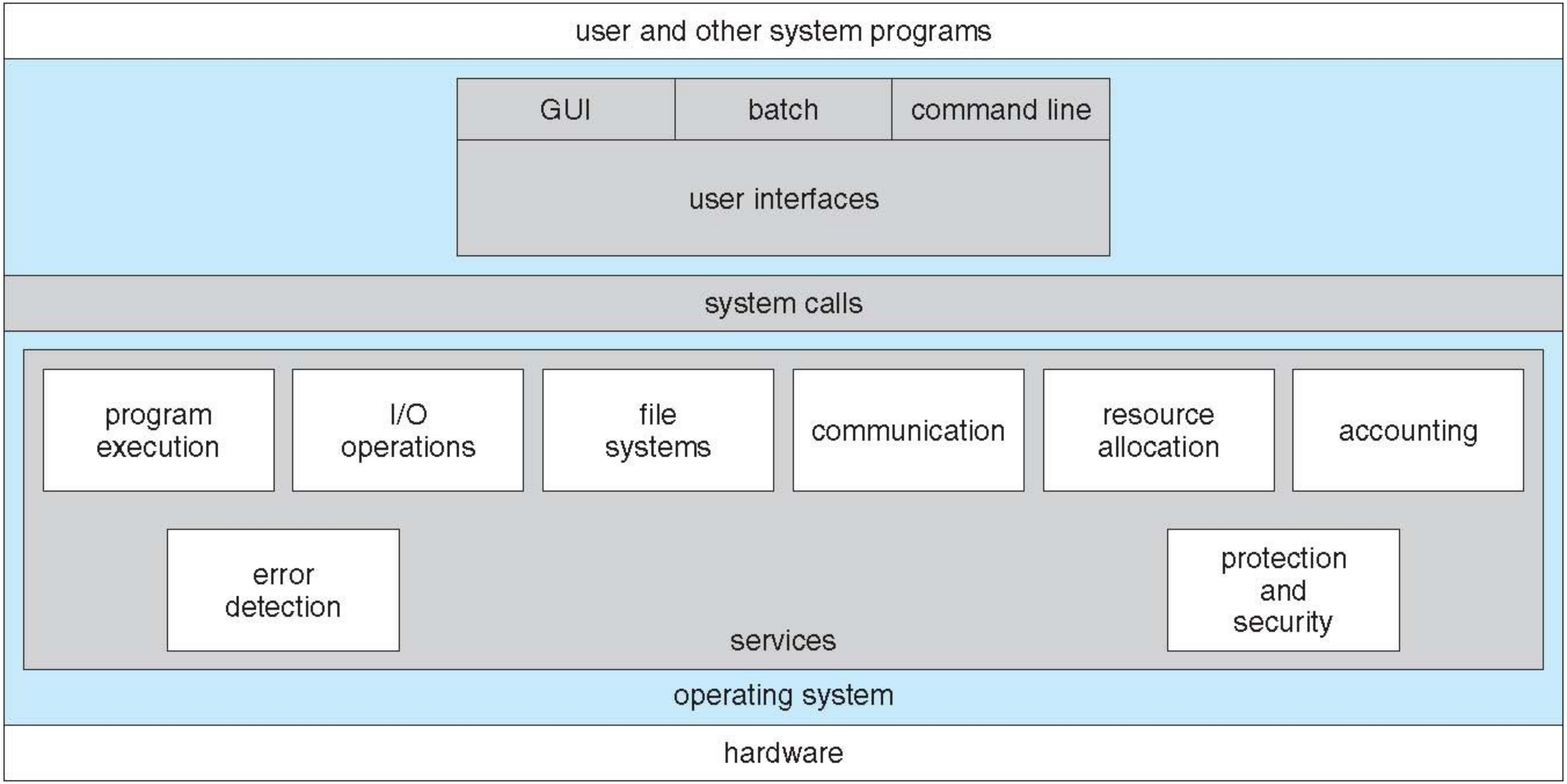
Operating System

Ch02: Operating System Structures

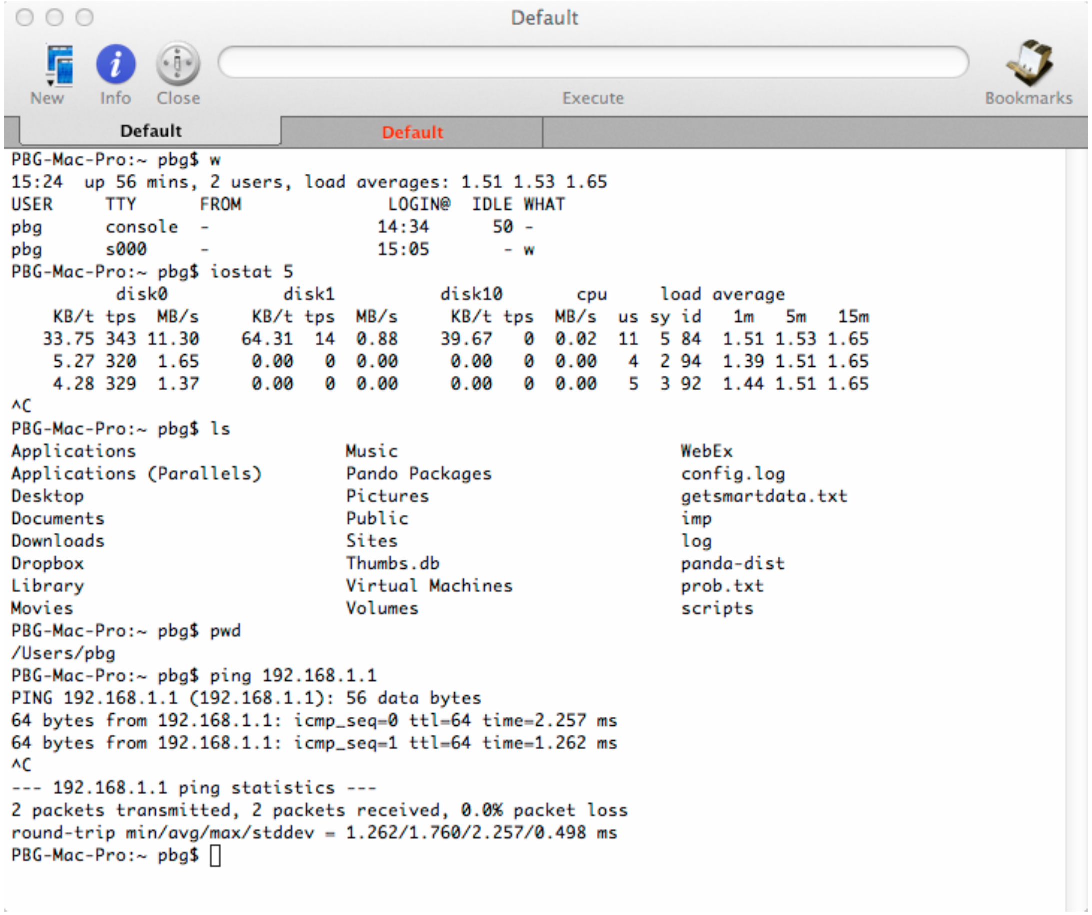
BeomSeok Kim
Department of Computer Engineering
KyungHee University
passion0822@khu.ac.kr

Operating System Services



Operating System Services

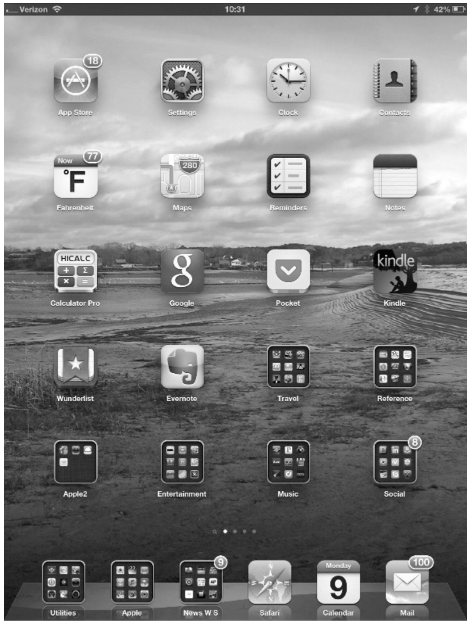
- User interface service
  - ✓ Command-Line Interpreter (CLI)
  - ✓ Bourne shell, bash, etc.

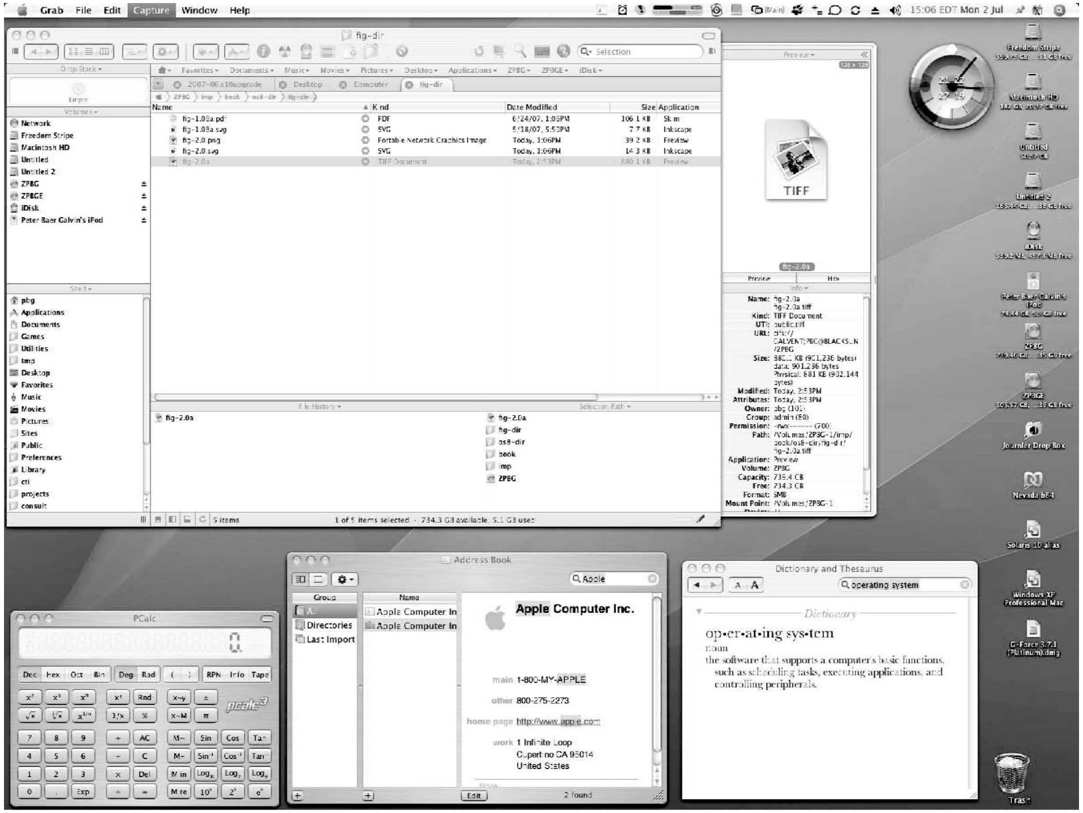


```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0              disk1             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$ 
```

Operating System Services

- User interface service
  - ✓ Graphical User Interface (GUI)
    - ➢ E.g.) Mac OS X
  - ✓ Touch screen interface
    - ➢ E.g.) iPhone





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.

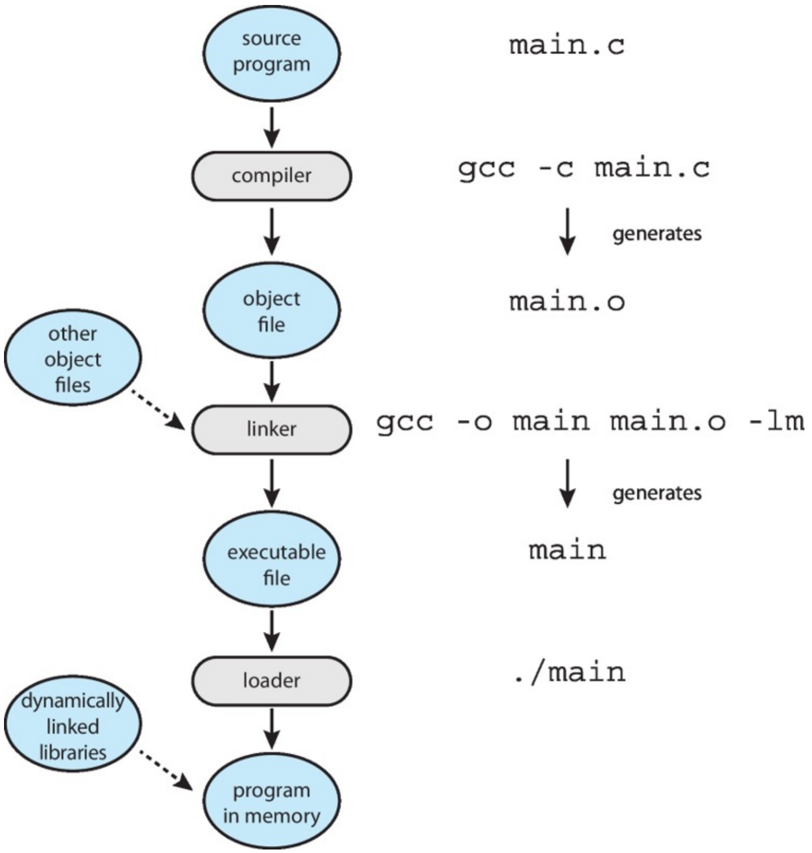
Operating System Services

- System programs
  - ✓ Provide a convenient environment for program development and execution
  - ✓ File manipulation
  - ✓ Status information sometimes stored in a file modification
  - ✓ Programming language support
  - ✓ Program loading and execution
    - ➢ Linker and loader
  - ✓ Communications
  - ✓ Background services
  - ✓ Cf) Application programs

Operating System Services

- Linkers and loaders
  - ✓ Static vs. Dynamic linking

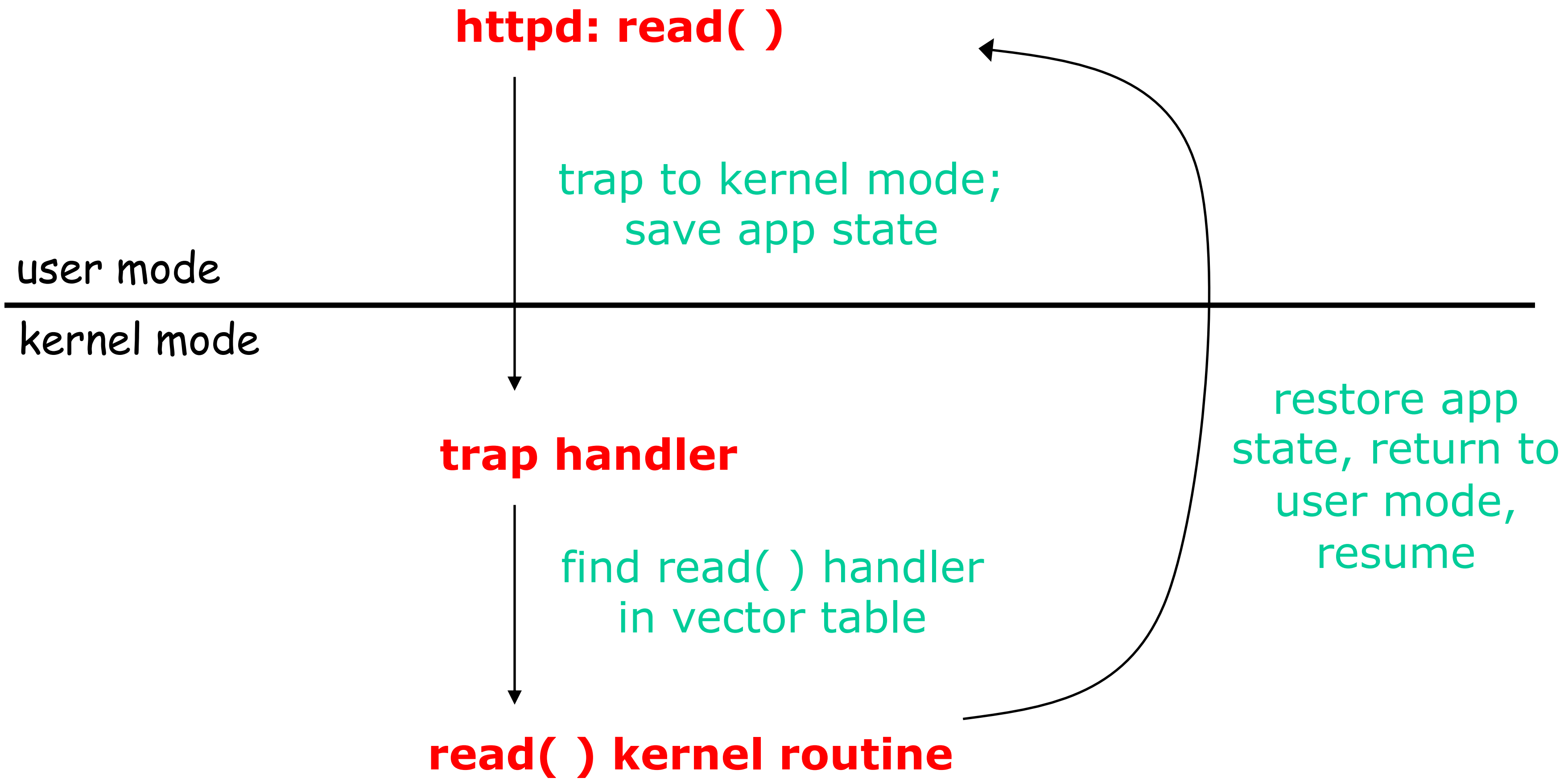
  - ✓ .dll (Dynamically Linked Library) in Windows
  - ✓ .sa & .so (shared library) in Linux



```
main.c
gcc -c main.c
main.o
gcc -o main main.o -lm
main
./main
```

Operating System Services

- System call service
  - ✓ Cf) Function call



httpd: read( )

trap to kernel mode;
save app state

user mode

kernel mode

trap handler

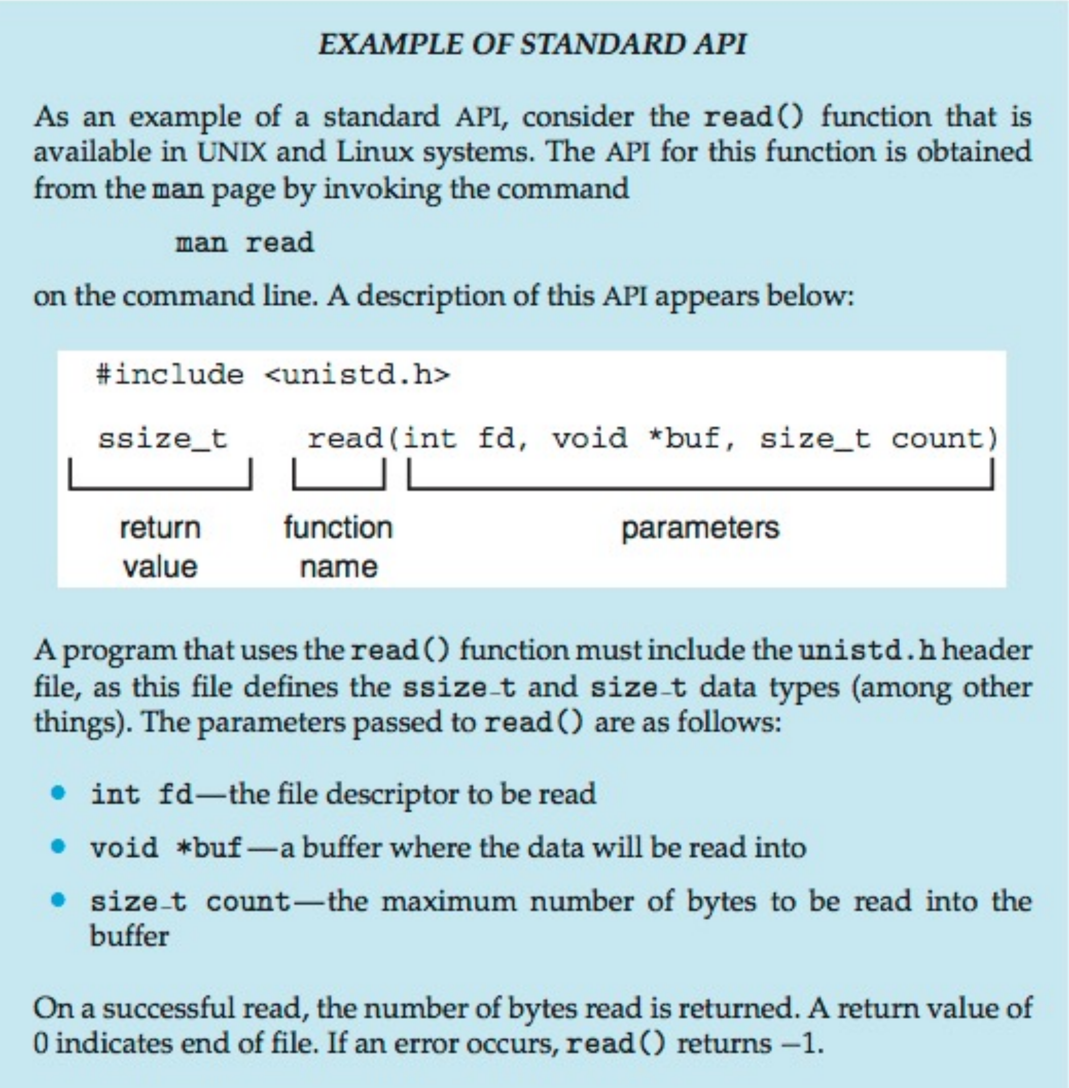
find read( ) handler
in vector table

read( ) kernel routine

restore app
state, return to
user mode,
resume

Operating System Services

- System call service
  - ✓ 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

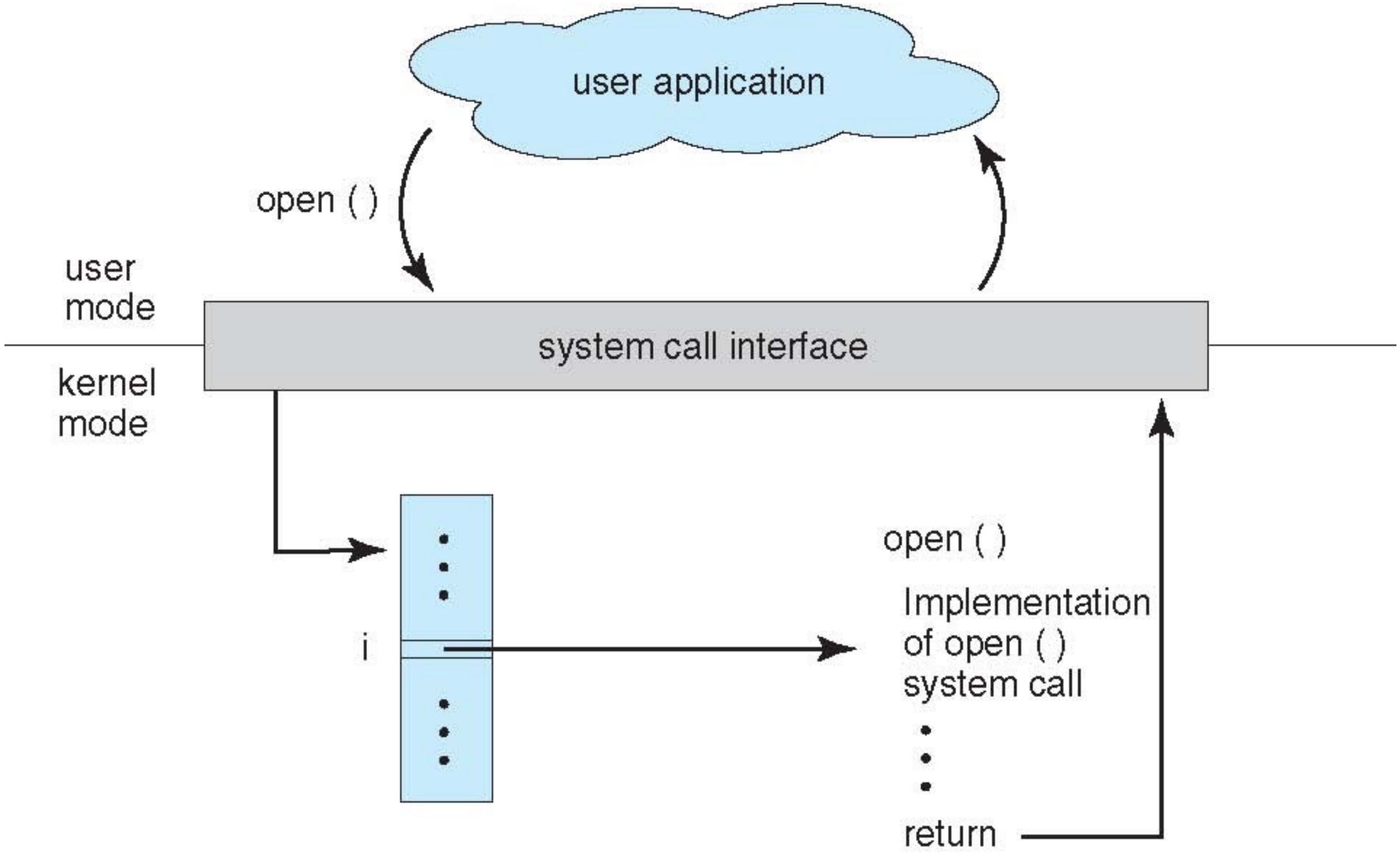
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `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.

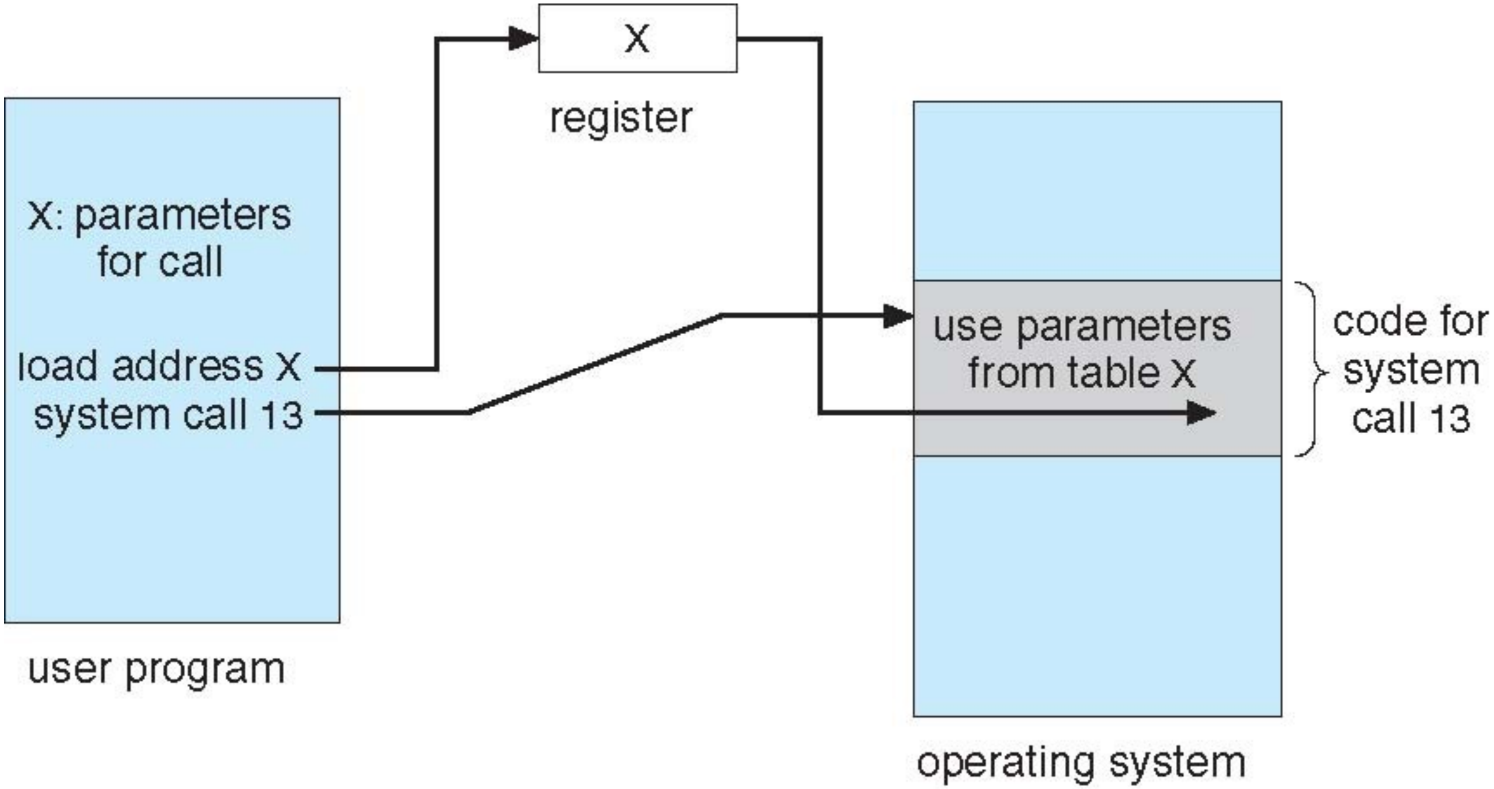
Operating System Services

- System call service
  - ✓ Handling in OS



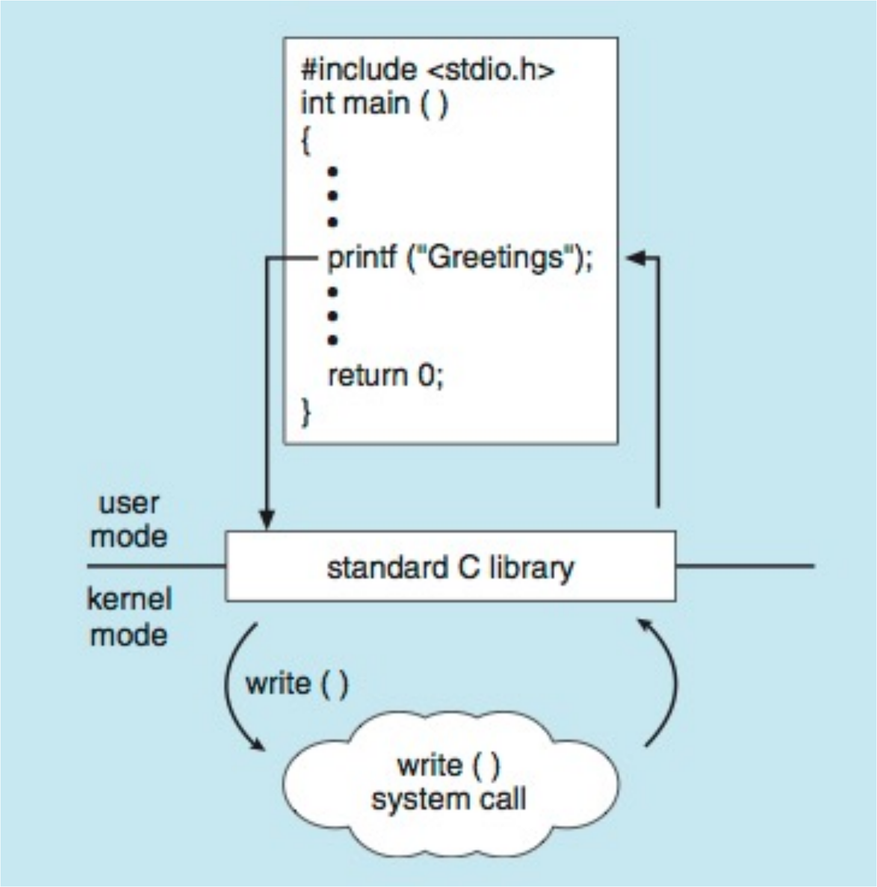
Operating System Services

- System call service
  - ✓ Parameter passing



Operating System Services

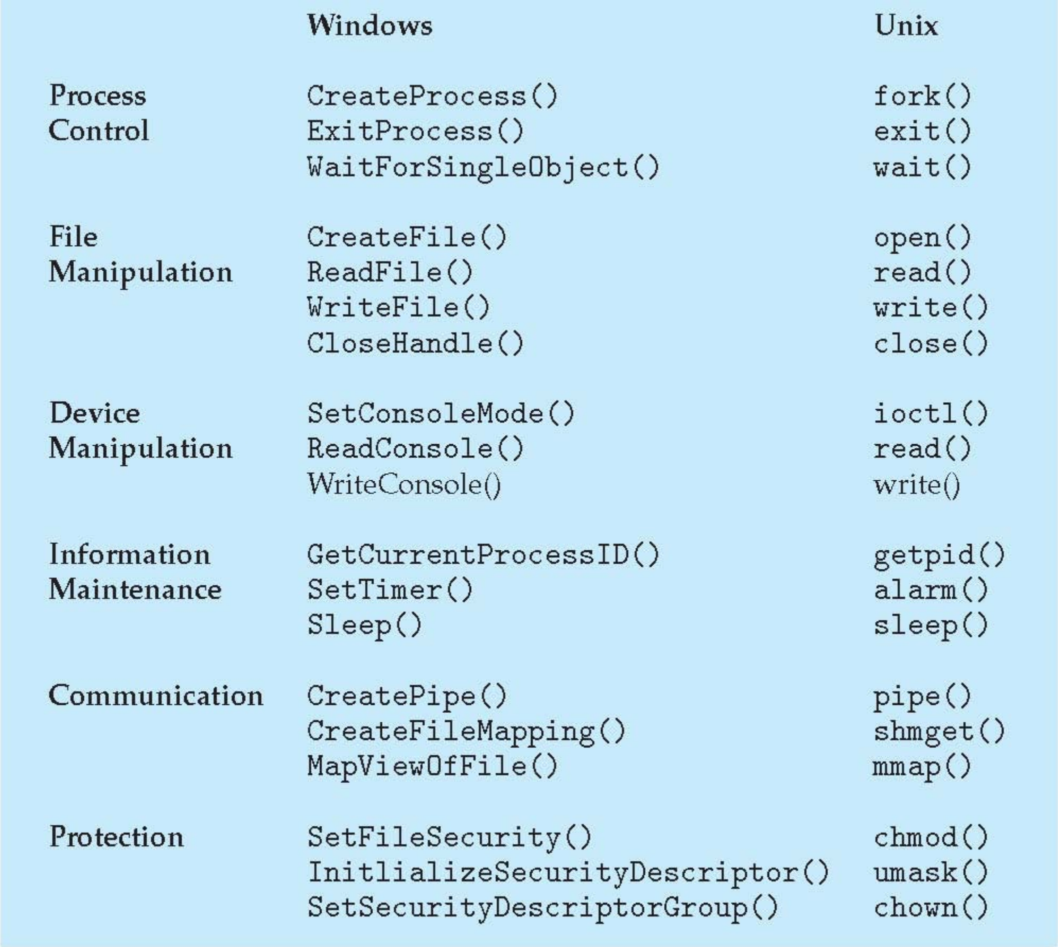
- System call service
  - ✓ Standard C library example



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

Operating System Services

- System call service
  - ✓ 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()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

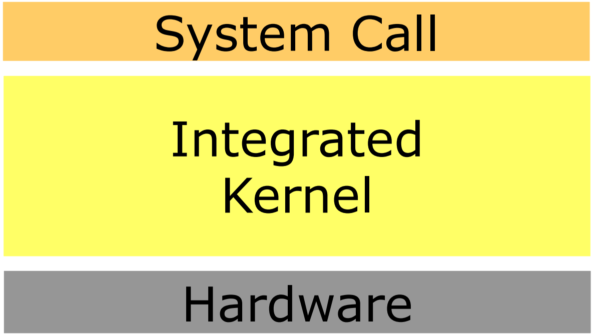
Operating System Services

- System call service

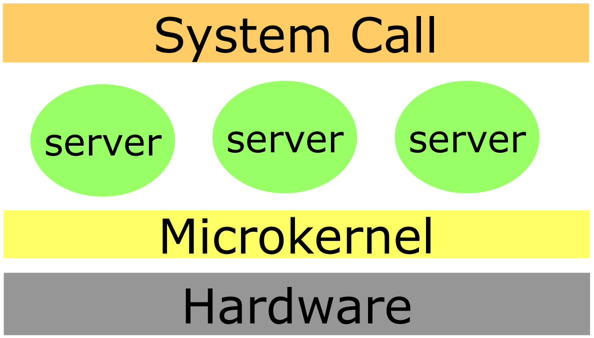
Process Management	fork	CreateProcess	Create a new process
	waitpid	WaitForSingleObject	Wait for a process to exit
	execve	(none)	CreateProcess = fork + execve
	exit	ExitProcess	Terminate execution
	kill	(none)	Send a signal
File Management	open	CreateFile	Create a file or open an existing file
	close	CloseHandle	Close a file
	read	ReadFile	Read data from a file
	write	WriteFile	Write data to a file
	lseek	SetFilePointer	Move the file pointer
	stat	GetFileAttributesEx	Get various file attributes
	chmod	(none)	Change the file access permission
File System Management	mkdir	CreateDirectory	Create a new directory
	rmdir	RemoveDirectory	Remove an empty directory
	link	(none)	Make a link to a file
	unlink	DeleteFile	Destroy an existing file
	mount	(none)	Mount a file system
	umount	(none)	Unmount a file system
	chdir	SetCurrentDirectory	Change the curent working directory

Operating System Structure

- Monolithic kernel
  - ✓ Function calls
  - ✓ Unixware, Solaris, AIX, HP-UX, Linux, etc.

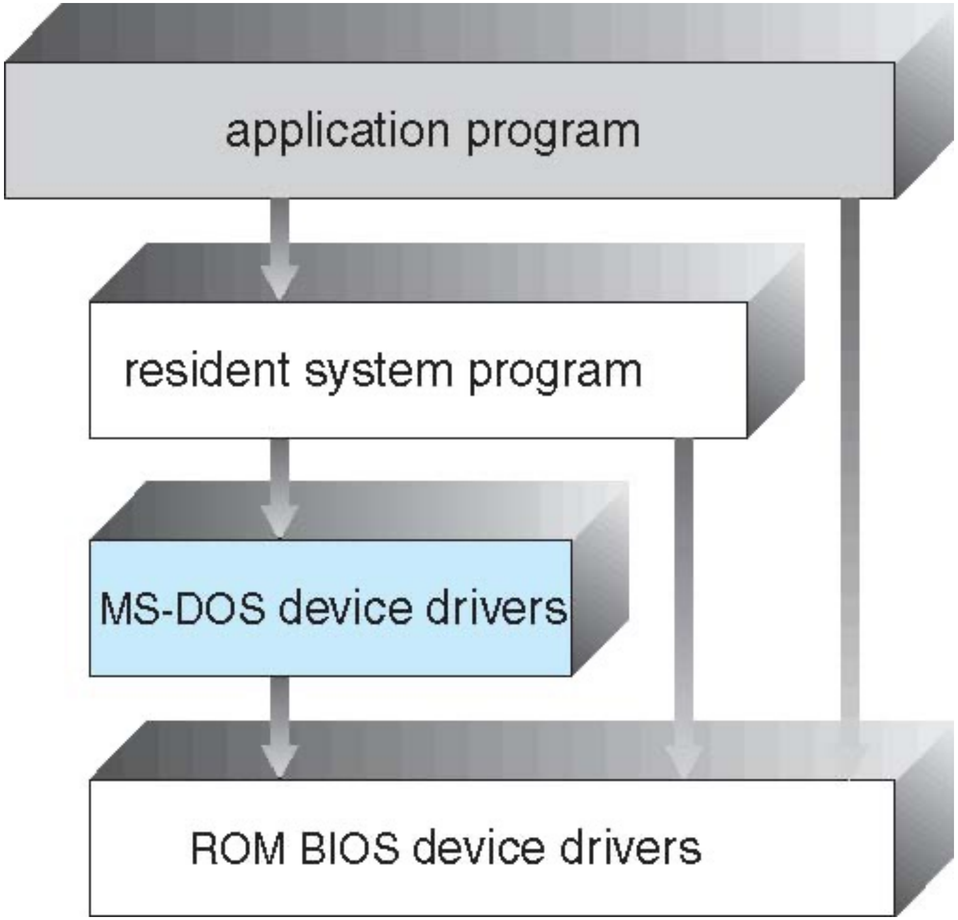


- Micro(μ) kernel
  - ✓ Multiple servers
  - ✓ Message passing
  - ✓ Mach, Chorus, Linux mk, etc.



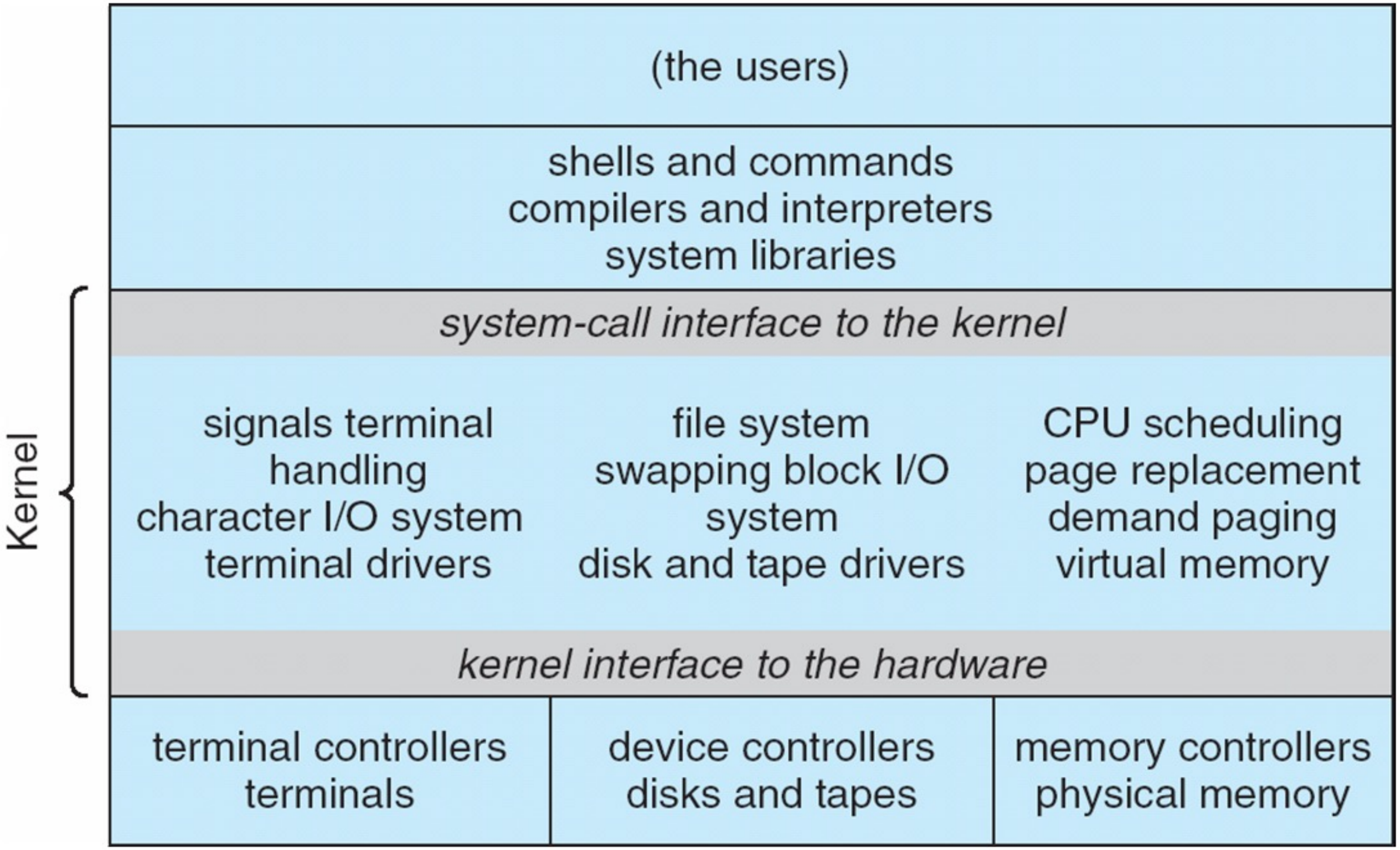
Operating System Structure

- Simple structure
  - ✓ MS-DOS



Operating System Structure

- Monolithic structure
  - ✓ Traditional Unix



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

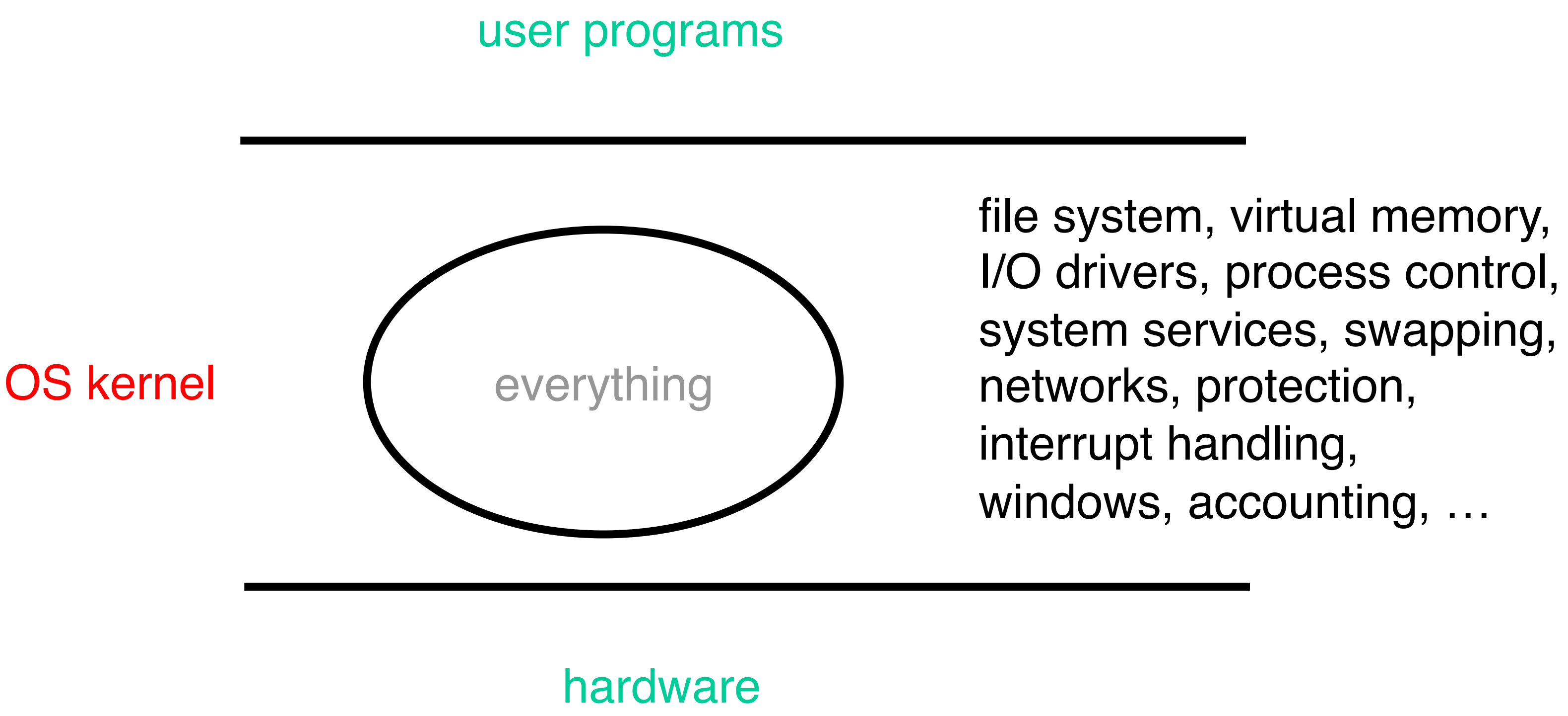
device controllers
disks and tapes

memory controllers
physical memory

Kernel

Operating System Structure

- Monolithic structure
  - ✓ Traditional Unix



user programs

OS kernel

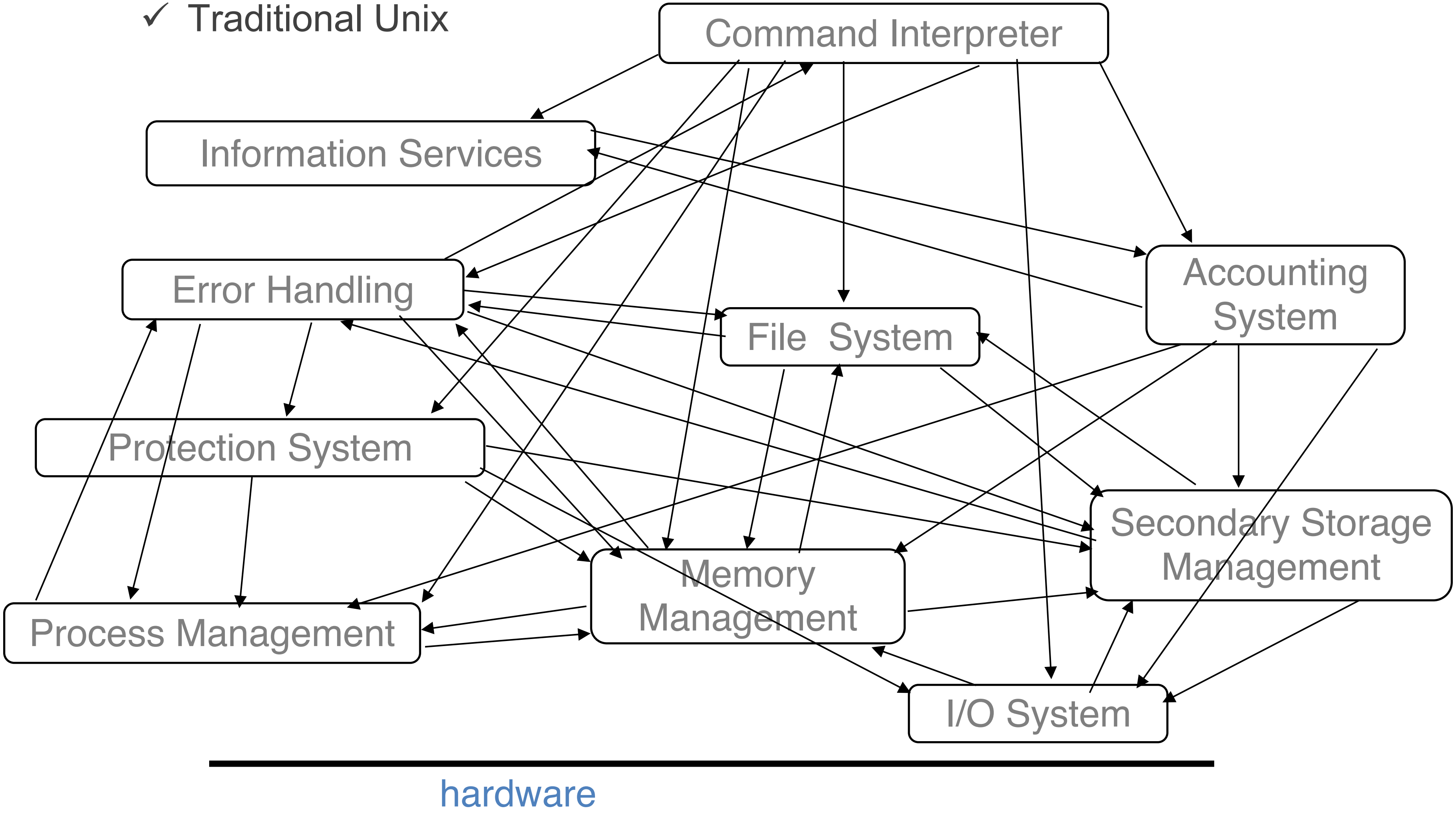
everything

file system, virtual memory,
I/O drivers, process control,
system services, swapping,
networks, protection,
interrupt handling,
windows, accounting, …

hardware

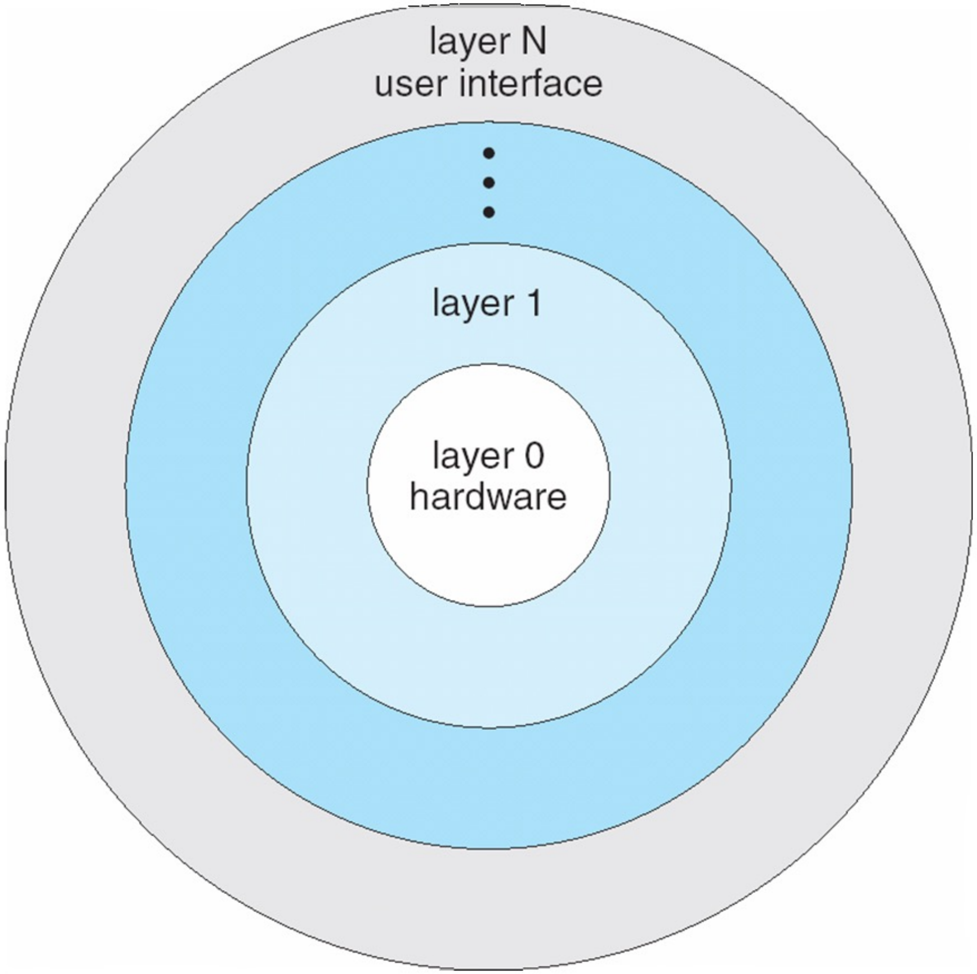
Operating System Structure

- Monolithic structure
  - ✓ Traditional Unix



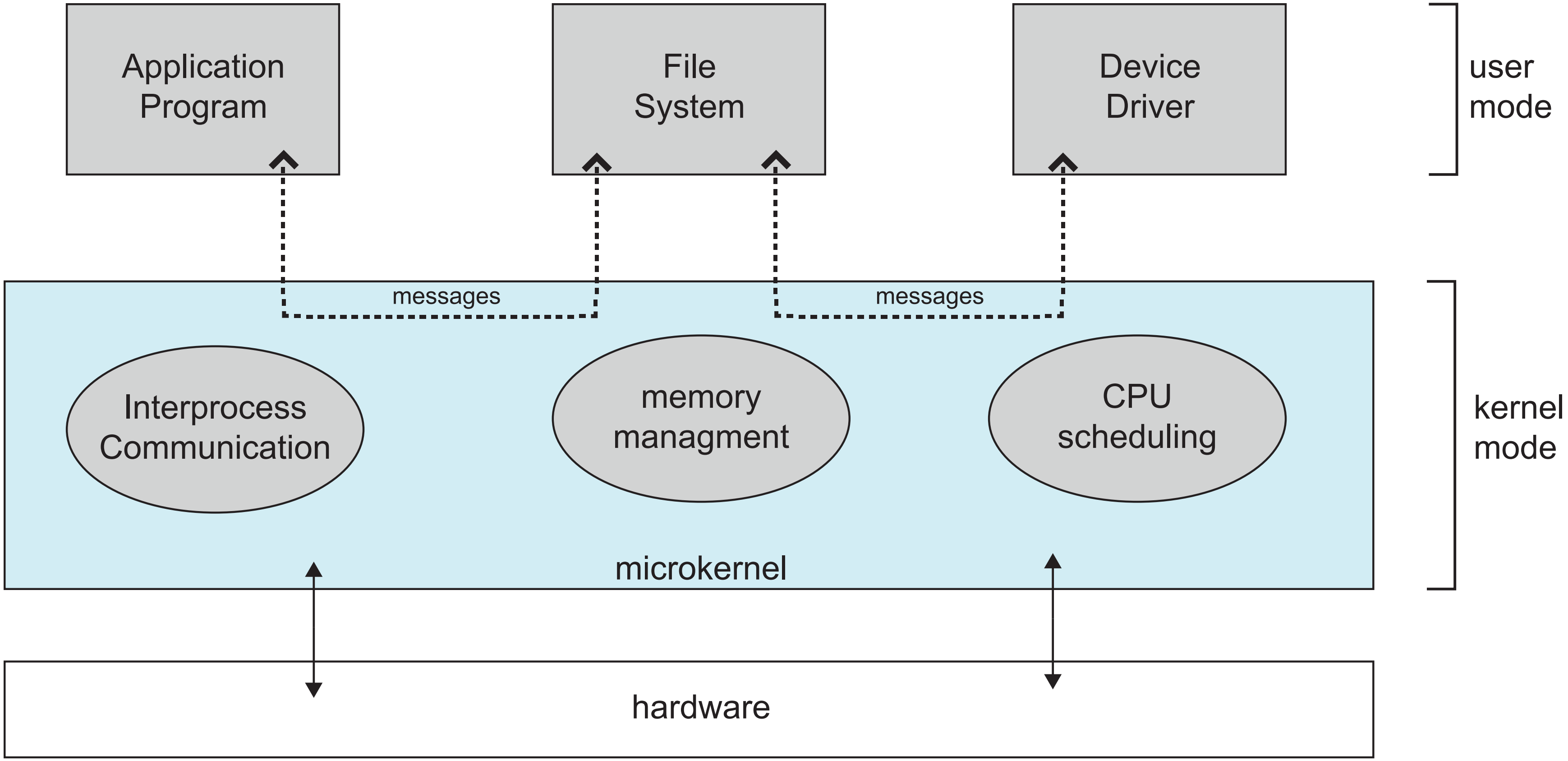
Operating System Structure

- Layered approach



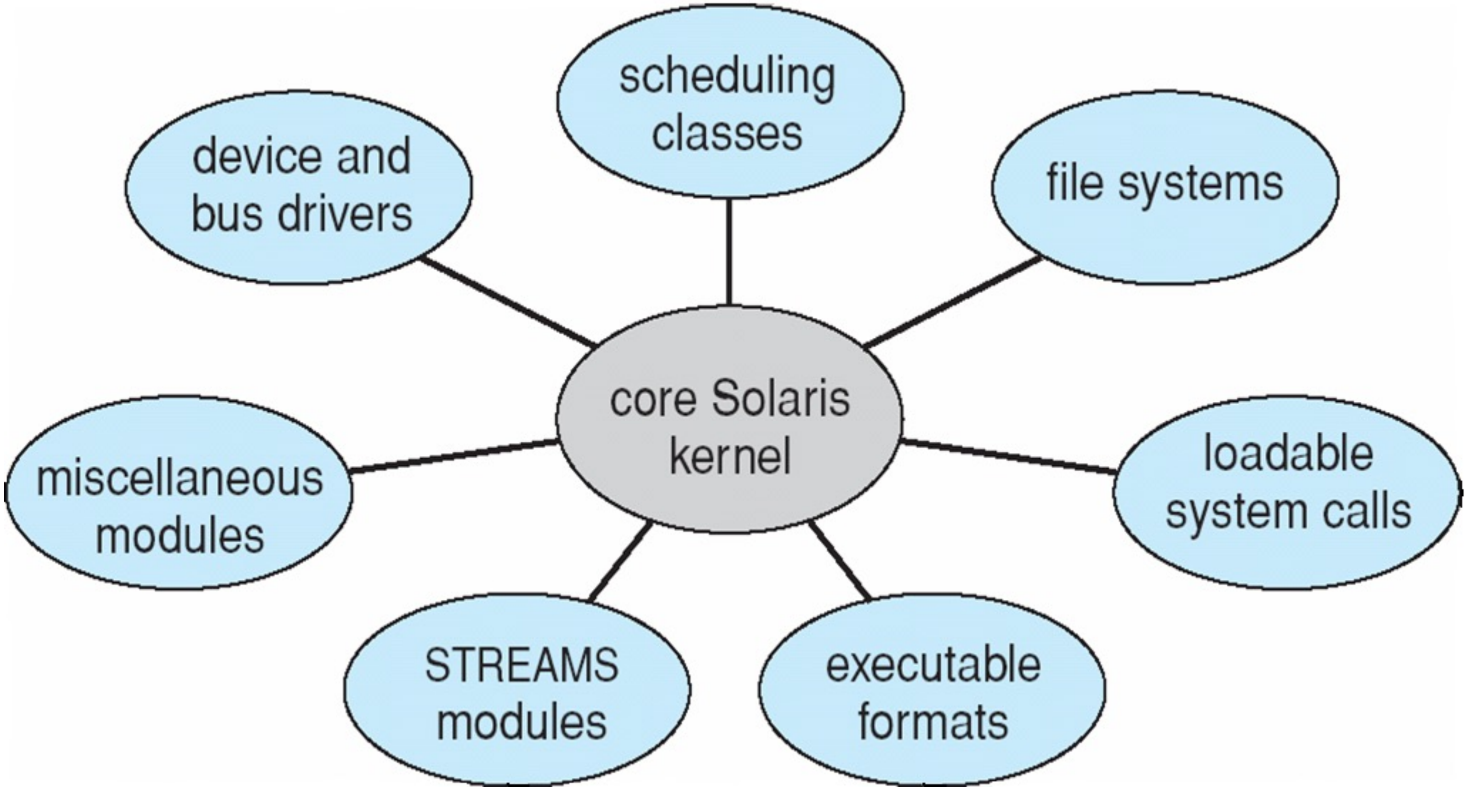
Operating System Structure

- Microkernel structure



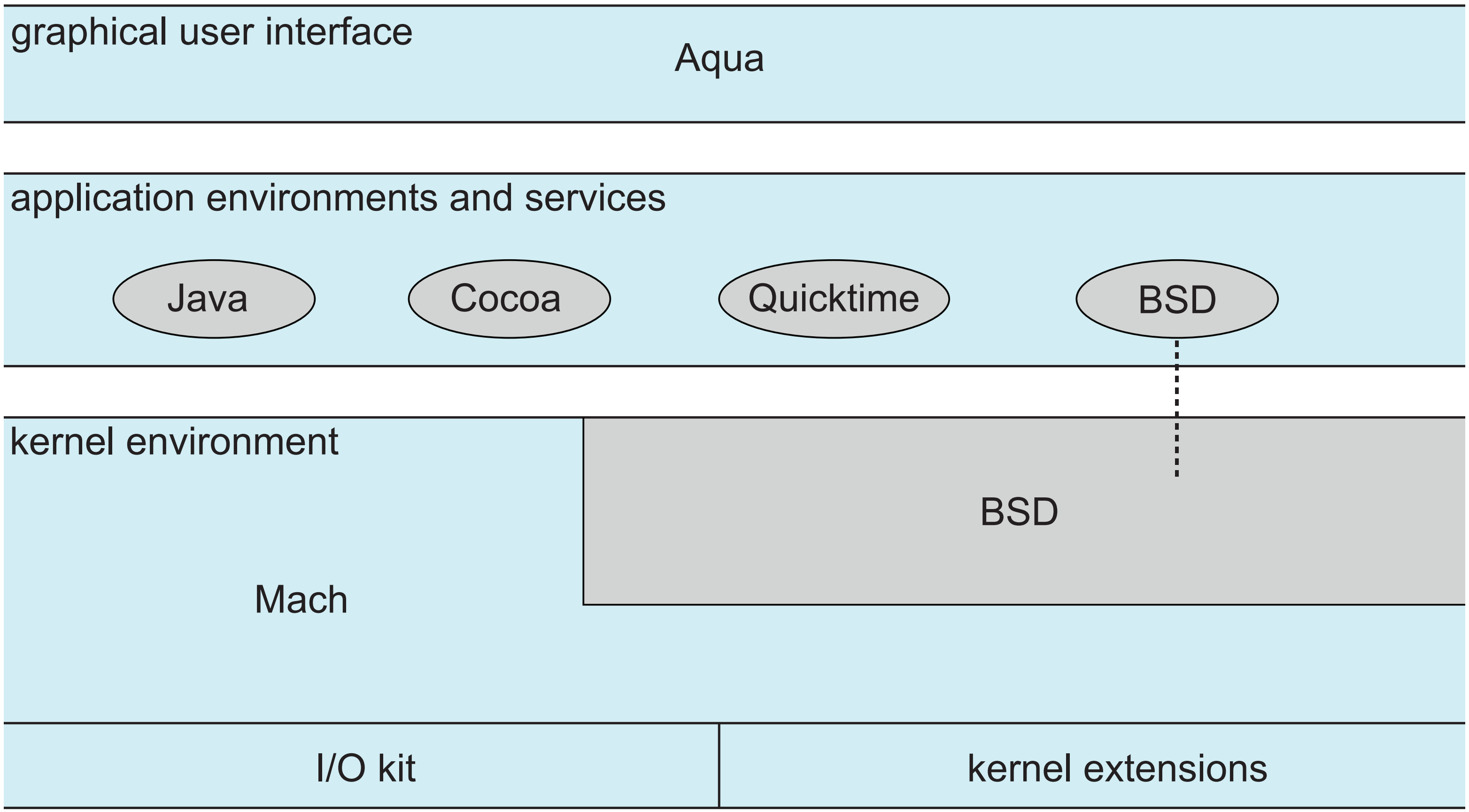
Operating System Structure

- Modular approach
  - ✓ Loadable Kernel Module (LKM)
  - ✓ Linux, Solaris, etc.



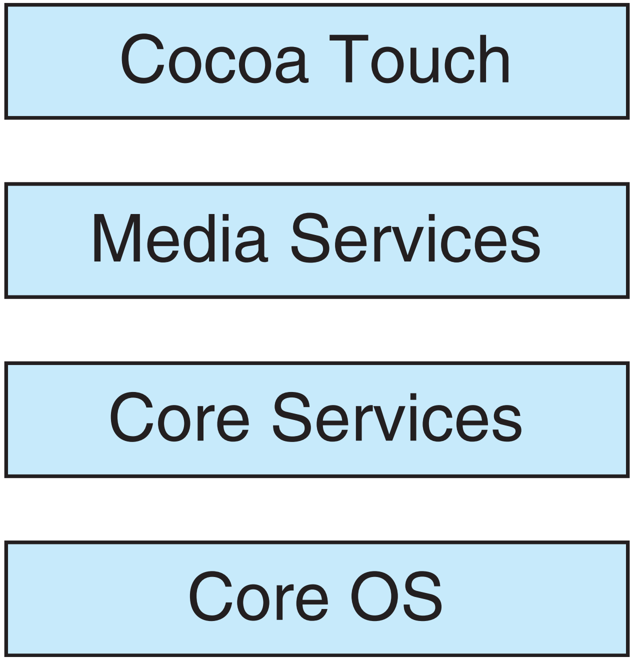
Operating System Structure

- Hybrid approach
  - ✓ Mac OS X



Operating System Structure

- Hybrid approach
  - ✓ iOS



Cocoa Touch

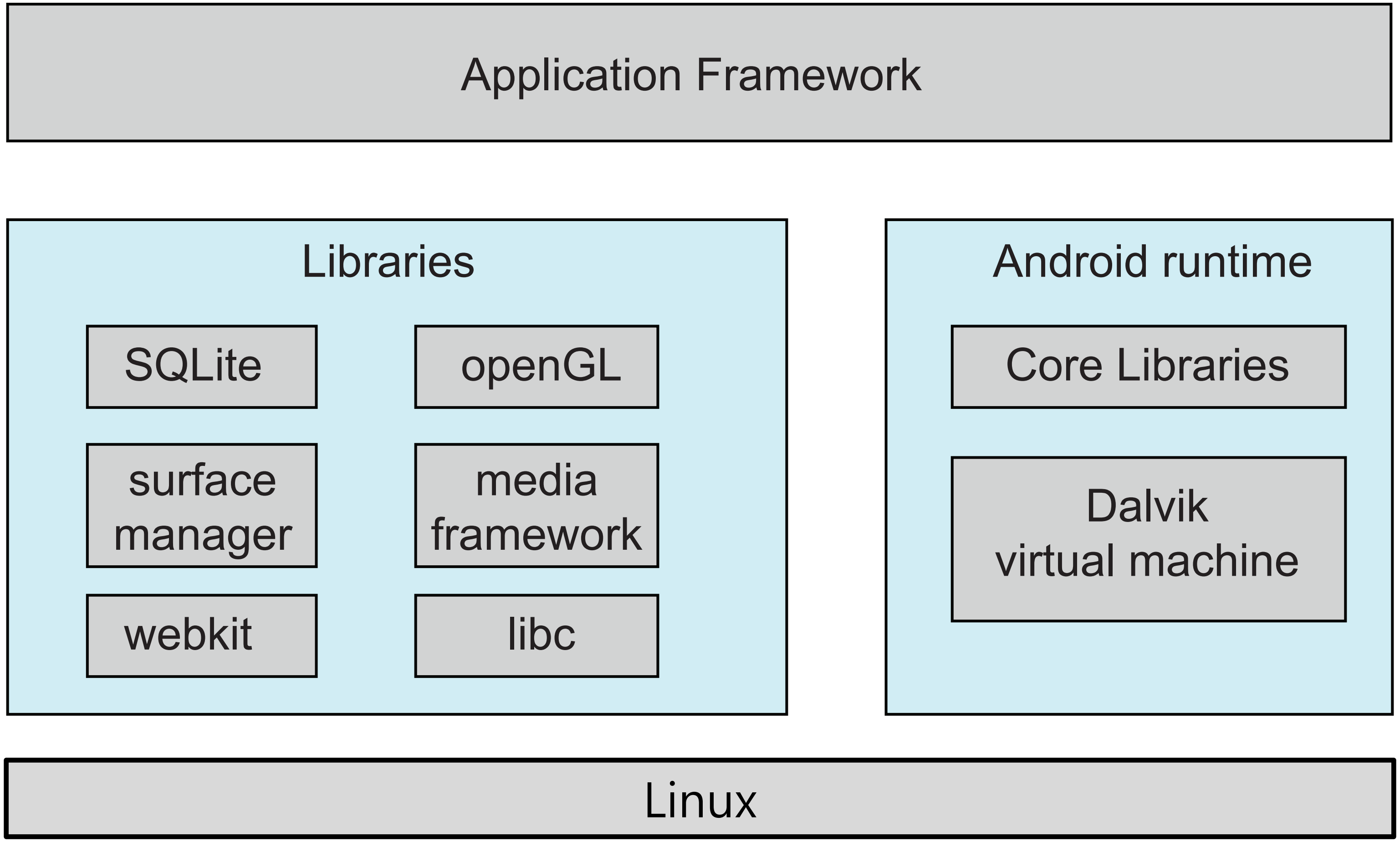
Media Services

Core Services

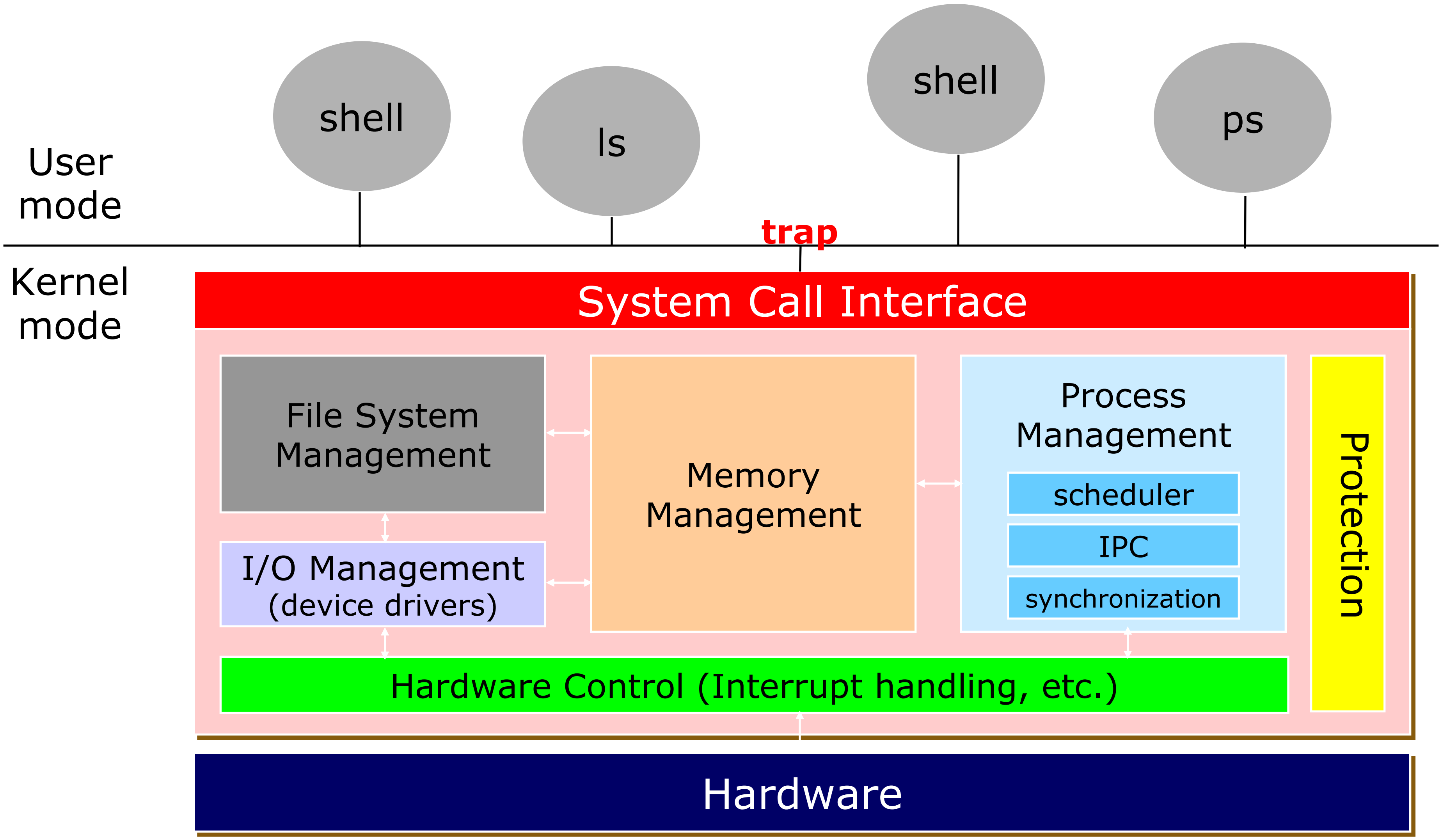
Core OS

Operating System Structure

- Hybrid approach
  - ✓ Android



Operating System



Operating System

- Manages computer HW resources
  - ✓ CPU management
    - Chapter 3: Processes
    - Chapter 4: Threads & Concurrency
    - Chapter 5: CPU Scheduling
    - Chapter 6: Synchronization Tools
    - Chapter 7: Synchronization Examples
    - Chapter 8: Deadlocks
  - ✓ Memory management
    - Chapter 9: Main Memory
    - Chapter 10: Virtual Memory
  - ✓ I/O management
    - Chapter 11: Mass-Storage Structure
    - Chapter 12: I/O Systems
    - Chapter 13: File-System Interface
    - Chapter 14: File-System Implementation
    - Chapter 15: File-System Internals
  - ✓ Chapter 16: Security

Thank You!
Q&A