System Structures

Operating Systems (CS-220)
Fall 2020, FAST NUCES

COURSE SUPERVISOR: ANAUM HAMID

anaum.hamid@nu.edu.pk

Roadmap

- **Operating-System Services**
- **User Operating-System Interfaces**
- **System Calls**
- **Types of System Calls**
- **System Programs**

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 (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** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and storage devices.

Operating System Services

- **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.

In addition to raw data storage, the OS is also responsible for maintaining directory and subdirectory structures, mapping file names to specific blocks of data storage, and providing tools for navigating and utilizing the file system.

- **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).
  - Inter-process communications, IPC
- **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

Operating System Services

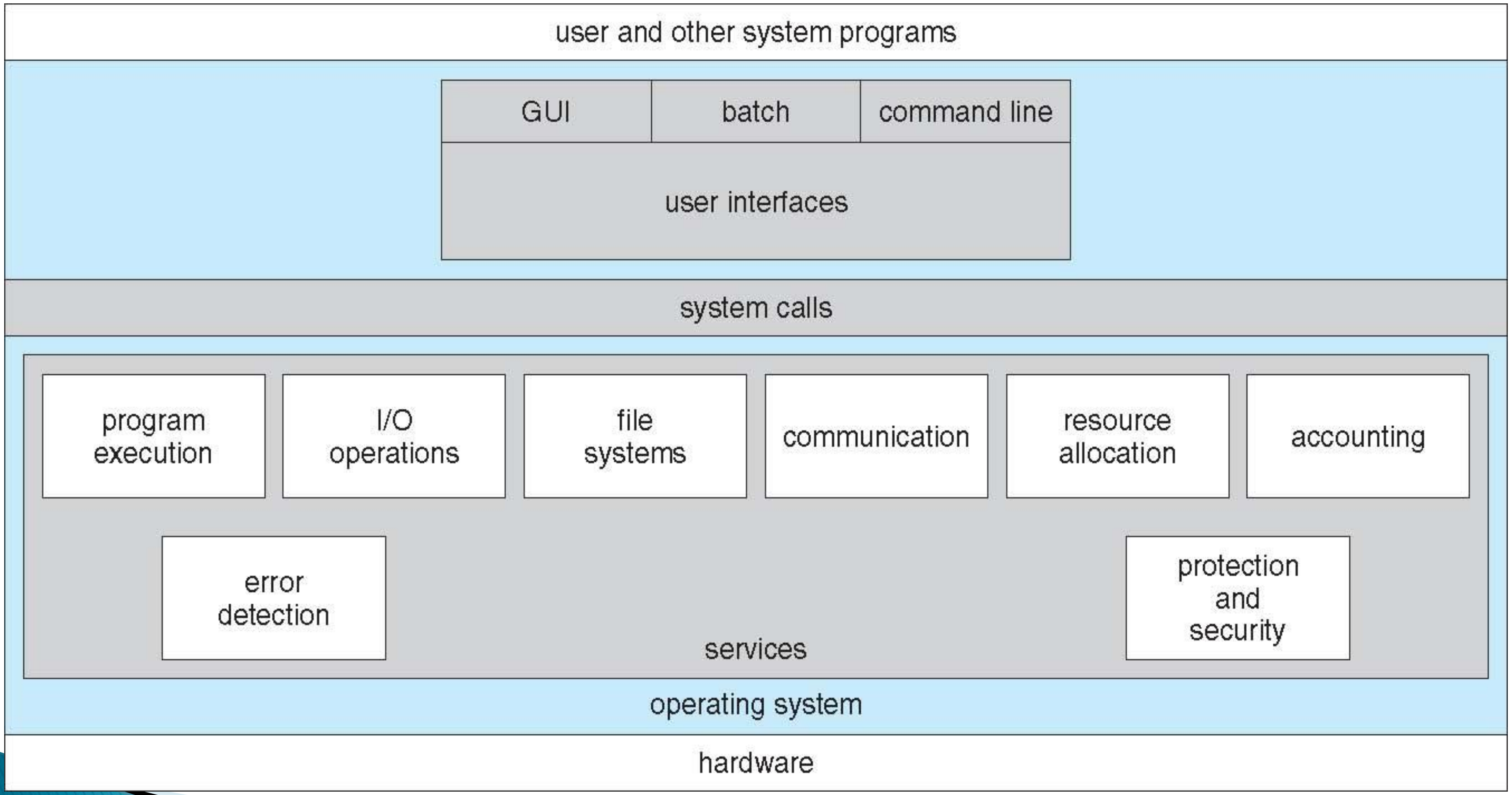
- **Resource allocation** – When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - E.g. CPU cycles, main memory, storage space, and peripheral devices. Some resources are managed with generic systems and others with very carefully designed and specially tuned systems, customized for a particular resource and operating environment.

- **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

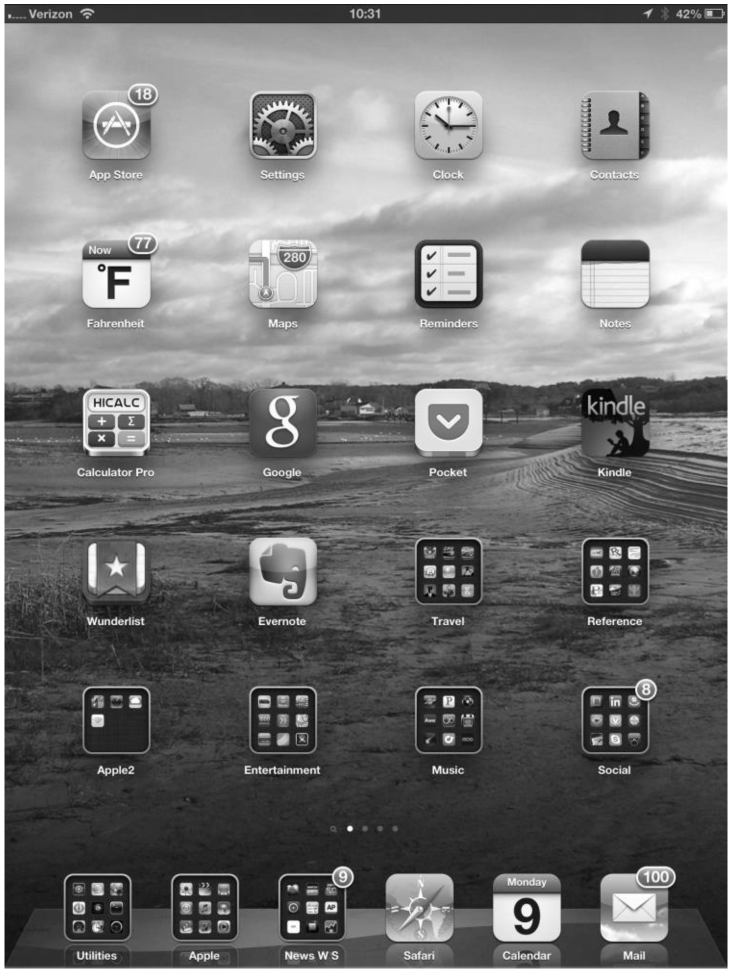
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs

User Operating System Interface – GUI

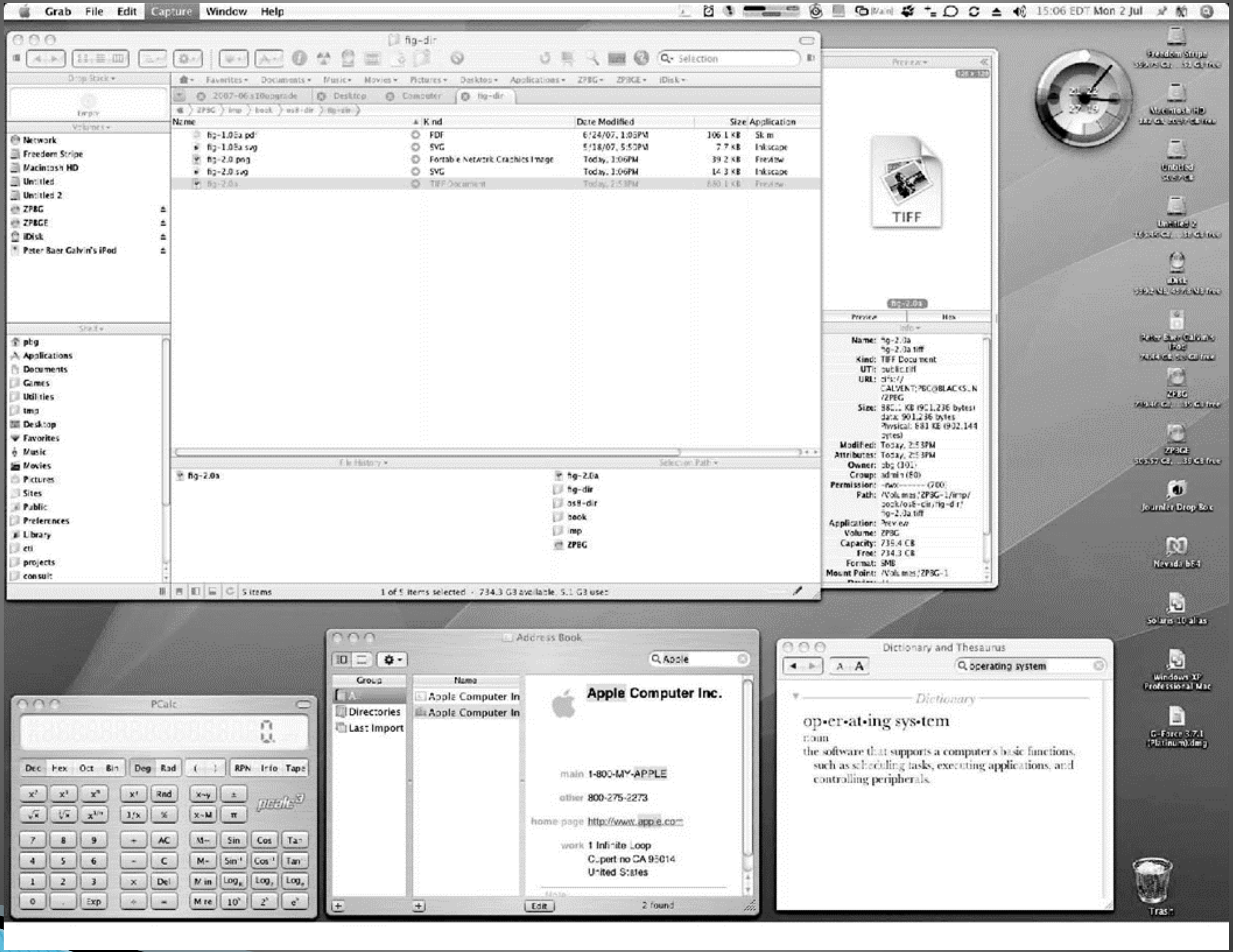
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
- Many systems now include both CLI and GUI interfaces
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)
  - Common Desktop Environment (**CDE**) – Unix based
  - K Desktop Environment (KDE) – Linux based
  - GNOME Unix variant

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



Name	Kind	Date Modified	Size	Application
fig-1.05a pdf	PDF	6/24/07, 1:05PM	106.1 KB	Skim
fig-1.05a svg	SVG	5/18/07, 5:53PM	7.7 KB	Inkscape
fig-2.0 png	Portable Network Graphics Image	Today, 3:06PM	39.2 KB	Preview
fig-2.0 svg	SVG	Today, 3:06PM	14.3 KB	Inkscape
fig-2.0a	TIFF Document	Today, 2:53PM	880.1 KB	Preview

Name:	fig-2.0a fig-2.0a tiff
Kind:	TIFF Document
UTI:	public.tiff
URL:	cifs://CALVENT;PBC@BLACKS_N/ZPBG
Size:	880.1 KB (901,236 bytes) data: 901,236 bytes Physical: 881 KB (902,144 bytes)
Modified:	Today, 2:53PM
Attributes:	Today, 2:53PM
Owner:	pbg (501)
Group:	admin (80)
Permission:	-rwx------ (700)
Path:	/Volumes/ZPBG-1/tmp/book/osx-dir/fig-dir/fig-2.0a tiff
Application:	Preview
Volume:	ZPBG
Capacity:	735.4 GB
Free:	734.3 GB
Format:	SMB
Mount Point:	/Volumes/ZPBG-1

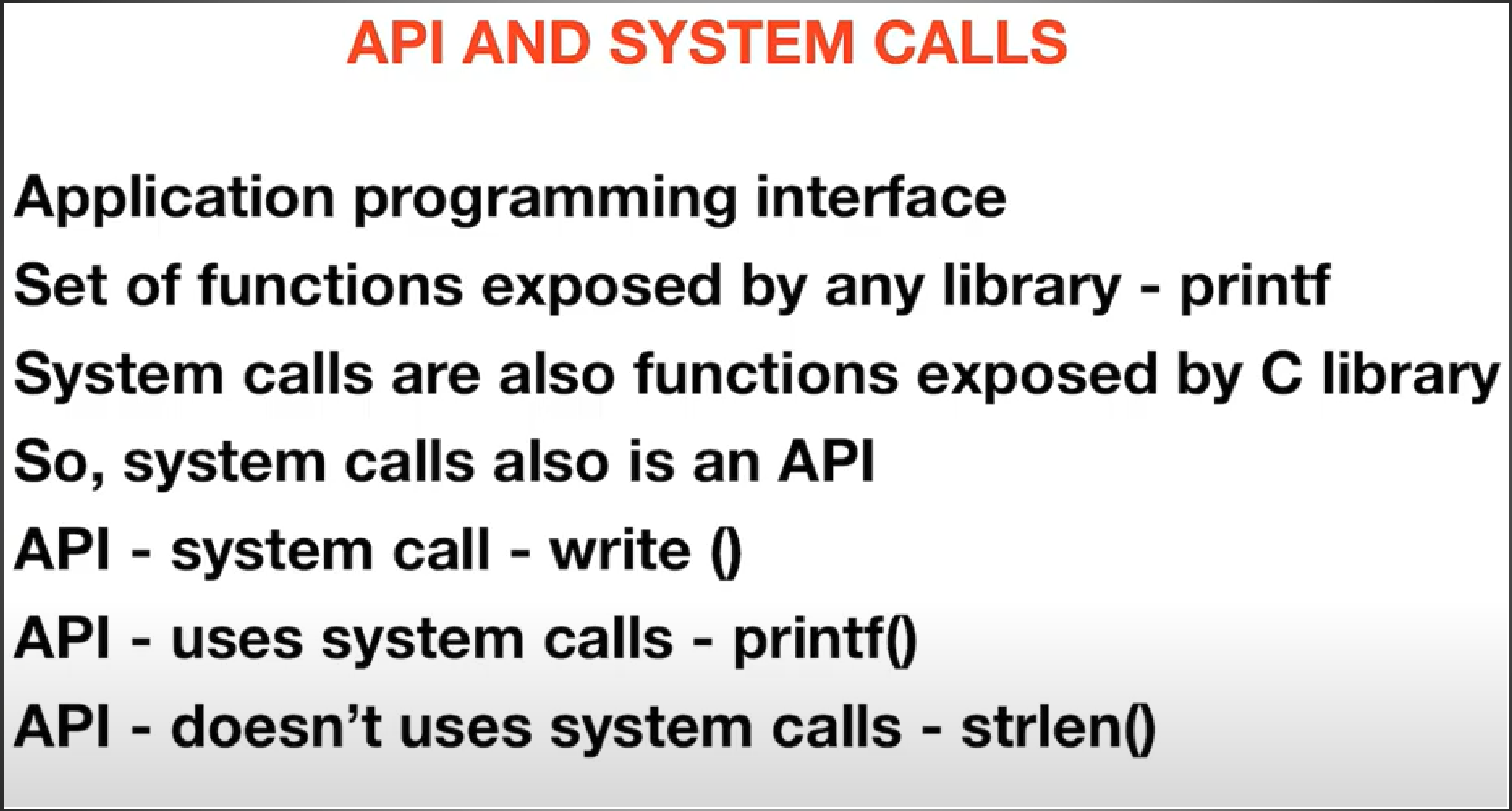
Apple Computer Inc.

main	1-800-MY-APPLE
other	800-275-2273
home page	http://www.apple.com
work	1 Infinite Loop Cupertino CA 95014 United States

op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

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)



API AND SYSTEM CALLS

Application programming interface

Set of functions exposed by any library - printf

System calls are also functions exposed by C library

So, system calls also is an API

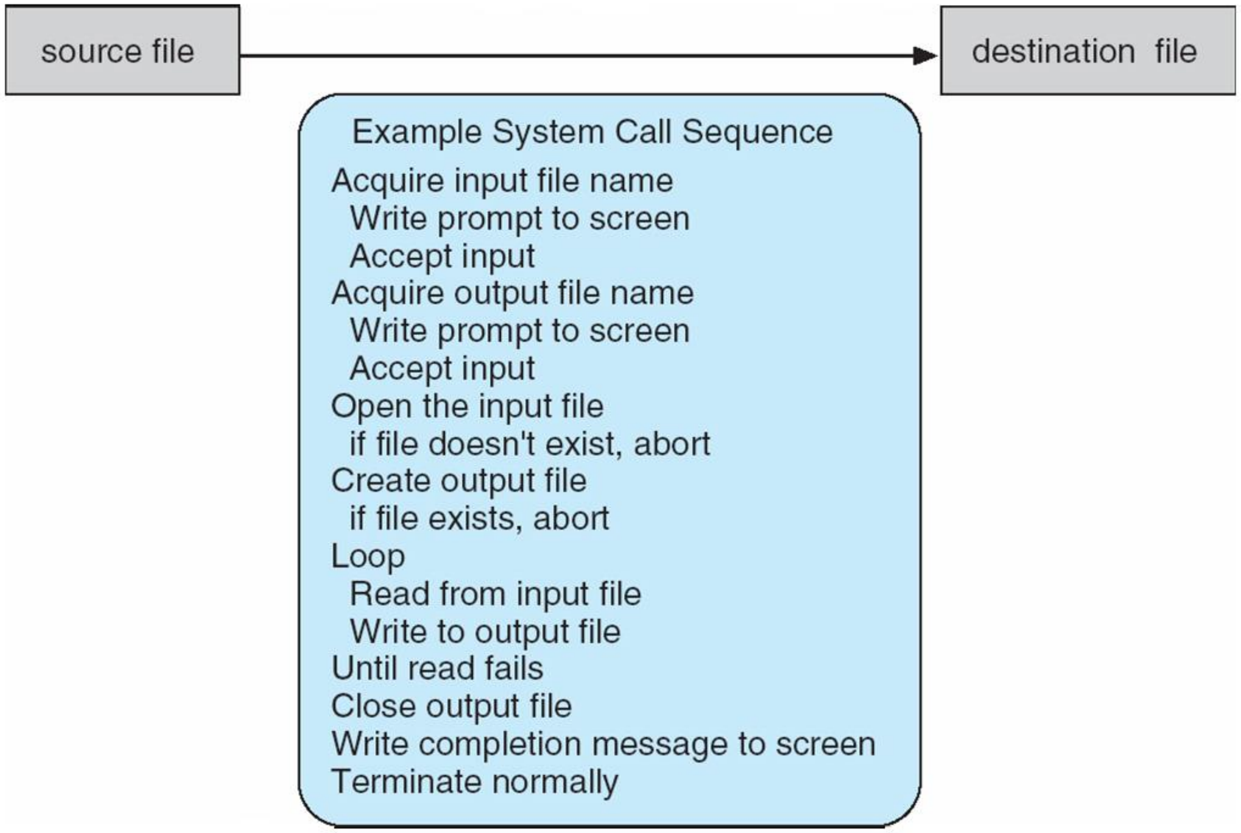
API - system call - write ()

API - uses system calls - printf()

API - doesn’t uses system calls - strlen()

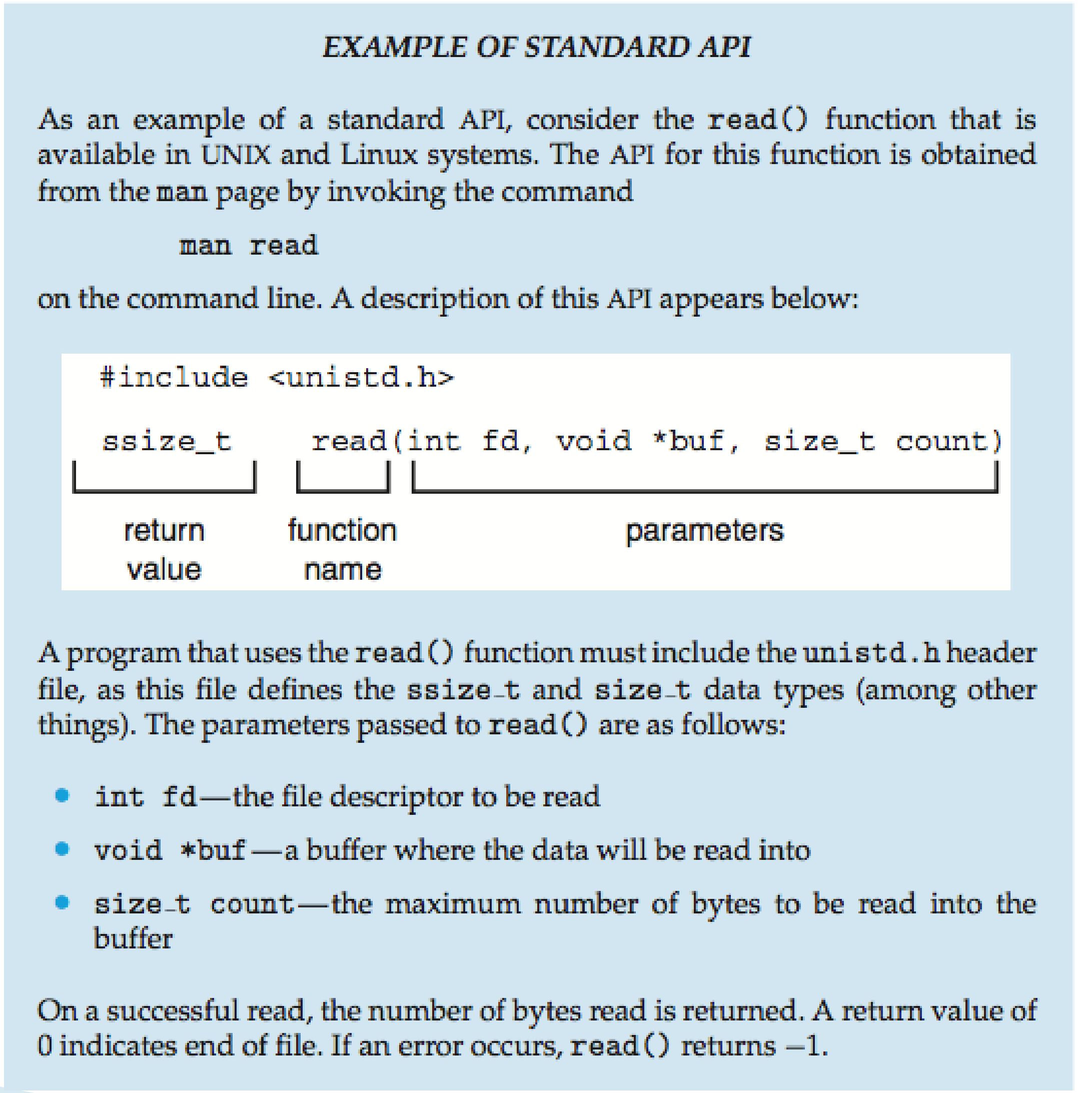
Example of System Calls

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



```
Example System Call Sequence
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

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:

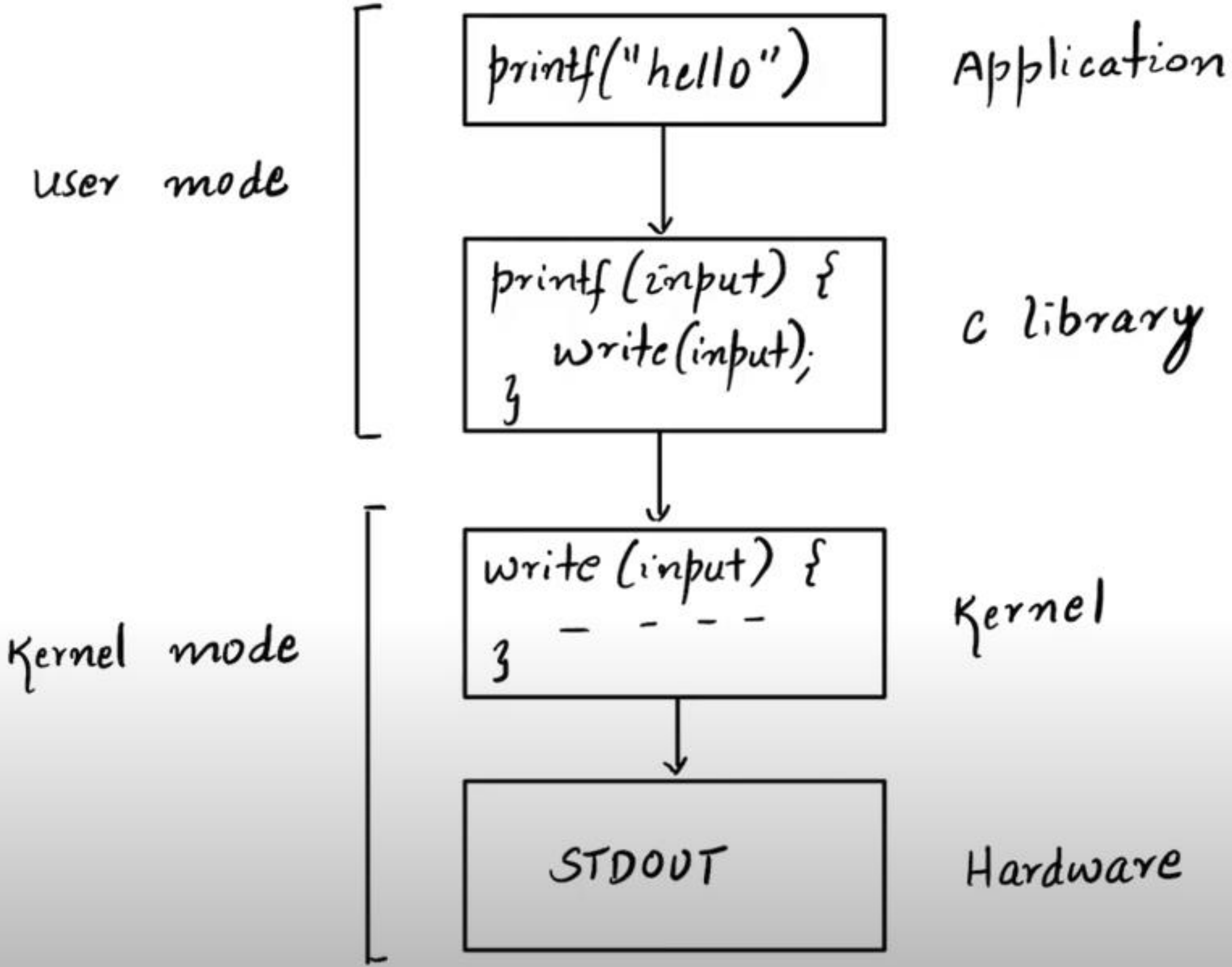
```
#include <unistd.h>
```

$$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\quad\underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$$

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:

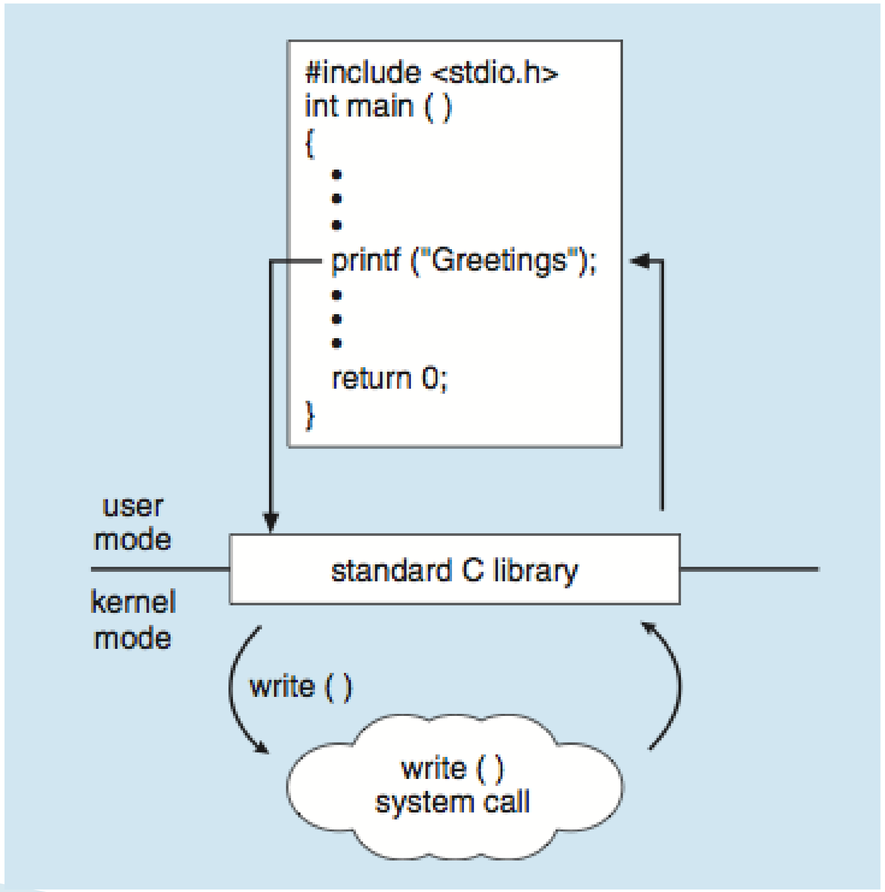
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

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.


user mode
Kernel mode
printf("hello")
Application
printf (input) {
write(input);
}
C library
write (input) {
- - - -
}
Kernel
STDOUT
Hardware

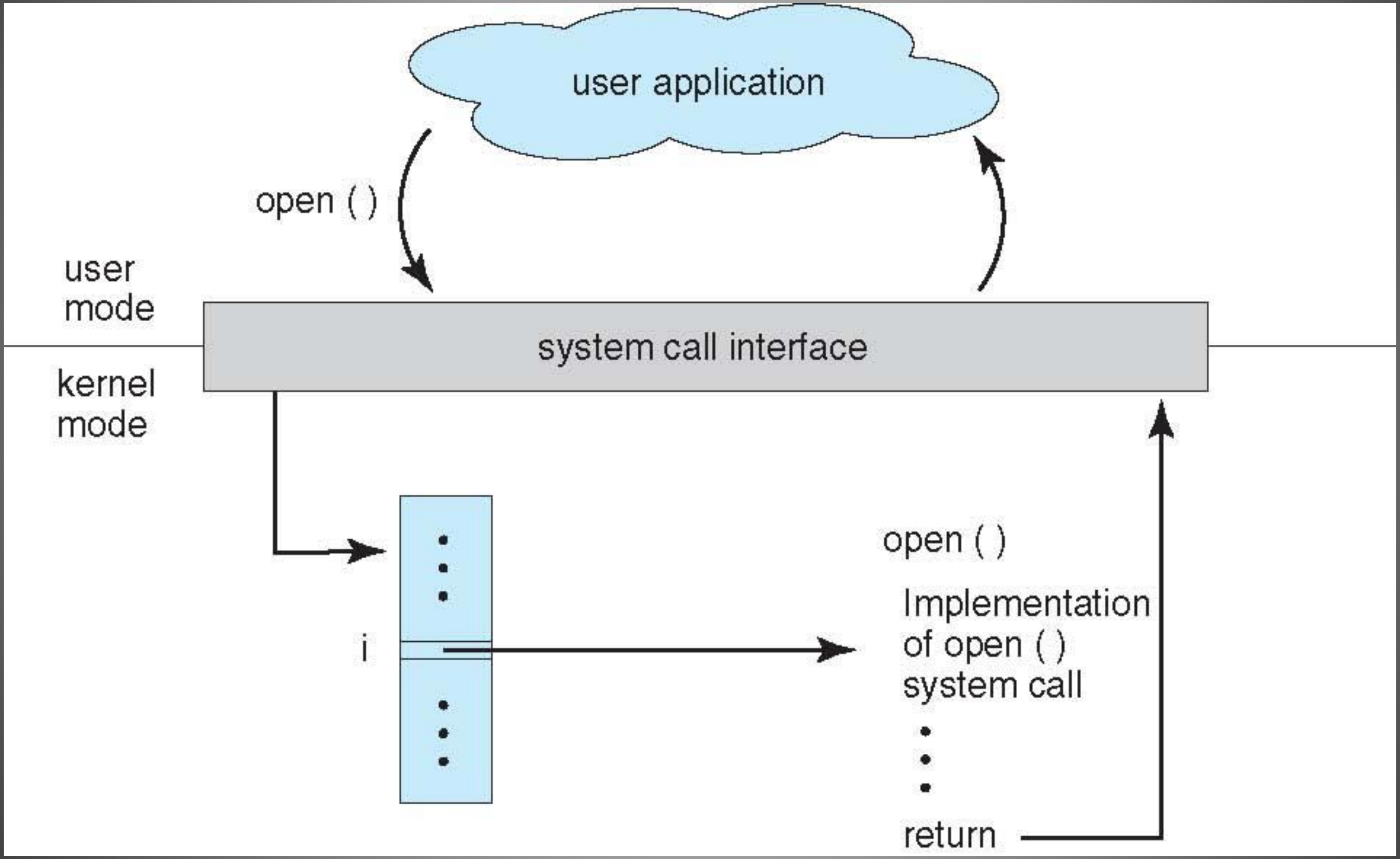
Standard C Library Example

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



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

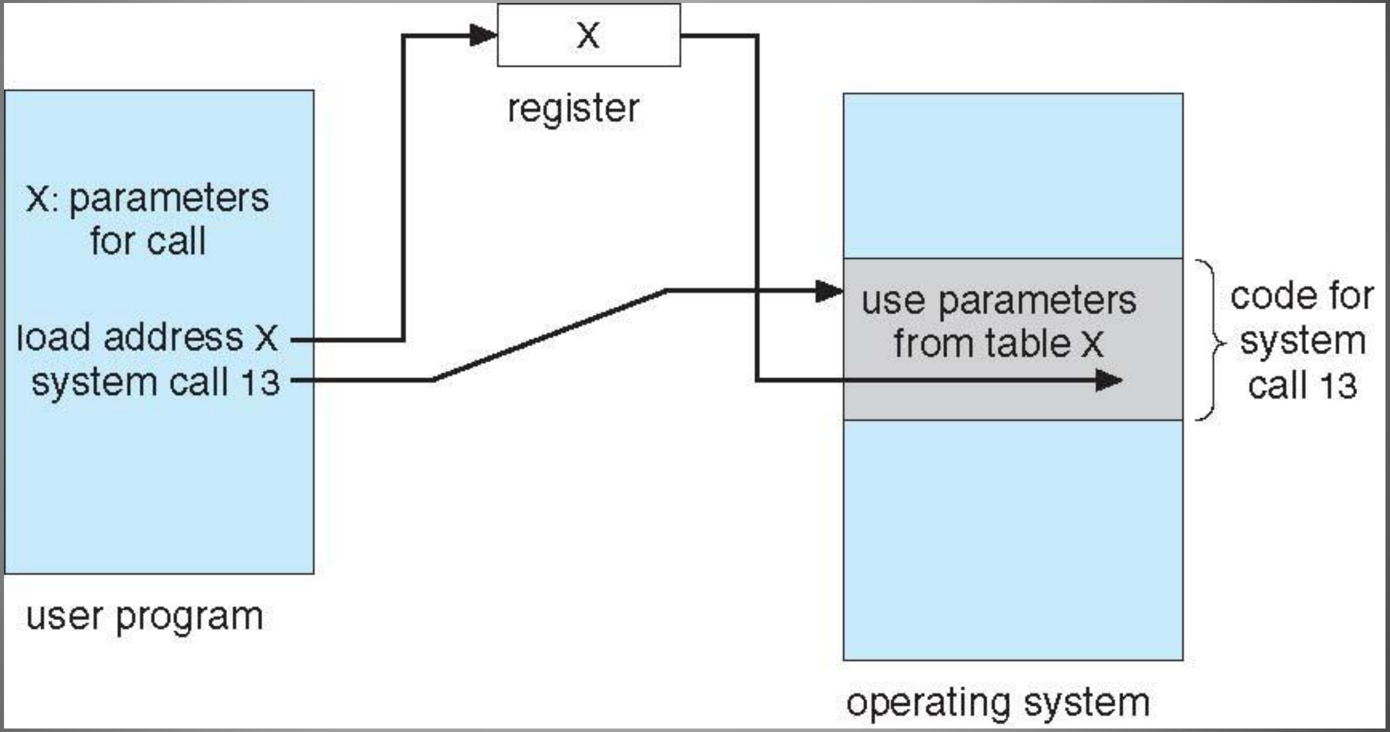
API – System Call – OS Relationship



System Call Parameter Passing

- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
  - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system

System Call Parameter Passing via Table



Types of System Calls

1.Process Control

- load
- execute
- create process (for example, fork on Unix-like systems or NtCreateProcess in the Windows NT Native API)
- terminate process
- get/set process attributes
- wait for time, wait event, signal event
- allocate free memory

Types of System Calls

System calls can be roughly grouped into five major categories:

1. Process Control
2. File management
3. Device Management
4. Information Maintenance
5. Communication
6. protection

Types of System Calls

2.File management

- create file, delete file
- open, close
- read, write, reposition
- get/set file attributes

3.Device Management

- request device, release device
- read, write, reposition
- get/set device attributes
- logically attach or detach devices

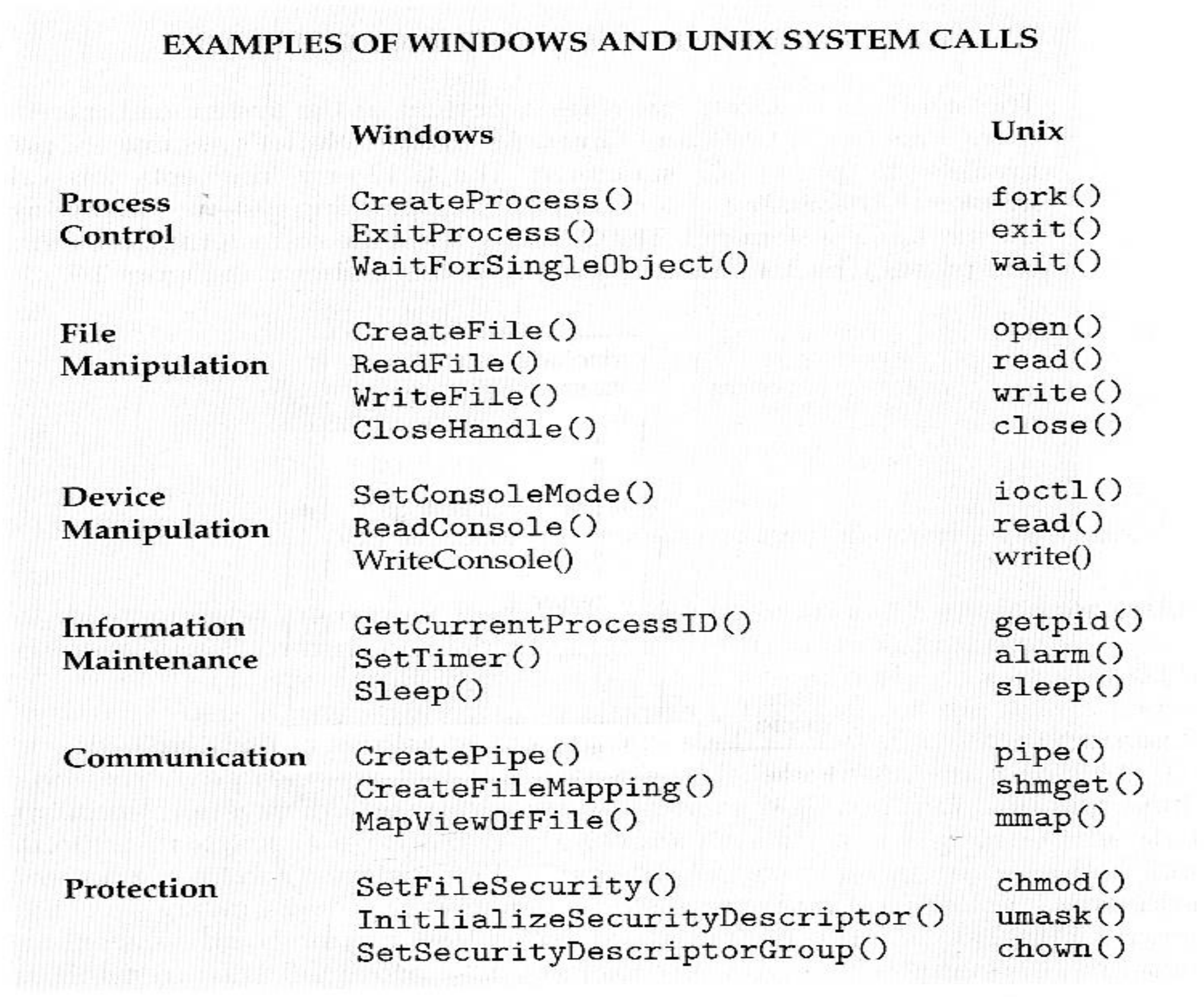
Types of System Calls

4.Information Maintenance

- get/set time or date
- get/set system data
- get/set process, file, or device attributes

5.Communication

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote device

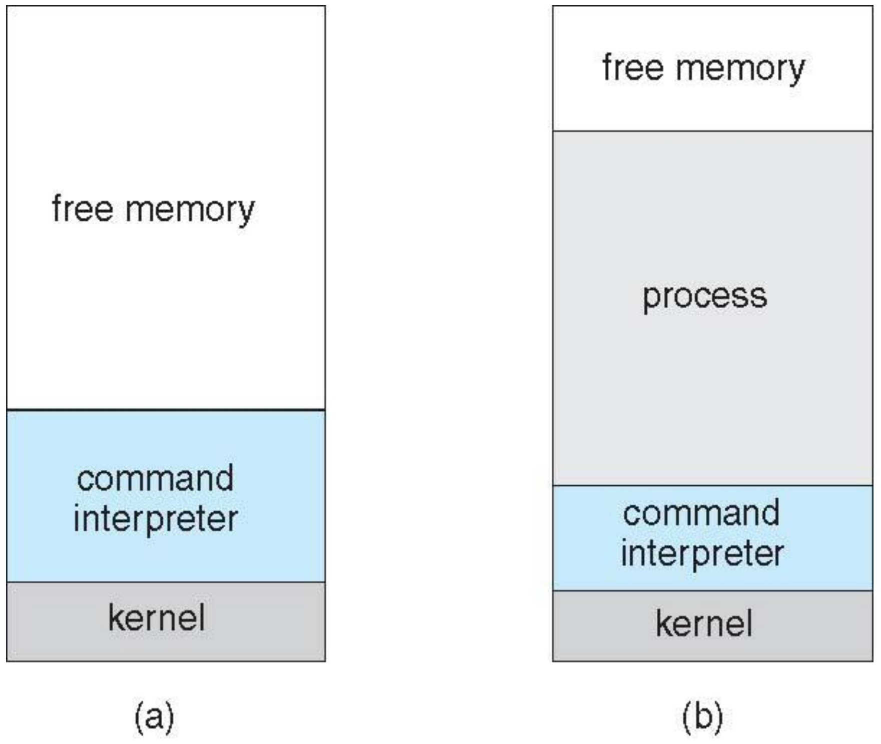


EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

Example: MS-DOS

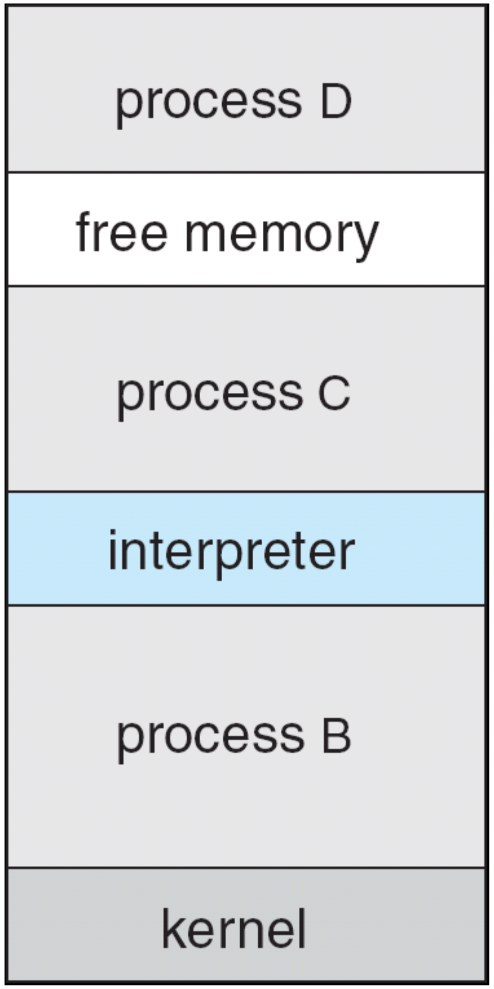
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup running a program

Example: FreeBSD (Berkeley Software Distribution)

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

- System programs provide a convenient environment for program development and execution. They can be divided into:
  1. File manipulation
  2. Status information
  3. File modification
  4. Programming language support
  5. Program loading and execution
  6. Communications
  7. Background services
  8. Application programs

System Programs

- **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

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**– debugging systems for higher-level and machine language
- **Communications** – Provide the mechanism for creating virtual connections among processes, users, and computer systems

System Programs (Cont.)

- **Background Services**
  - Launch at boot time
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as services, subsystems, daemons
- **Application programs**
  - Run by users