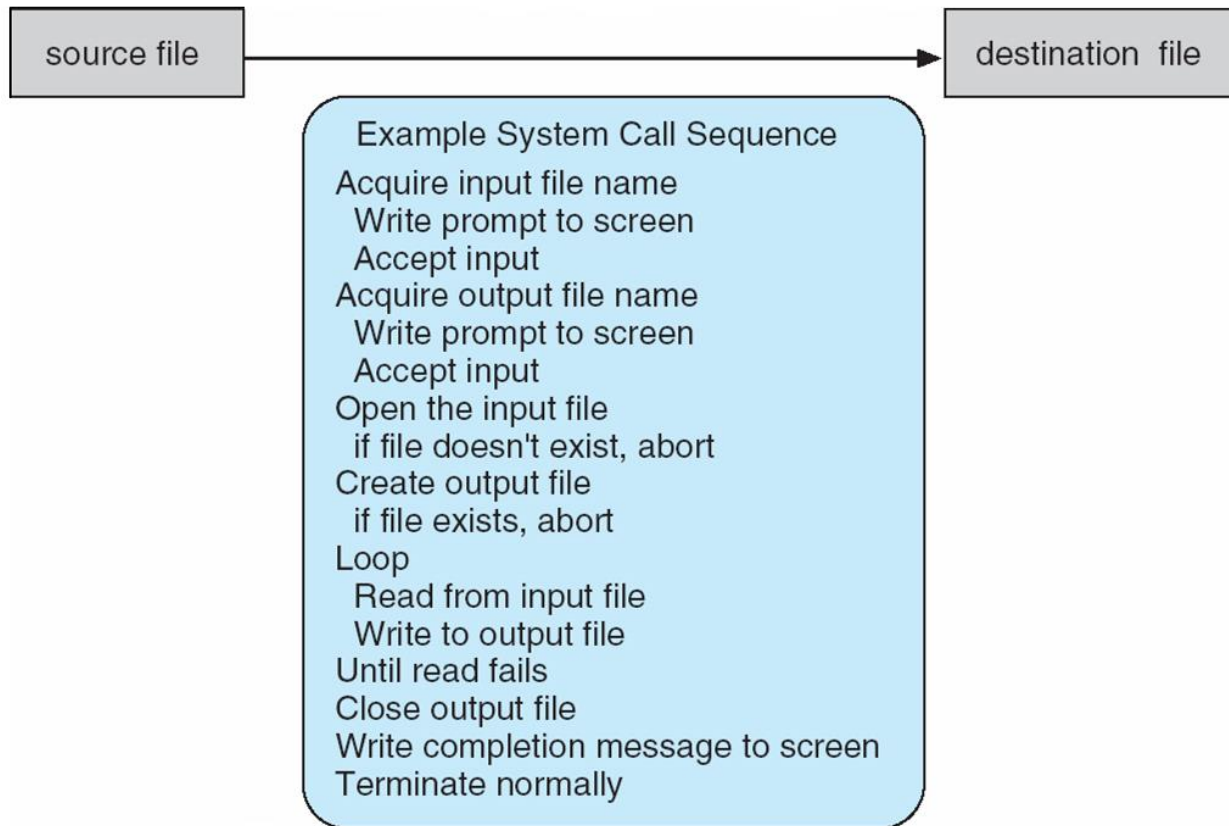






# Example of System Calls

- 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:

```
#include <unistd.h>

ssize_t  read(int fd, void *buf, size_t count)
```

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

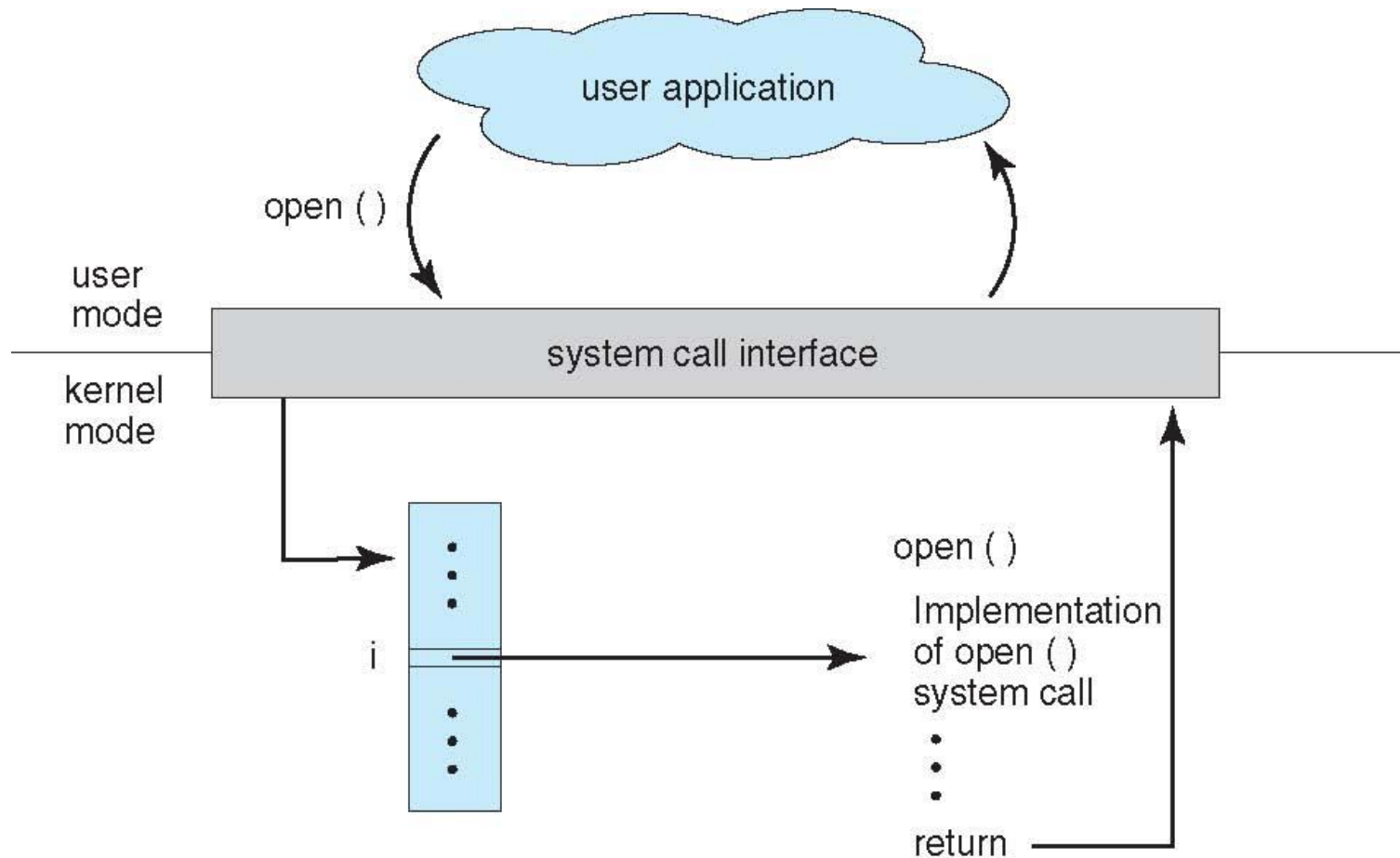
---

- Typically, a number 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
  - 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
  - **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

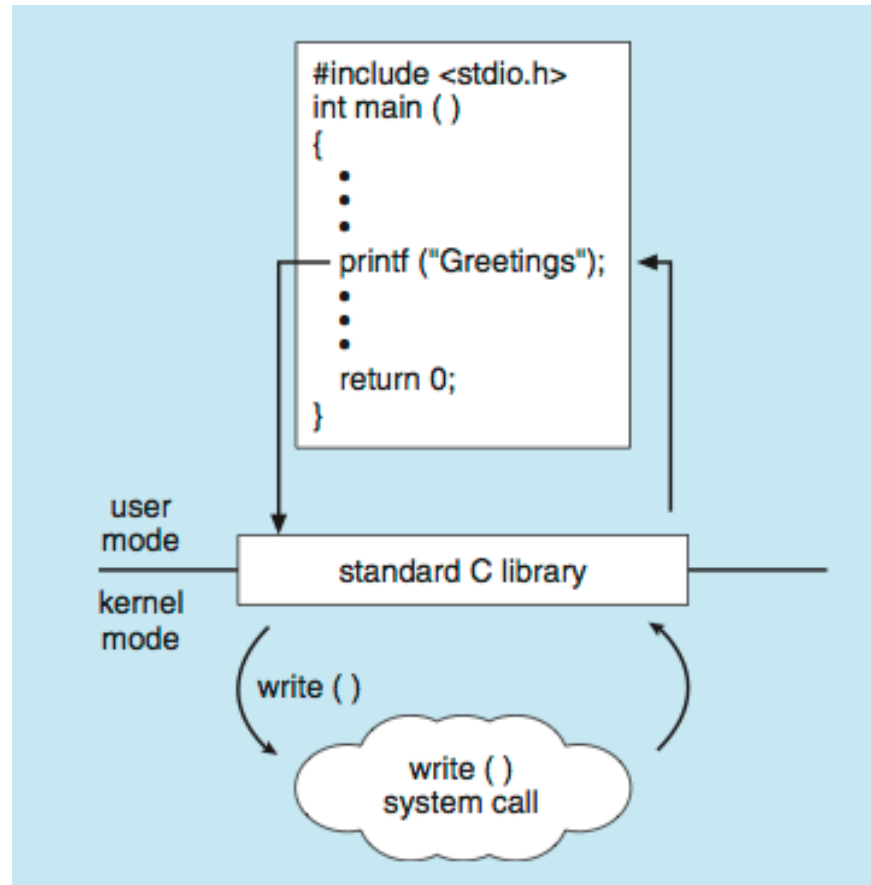
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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





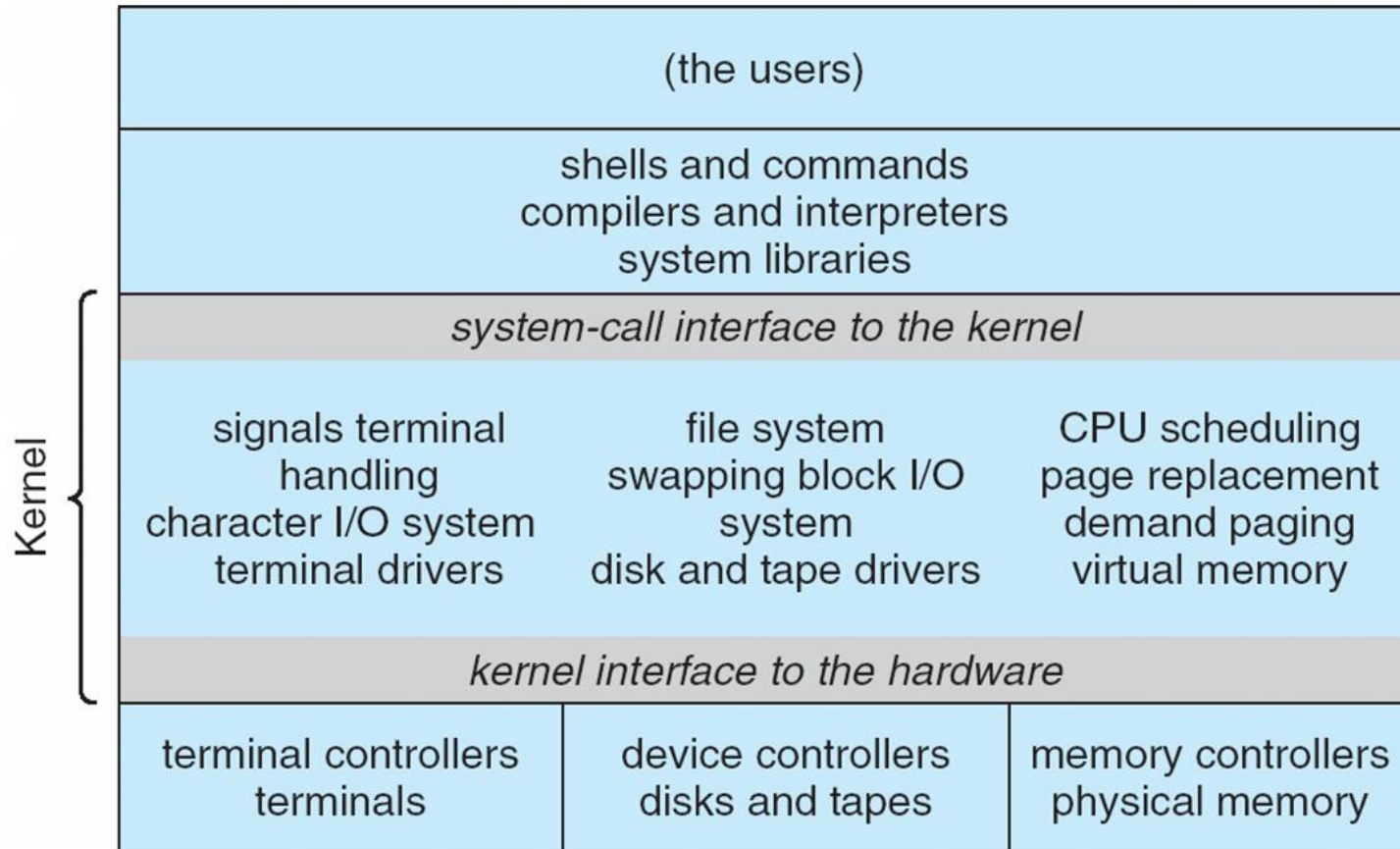
# Standard C Library Example

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



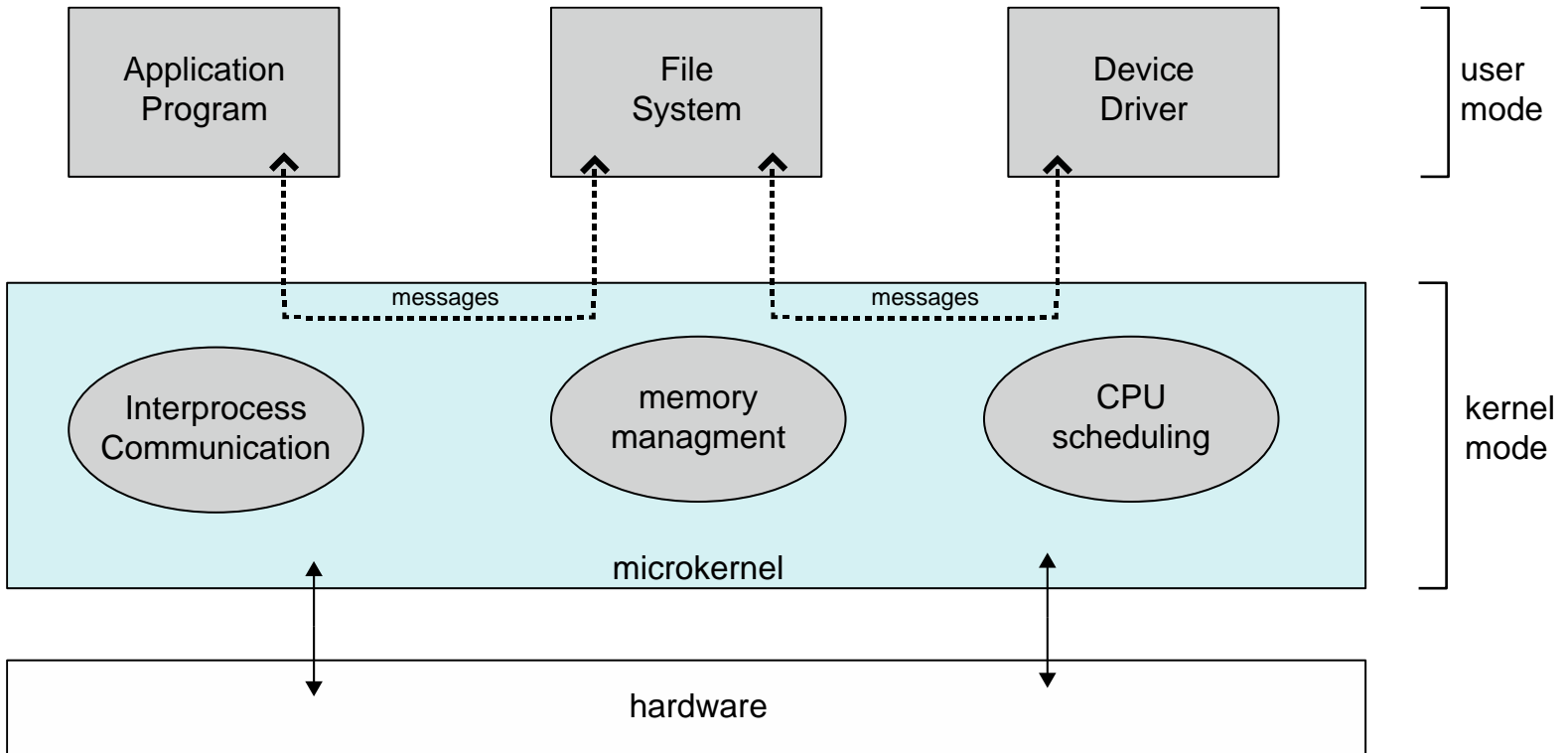


# Traditional UNIX System Structure





# Microkernel System Structure





# 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

---

