Chapter 2: Operating-System Structures

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines

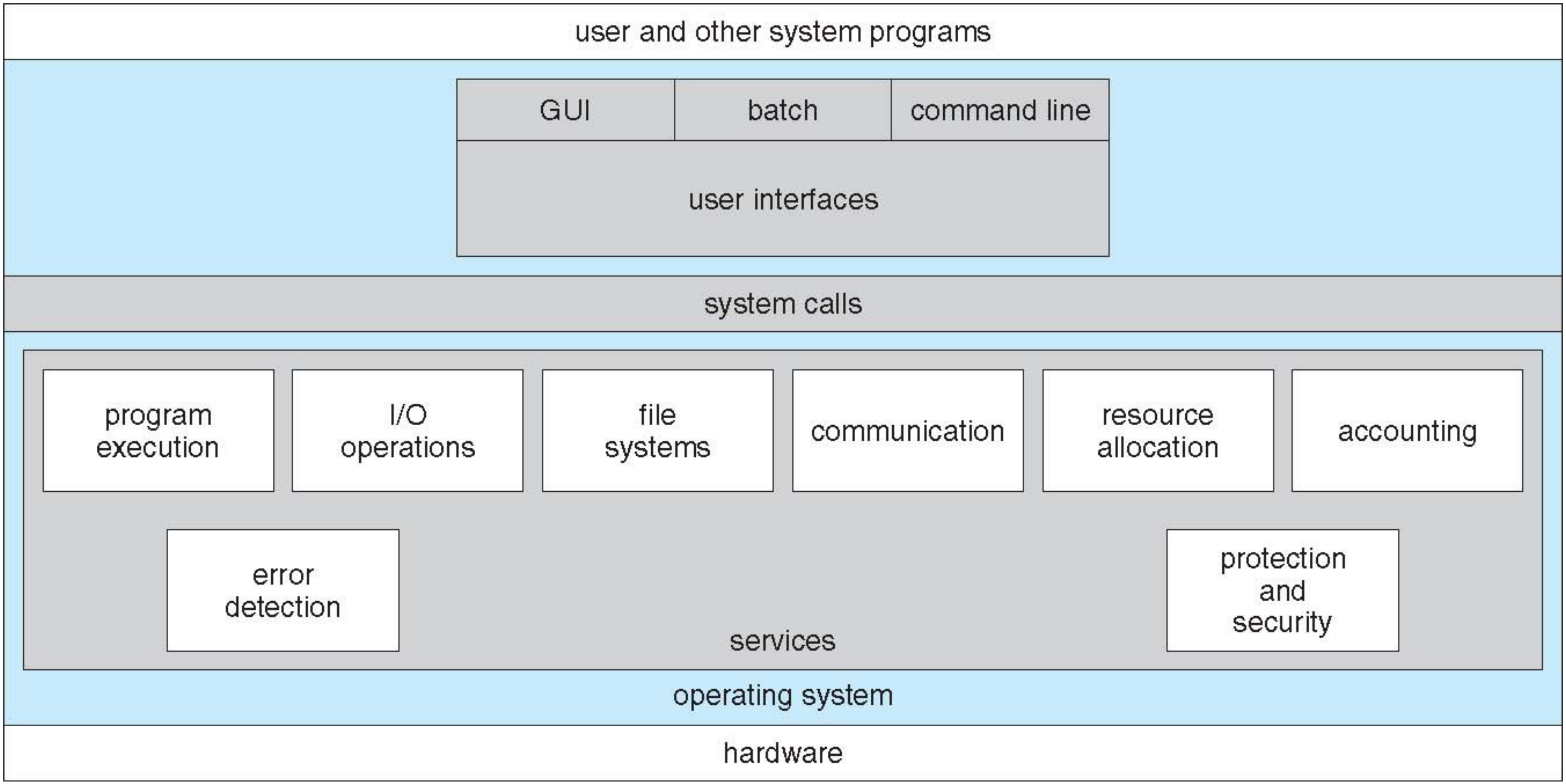
Objectives

- To describe **the services** an operating system provides to users, processes, and other systems
- To discuss the various ways of **structuring** an operating system

2.1 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)

A View of Operating System Services



2.1 Operating System Services (Cont)

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Obviously, 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)

2.1 Operating System Services (Cont)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **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

2.1 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 - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources

2.1 Operating System Services (Cont)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **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
    - If a system is to be protected and secure, precautions must be instituted throughout it. **A chain is only as strong as its weakest link.**

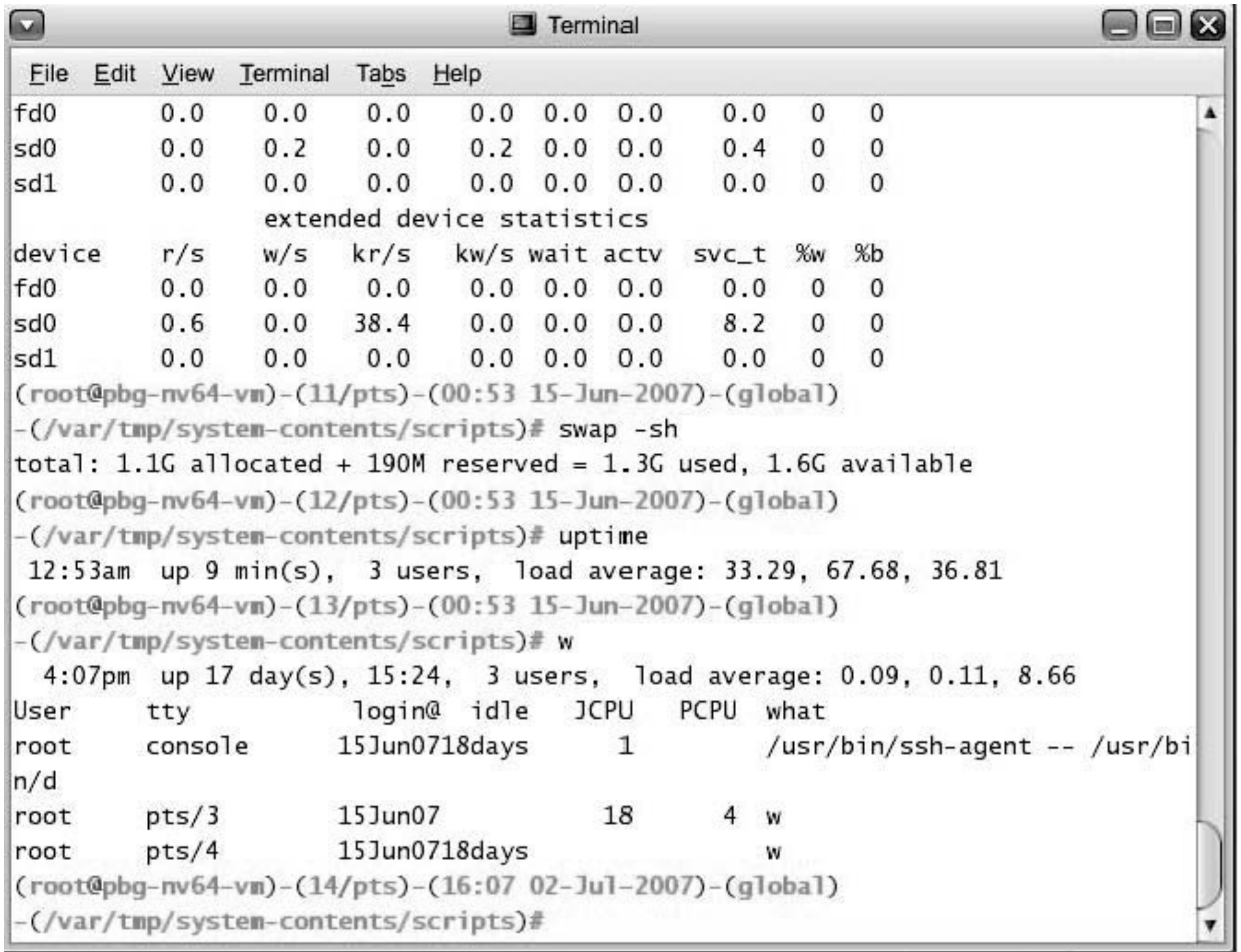
2.2 User Operating System Interface

- **Command Line Interface (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
- 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)

2.2 User Operating System Interface

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

Bourne Shell Command Interpreter



The Mac OS X GUI

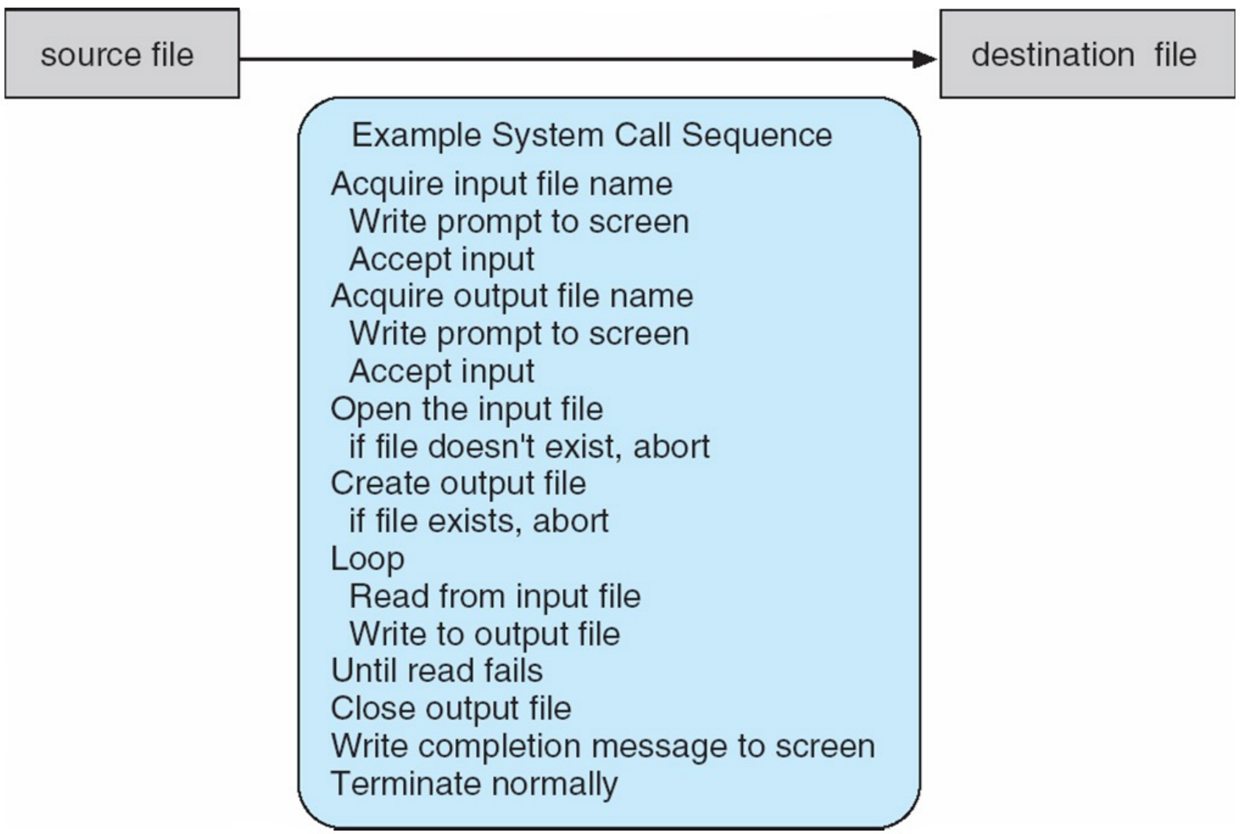
2.3 System Calls

- **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 Program 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)
- Why use APIs rather than system calls?

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

Example of System Calls

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



Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file

return value

```
BOOL   ReadFile c  (HANDLE          file,
                    LPVOID          buffer,
                    DWORD           bytes To Read,
                    LPDWORD         bytes Read,
                    LPOVERLAPPED    ovl);
```

function name

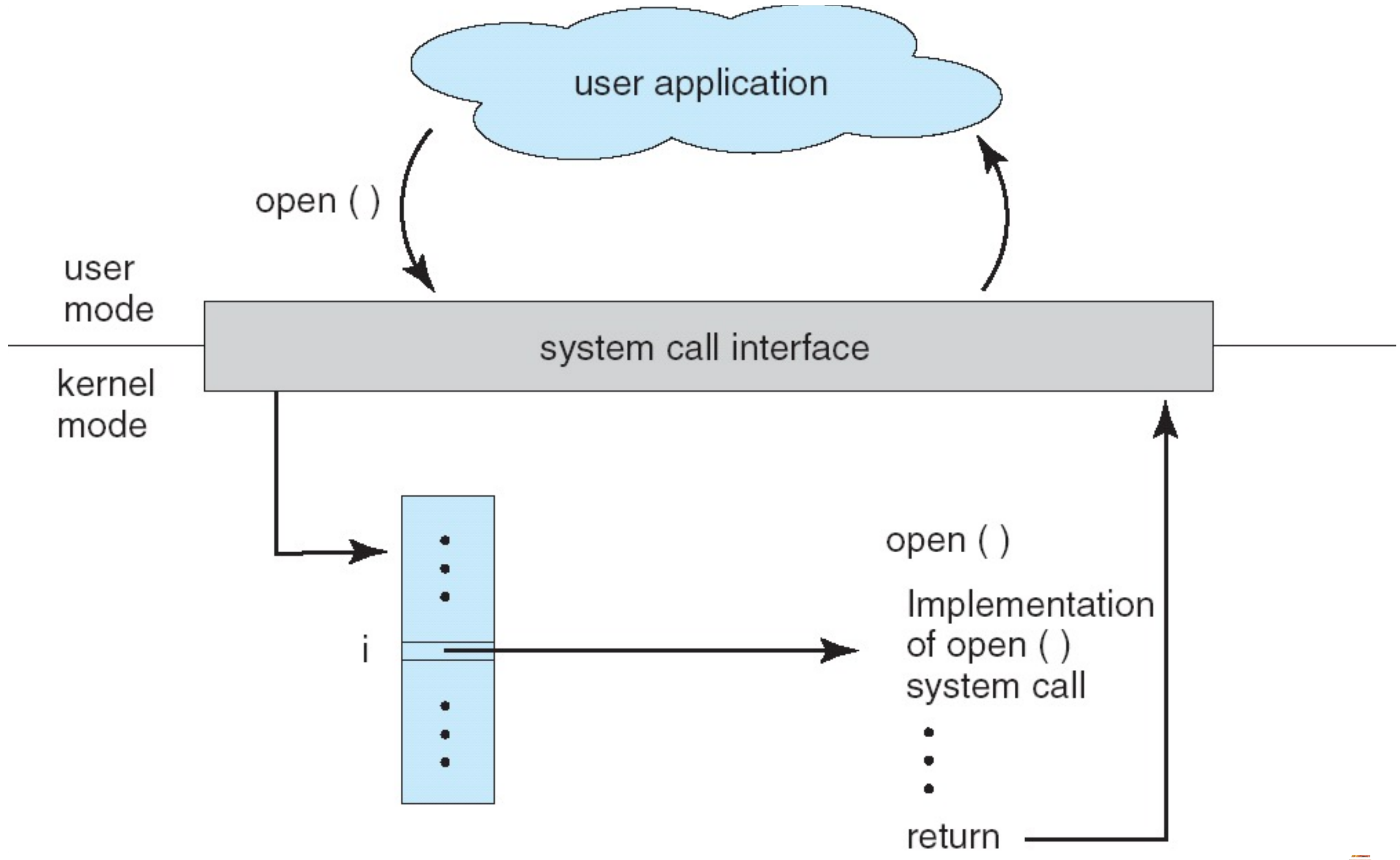
parameters

- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

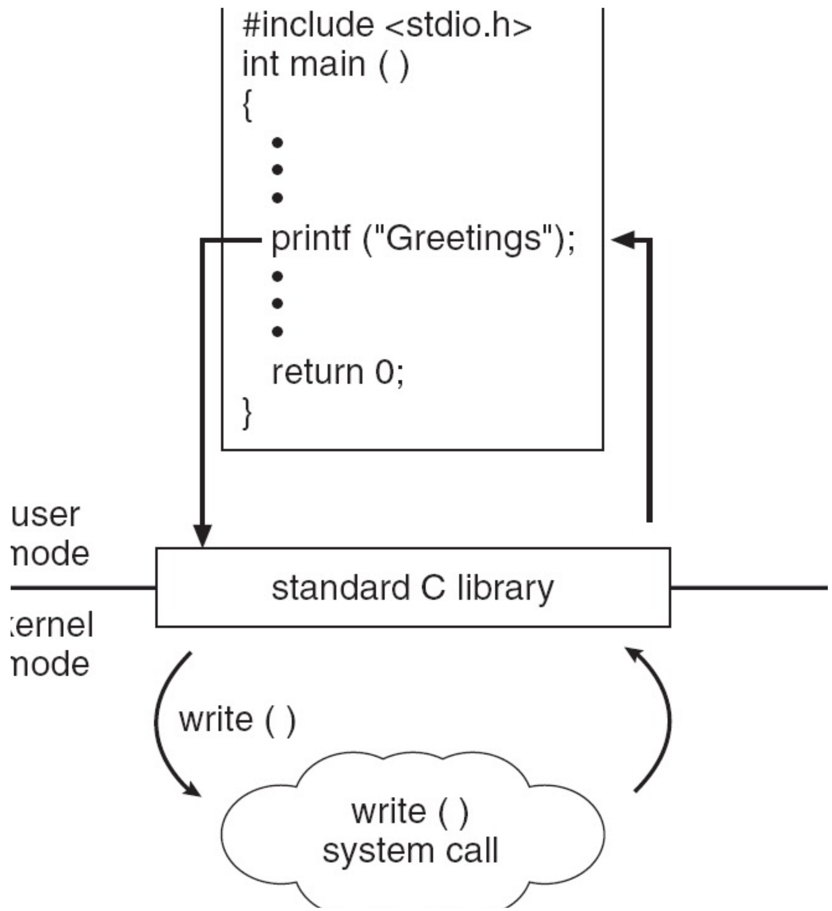
- Typically, **a number** associated with **each system call**
  - System-call interface maintains a table indexed according to these numbers
- The system call interface **invokes** intended system call in OS kernel and **returns** status of the system call and any return values
- The caller need know **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

API – System Call – OS Relationship



Standard C Library Example

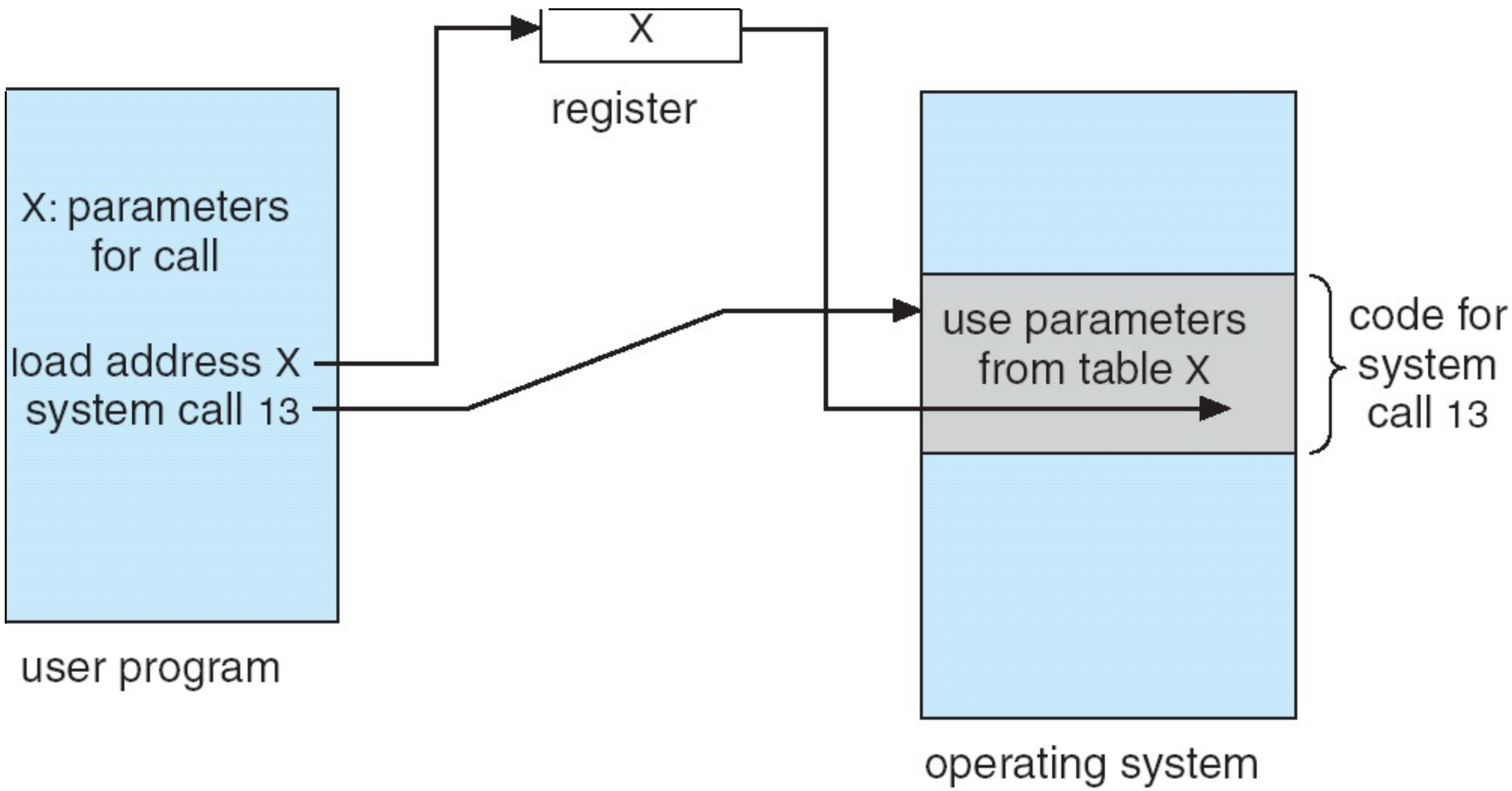
- C program invoking printf() library call, which calls write() system call



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



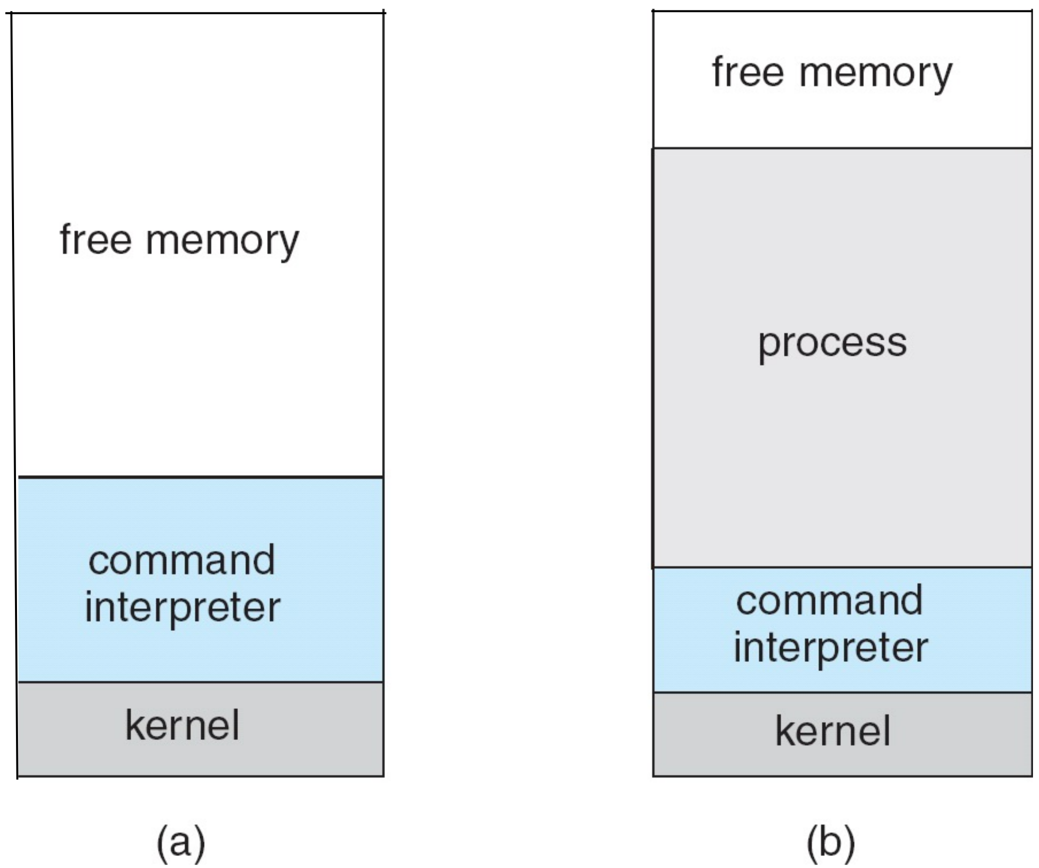
2.4 Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples of Windows and Unix System Calls

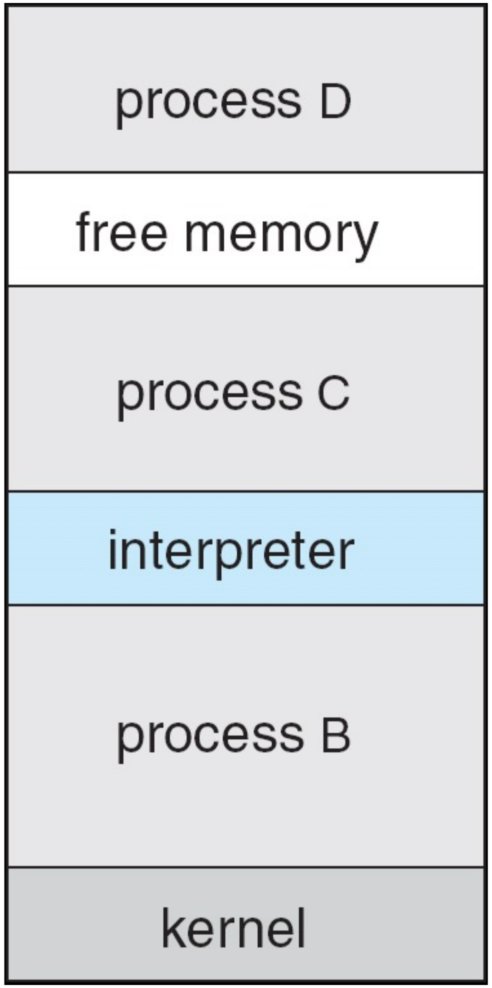
	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information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

MS-DOS execution



(a) At system startup (b) running a program

FreeBSD Running Multiple Programs



- Example of multi-tasking system.
- Command interpreter may continue to running while another program is executed.
- fork(): start a new process
- exec(): load a selected program to memory

2.5 System Programs

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

System Programs (cont'd)

- 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'd)

- 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

System Programs (cont'd)

- 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

2.6 Operating System Design and Implementation

- Design and Implementation of OS is not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start 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

2.6 Operating System Design and Implementation (Cont)

- Important principle to separate

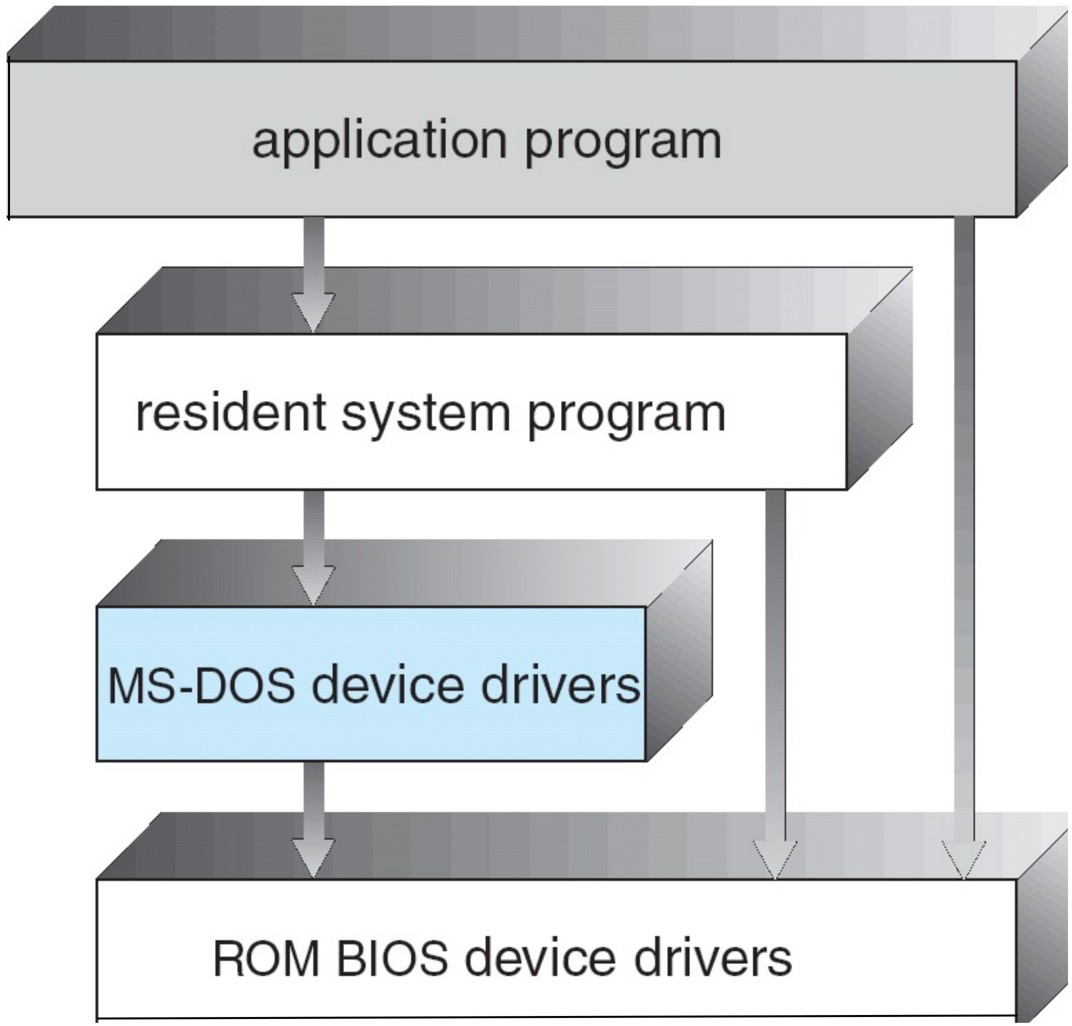
  Policy: What will be done?
  Mechanism: How to do it?

- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, **it allows maximum flexibility** if policy decisions are to be changed later

Simple Structure

- 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

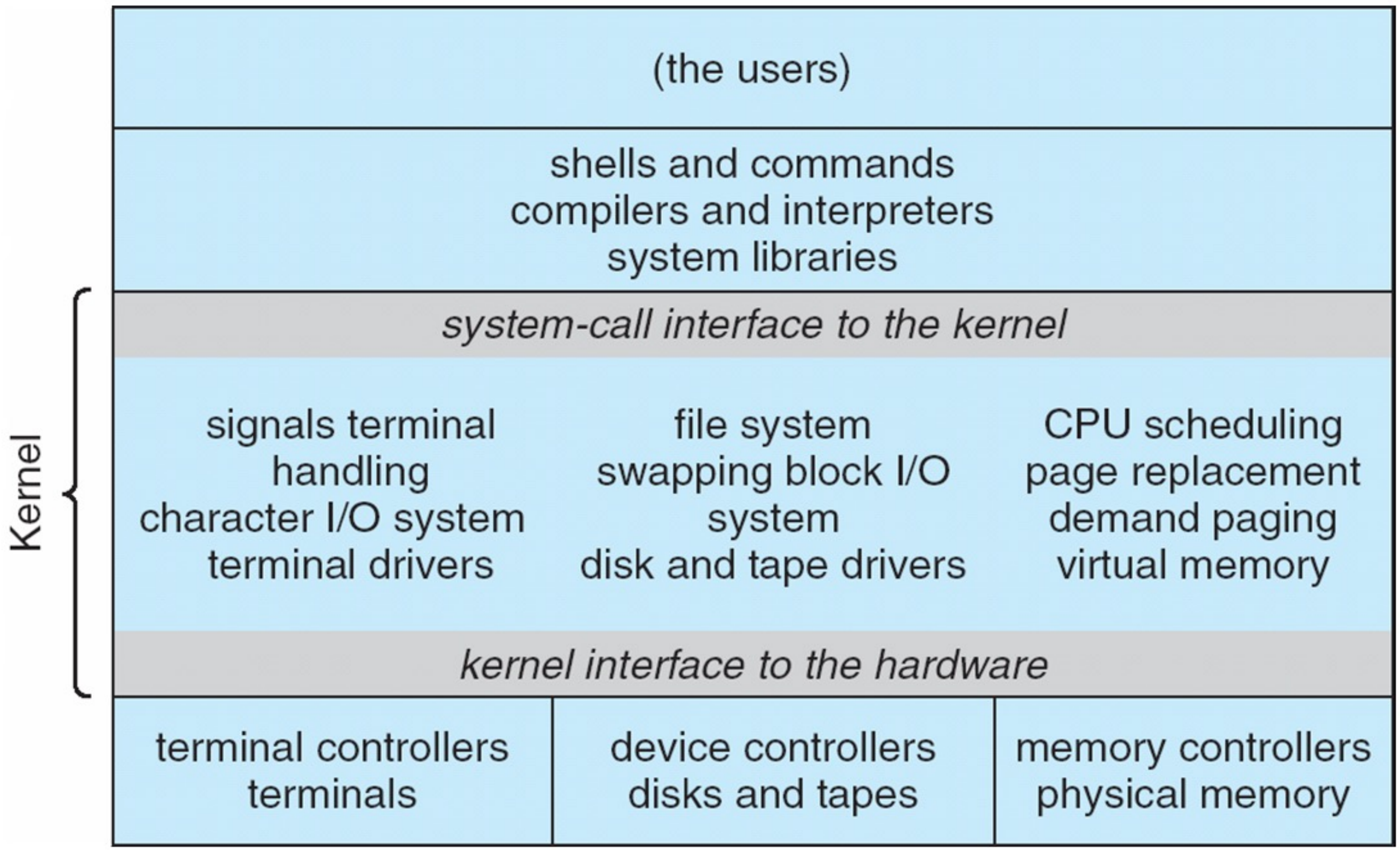
MS-DOS Layer Structure



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

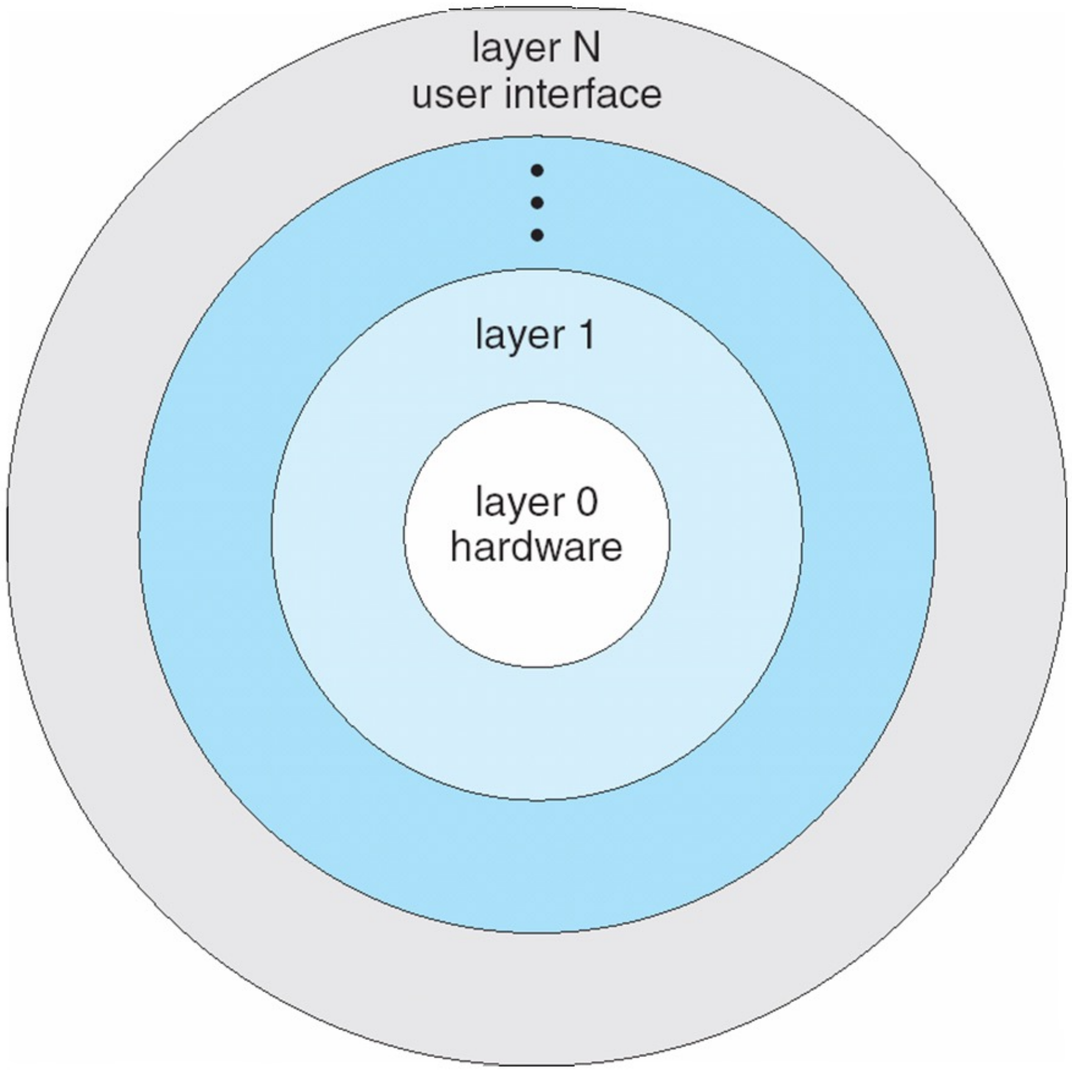
Traditional UNIX System Structure



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

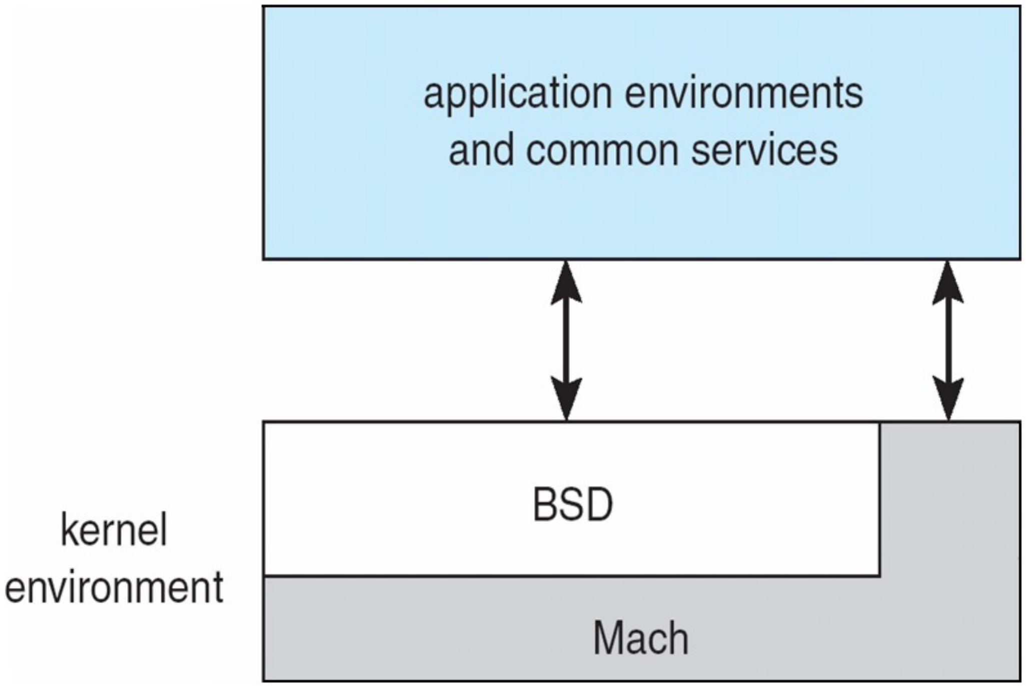
Layered Operating System



Microkernel System Structure

- Moves as much from the kernel into "*user*" space
- 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

Mac OS X Structure

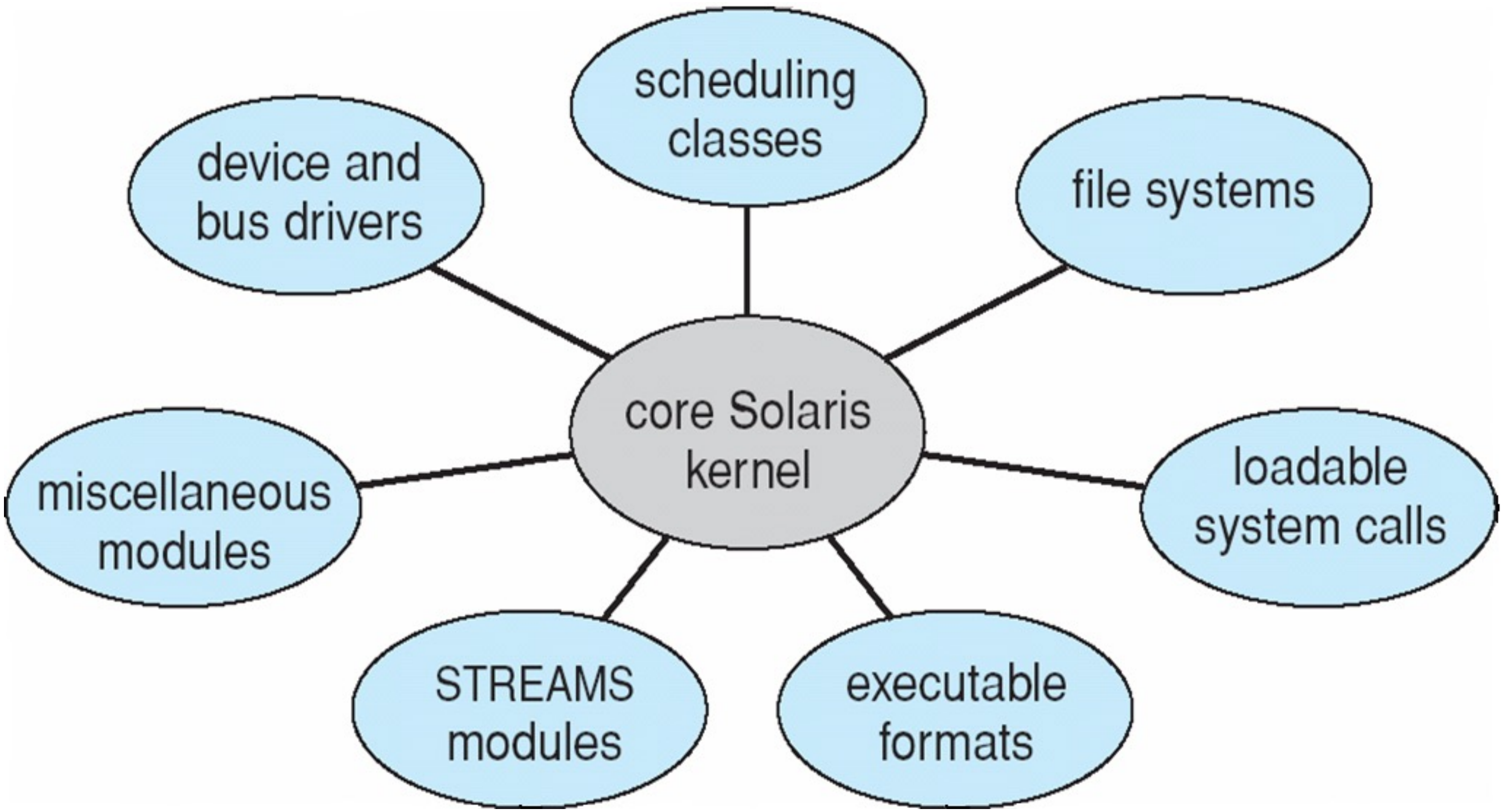


- Hybrid of layered system and microkernel.
- Mach provides memory management and interprocess communication
- BSC provide cli, networking, file system and POSIX APIs.

Modules

- Most modern operating systems implement 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

Solaris Modular Approach



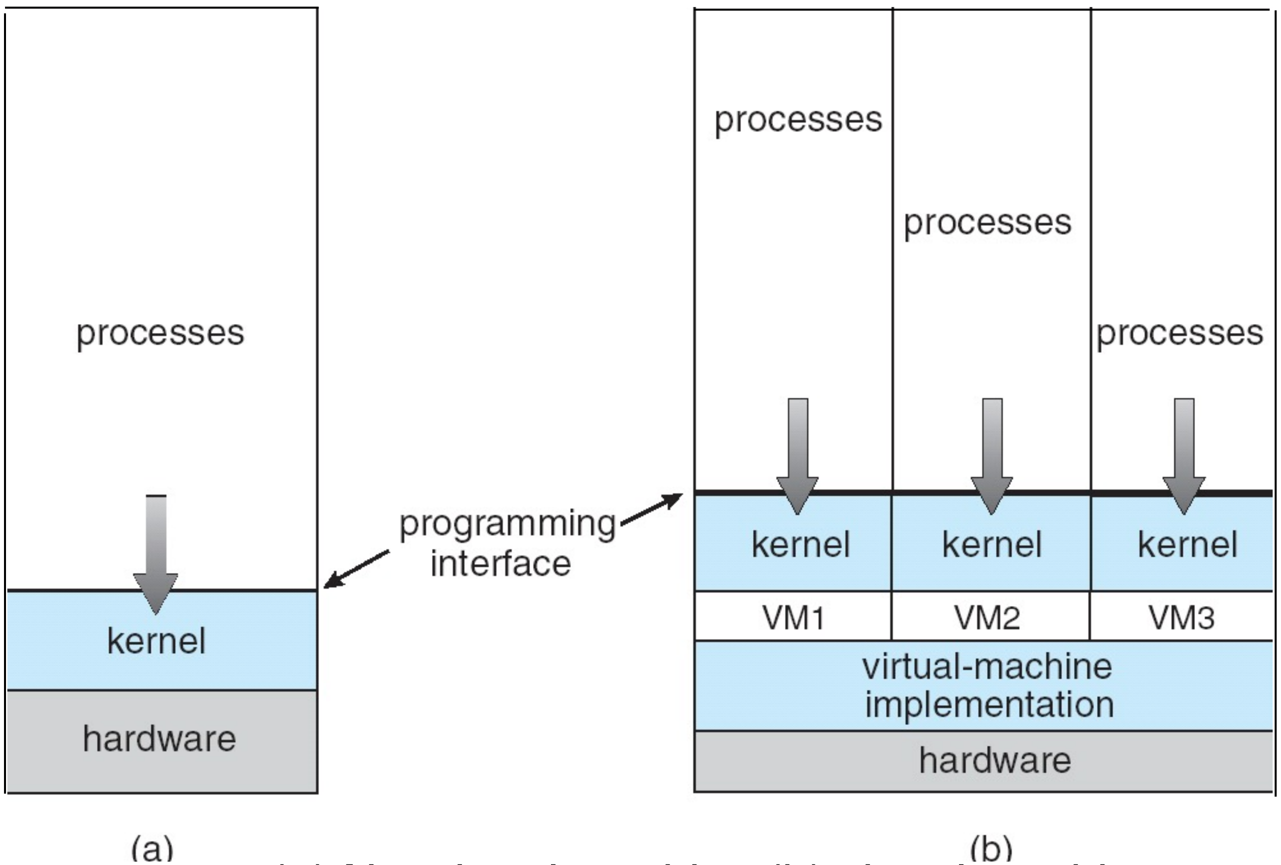
Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel **as though they were all hardware**
- A virtual machine provides an interface **identical** to the underlying bare hardware
- The operating system host creates the illusion that a process has its own processor and (virtual memory)
- Each guest provided with a (virtual) copy of underlying computer

Virtual Machines History and Benefits

- First appeared commercially in IBM mainframes in 1972
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled
- Commutate with each other, other physical systems via networking
- Useful for development, testing
- Consolidation of many low-resource use systems onto fewer busier systems
- "Open Virtual Machine Format", standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms

Virtual Machines (Cont)

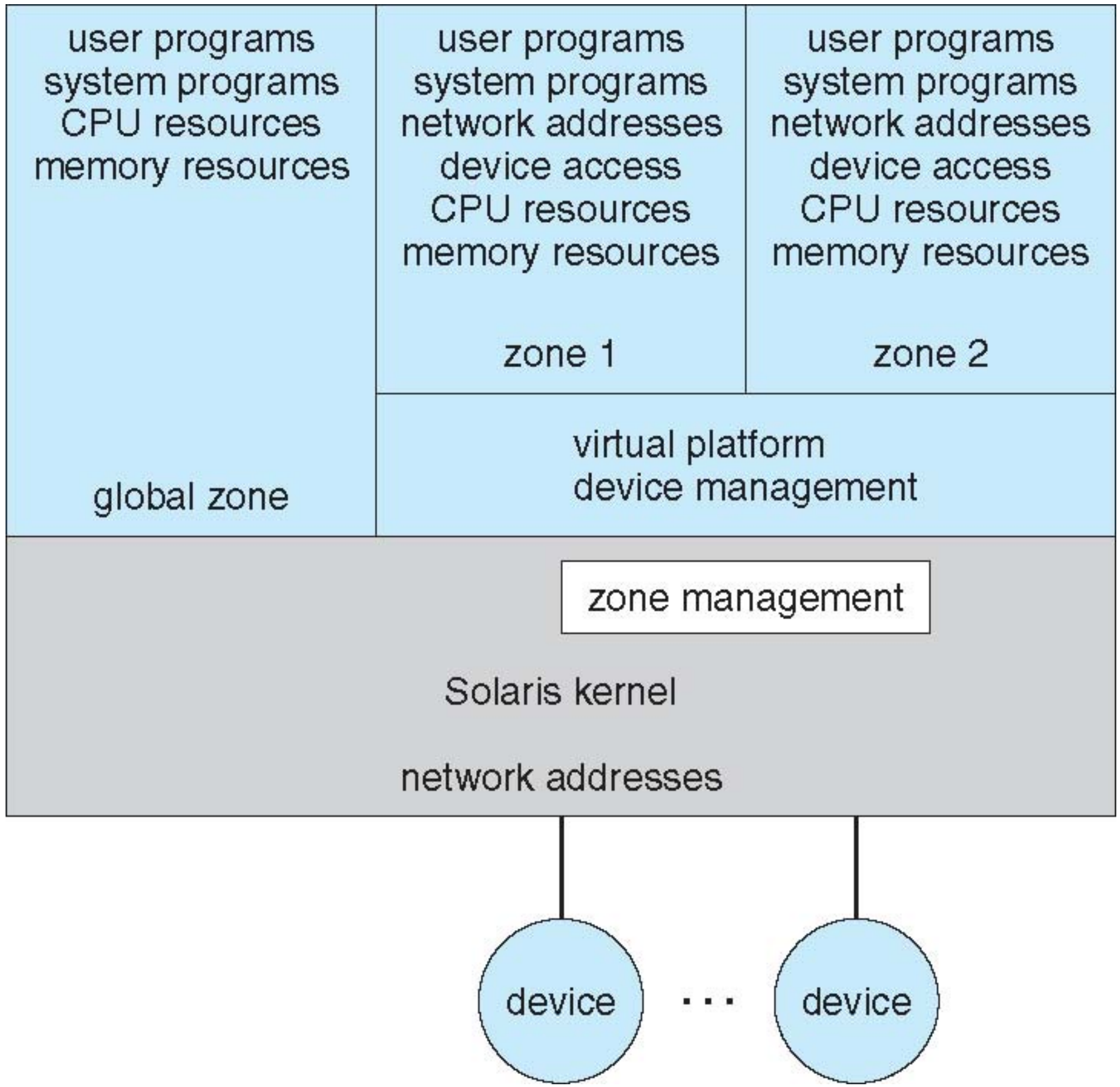


(a) Nonvirtual machine (b) virtual machine

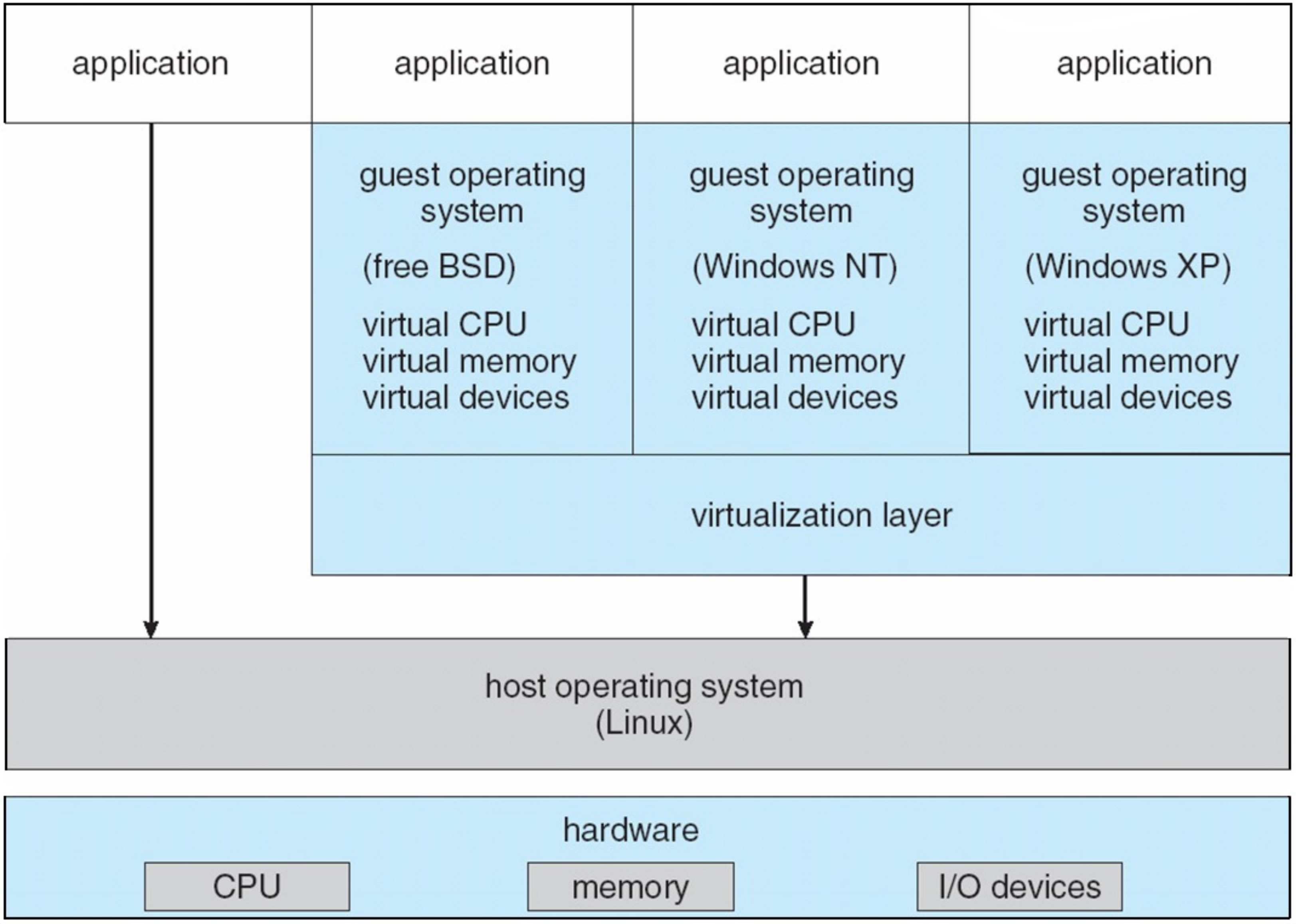
Para-virtualization

- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardwareF
- Guest can be an OS, or in the case of Solaris 10 applications running in containers

Solaris 10 with Two Containers



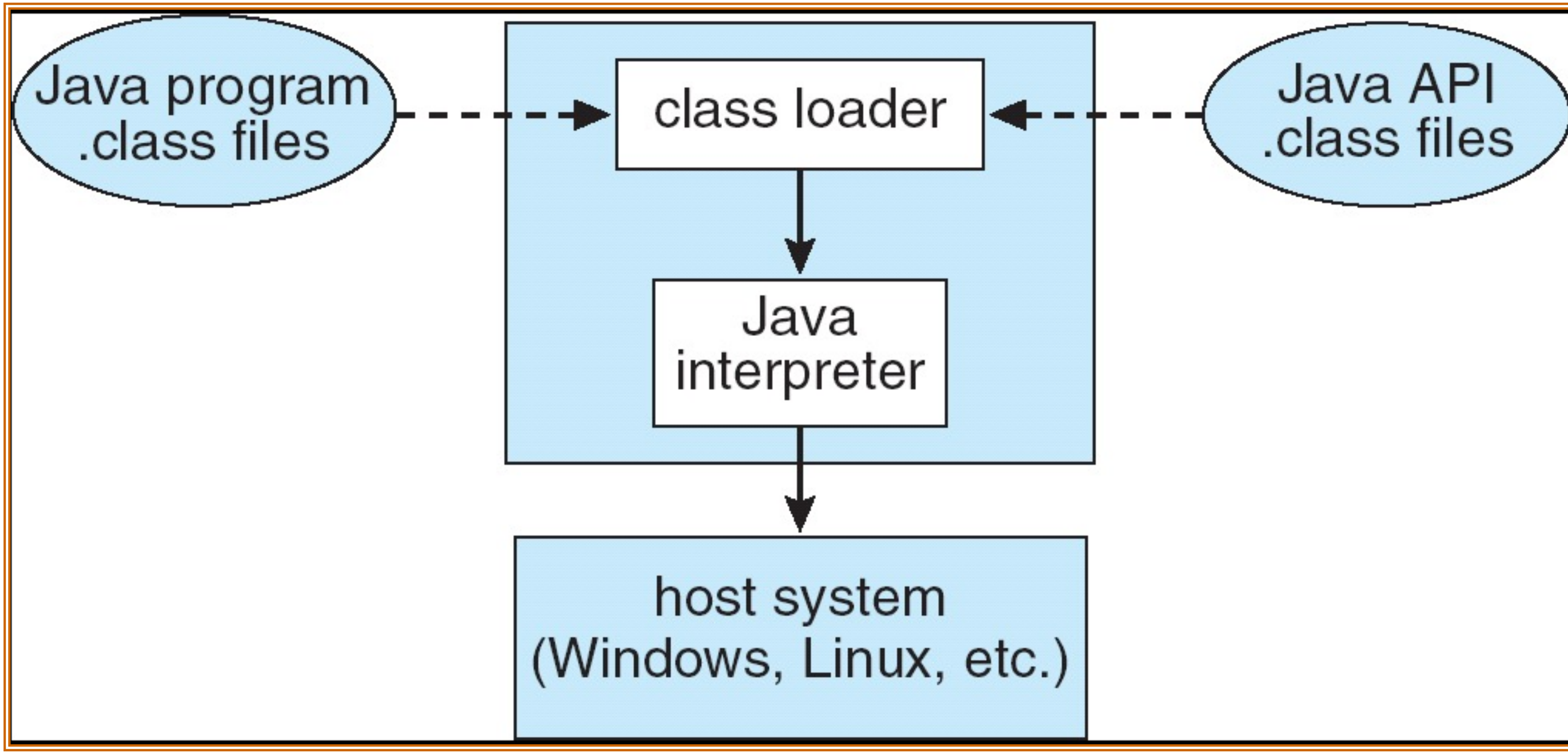
VMware Architecture



Java

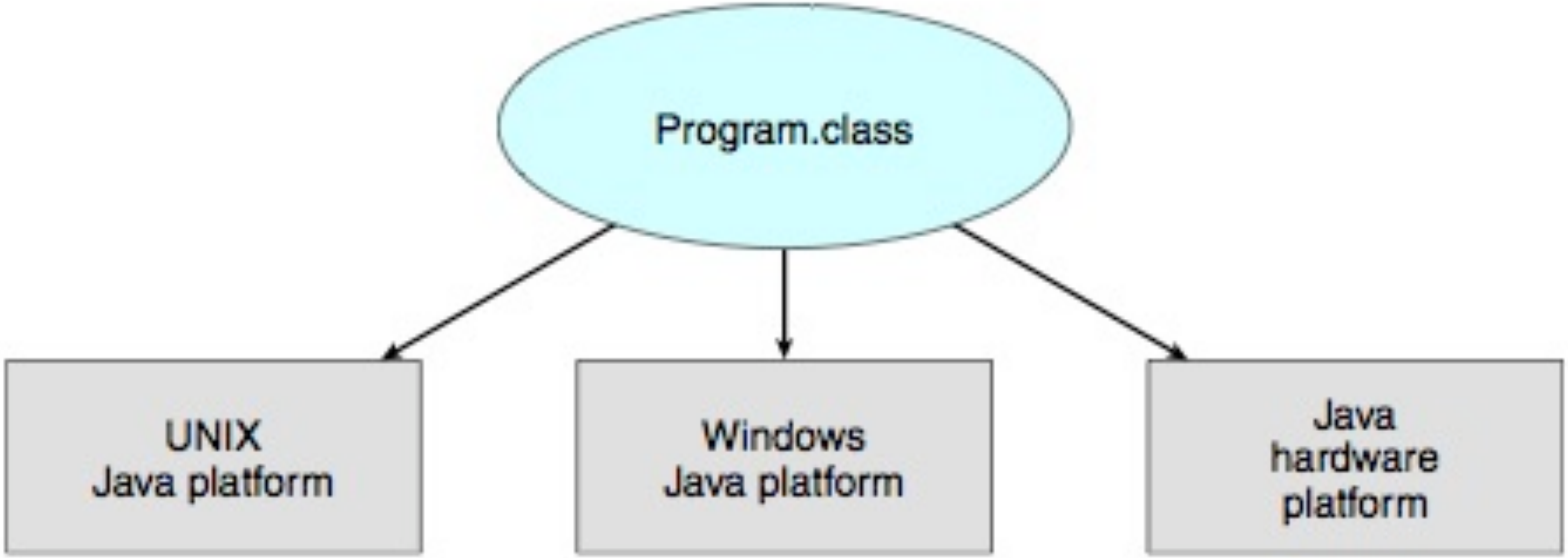
- Java consists of
  1. Programming language specification
  2. Application programming interface (API)
  3. Virtual machine specification

The Java Virtual Machine

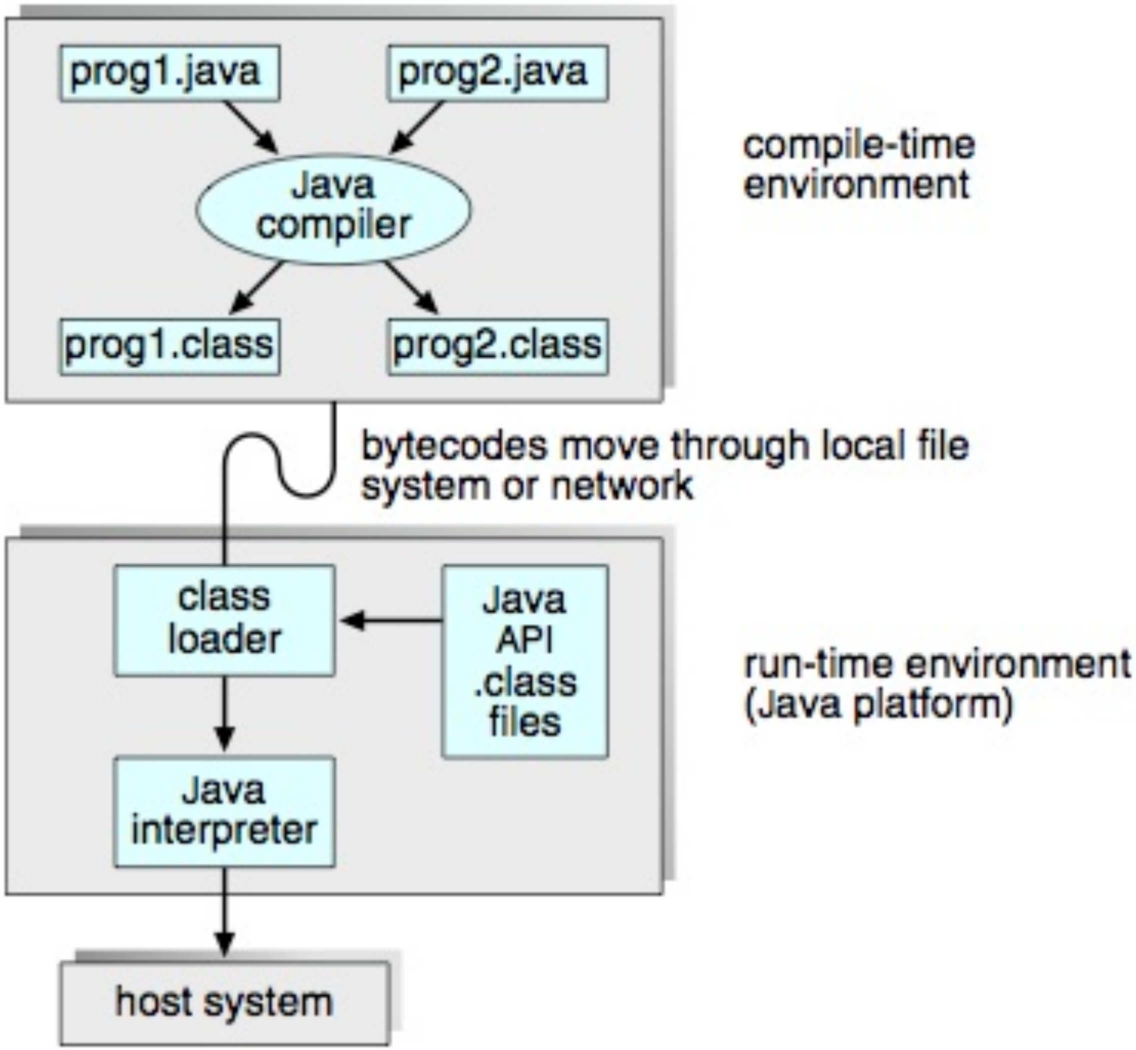


The Java Virtual Machine

Java portability across platforms.

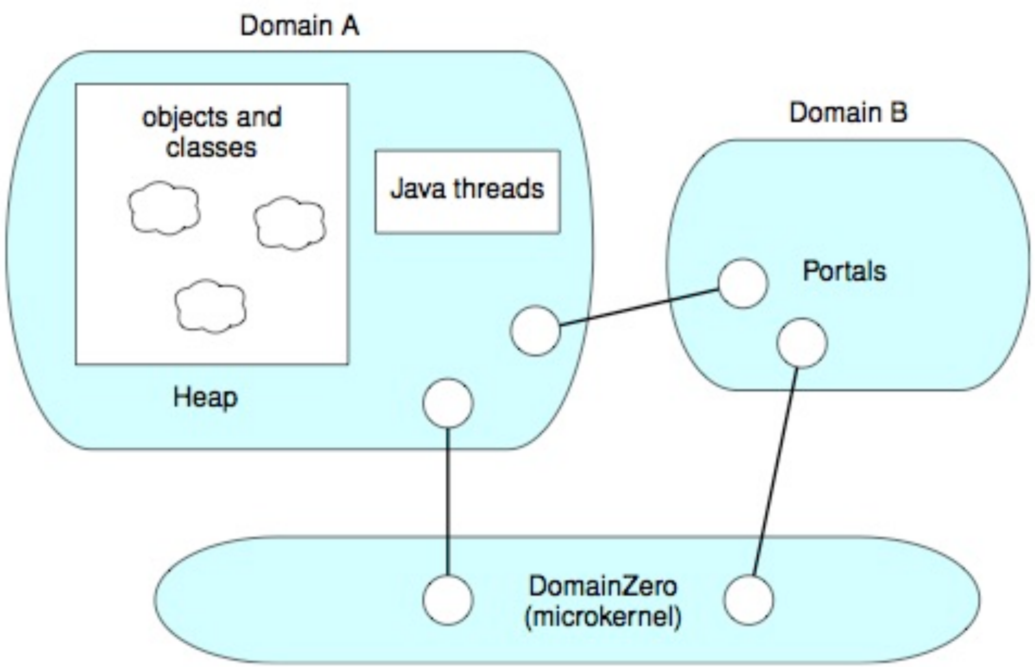


The Java Development Kit



Java Operating Systems

The JX operating system



End of Chapter 2