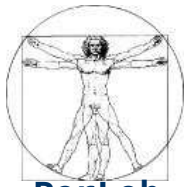
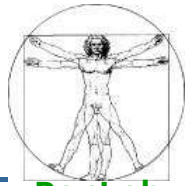
1. Basic Concepts

Giuseppe Anastasi
g.anastasi@iet.unipi.it
Pervasive Computing & Networking Lab. (PerLab)
Dept. of Information Engineering, University of Pisa



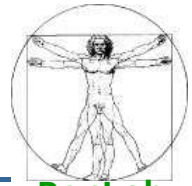
Based on original slides by Silberschatz, Galvin and Gagne

Overview



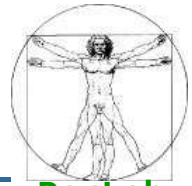
- Preliminary Concepts
- Services
- System Calls
- System Programs
- Internal Structure
- System Boot

Overview



- **Preliminary Concepts**
- Services
- System Calls
- System Programs
- Internal Structure
- System Boot

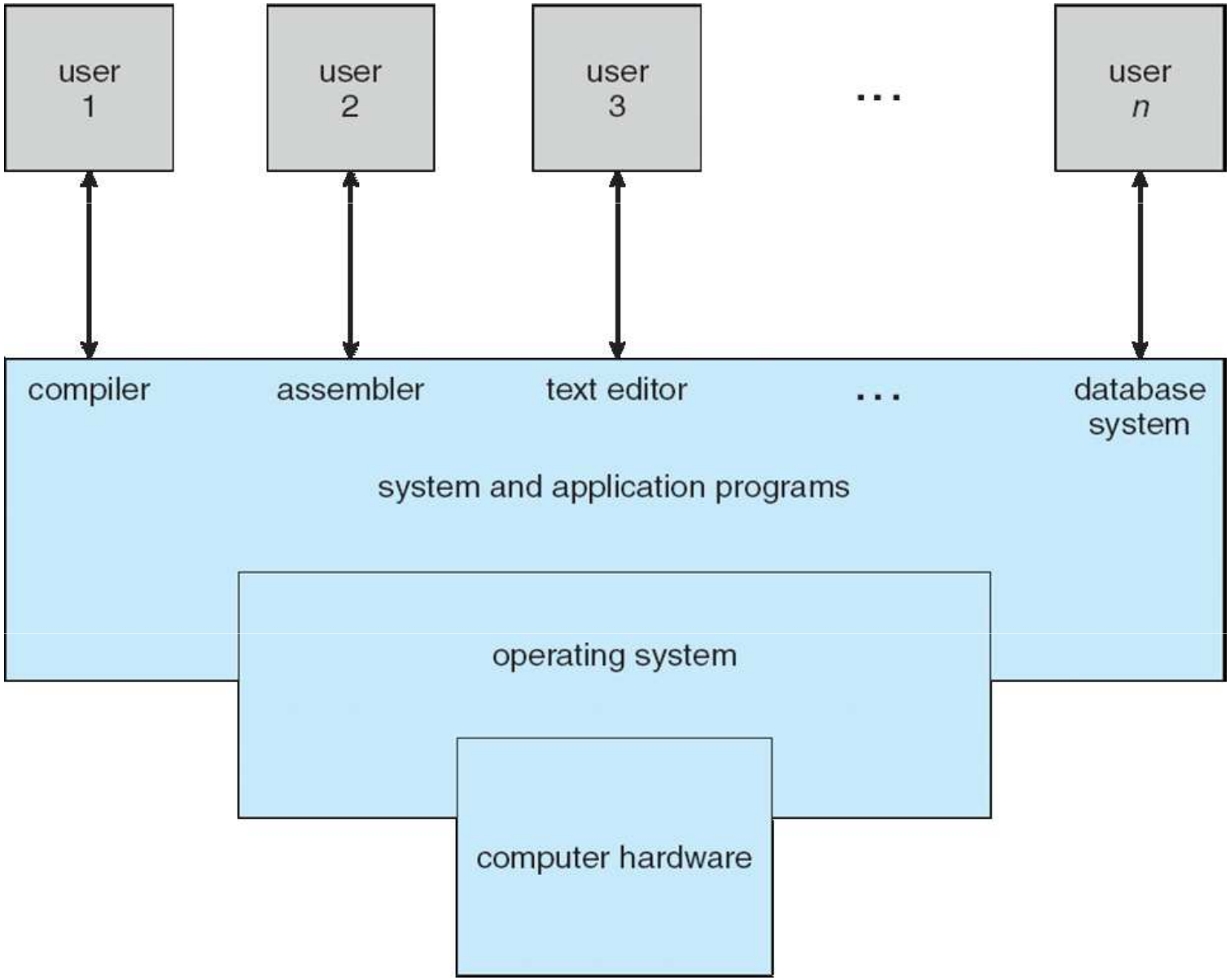
What is an Operating System?



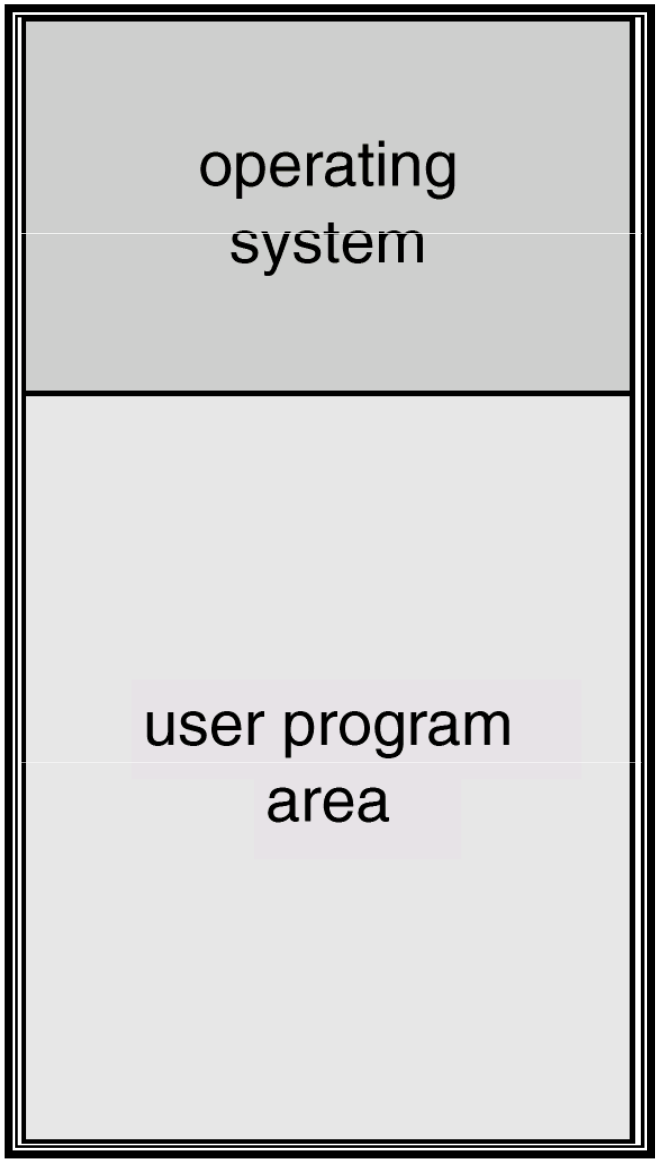
- A program that acts as an intermediary between a user of a computer and the computer hardware.

- Operating system goals:
  - Provide an environment for executing user programs and making solving user problems easier.
  - Make the computer system convenient to use.
  - Use the computer hardware in an efficient manner.

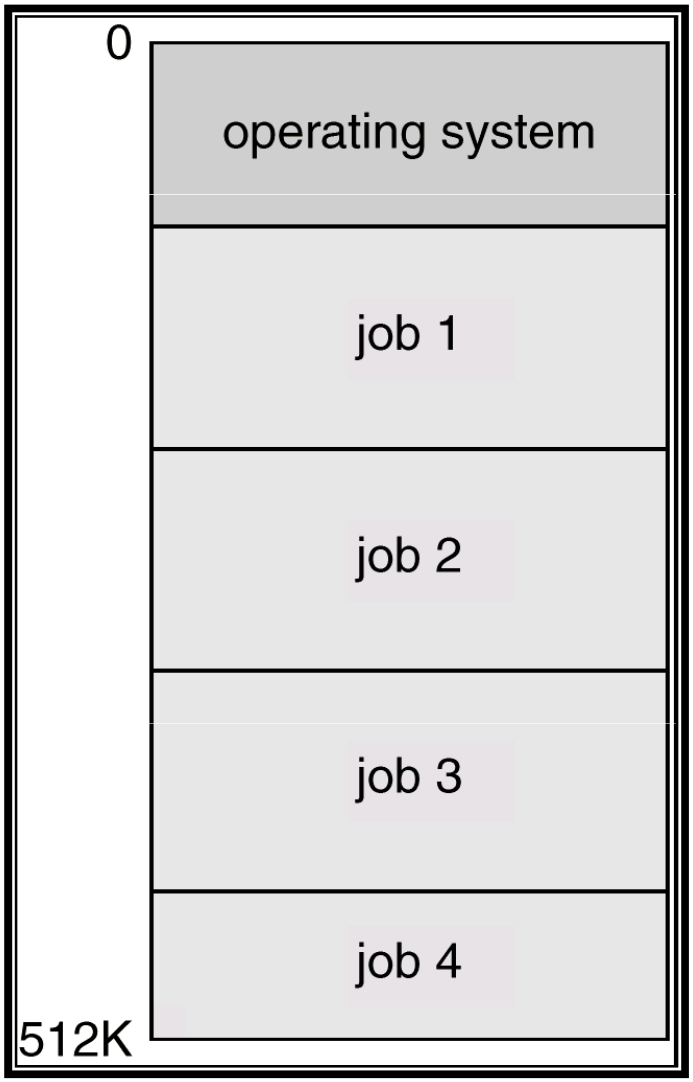
Abstract View



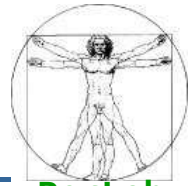
Mono-programmed Systems



Multi-programmed Systems



Features for Multiprogramming

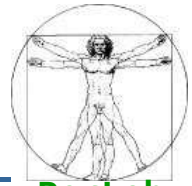


- **Memory management**
  - the system must allocate the memory to several jobs.
- **CPU scheduling**
  - the system must choose among several jobs ready to run.
- **Device management**
  - Allocation of devices to concurrent processes

System Classification (1)

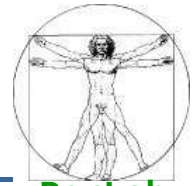
- Batch systems
  - No interaction with the user
- Interactive systems
  - The user interacts with the systems during process execution
  - Response times should be short
- Real time systems
  - Soft real-time systems
  - Hard real-time systems
- General-purpose systems
  - The system has to manage a mix of batch, interactive and soft real-time processes

System Classification (2)



- Mainframe systems
- Desktop systems
- Server systems
- Parallel systems
- Distributed systems
- Cluster systems
- Embedded systems
- Hand-held systems

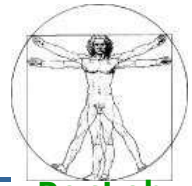
Goals of an Operating System



- Make the computer easy to use (e.g. PC)
- Optimize the computational resources (e.g. mainframe)
- Optimize shared resources (e.g., distributed systems)
- Make the computer easy to use and optimize energetic resources (e.g., handheld computers)
- …

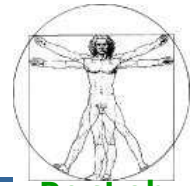
Convenience vs. efficiency

Overview



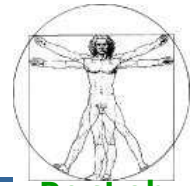
- Preliminary Concepts
- **Services**
- System Calls
- System Programs
- Internal Structure
- System Boot

Operating System Services



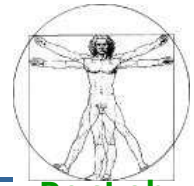
- User Interface
  - Command-Line (CLI)
  - Graphics User Interface (GUI)
  - Batch
- Program execution
  - system capability to load a program into memory and to run it.
- I/O operations
  - since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation
  - program capability to read, write, create, and delete files.

Operating System Services



- Communications
  - exchange of information between processes on the same computer or on different systems tied together by a network.
  - Implemented via *shared memory* or *message passing*.
- Error detection
  - ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

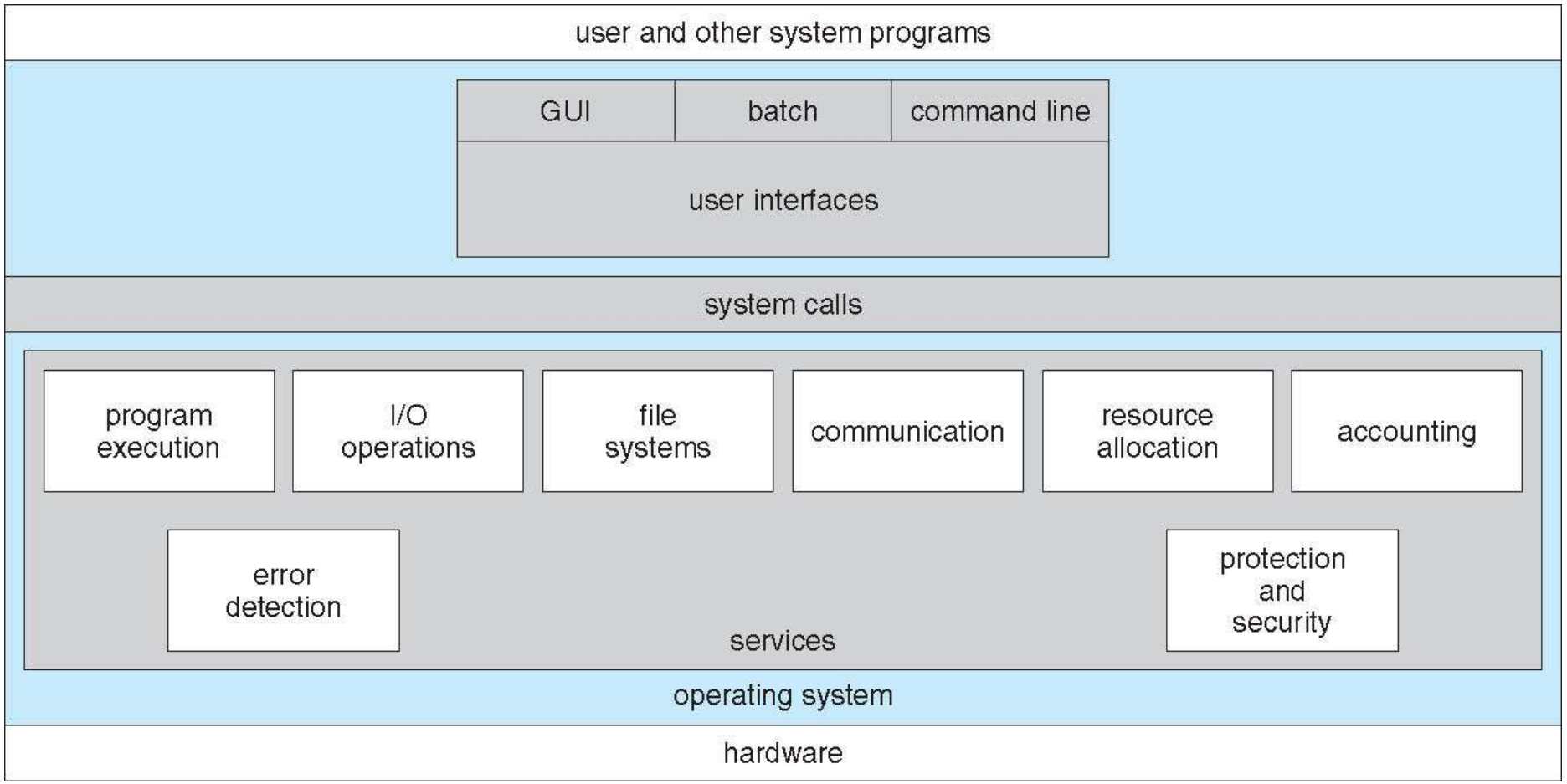
Operating System Services



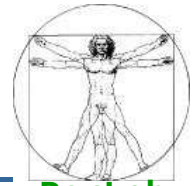
Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- **Resource allocation**
  - 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
- **Protection and security**
  - **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 OS Services



User Interface

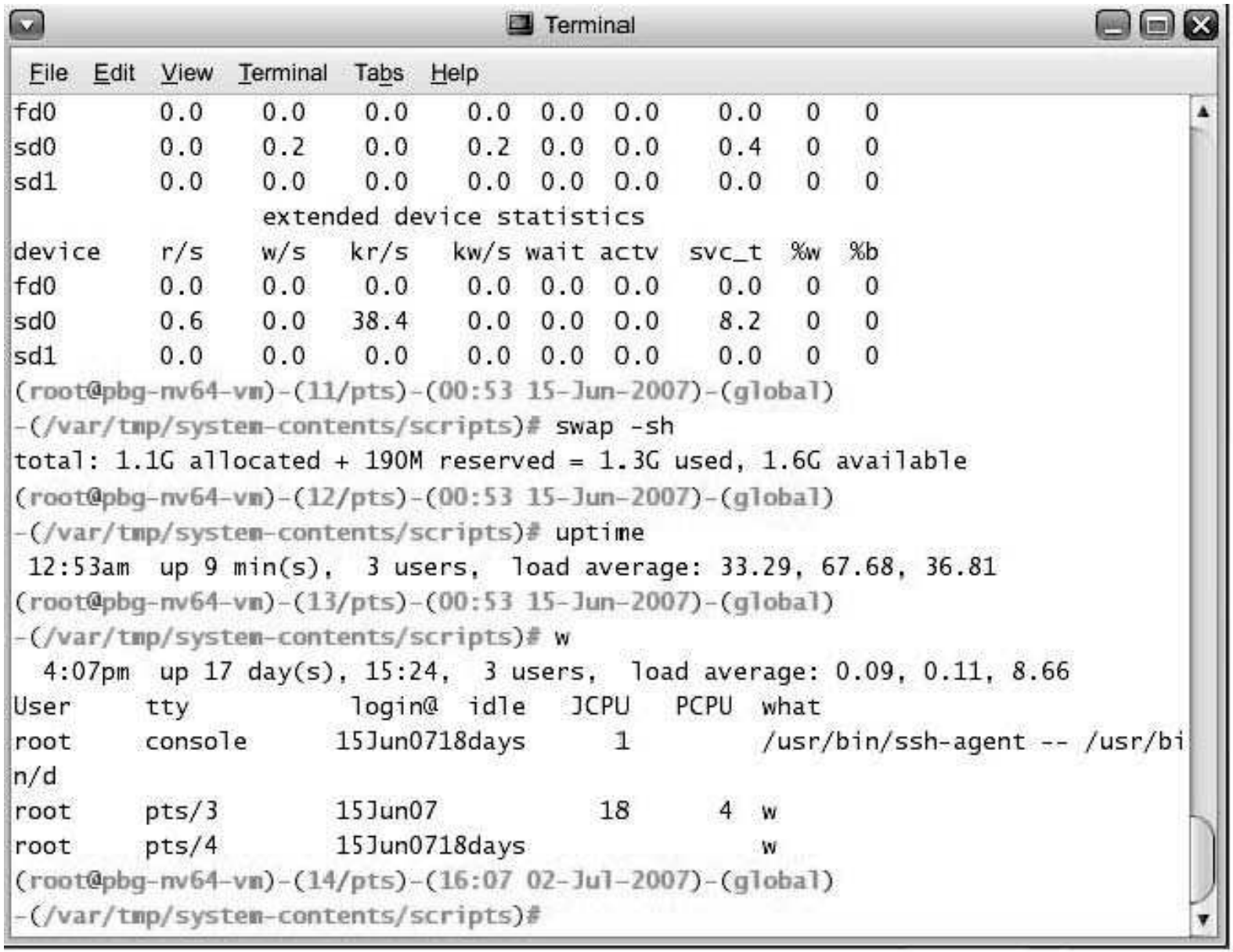


- Command Line Interface (CLI) – Command Interpreter
  - Sometimes implemented in kernel, sometimes by systems program
    - Sometimes multiple flavors implemented – shells
  - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
      - If the latter, adding new features doesn't require shell modification

Graphic User Interface - GUI

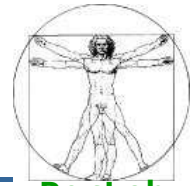
- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC
- 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



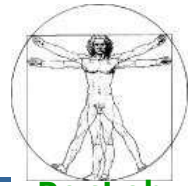
The Mac OS X GUI

Overview



- Preliminary Concepts
- Services
- **System Calls**
- System Programs
- Internal Structure
- System Boot

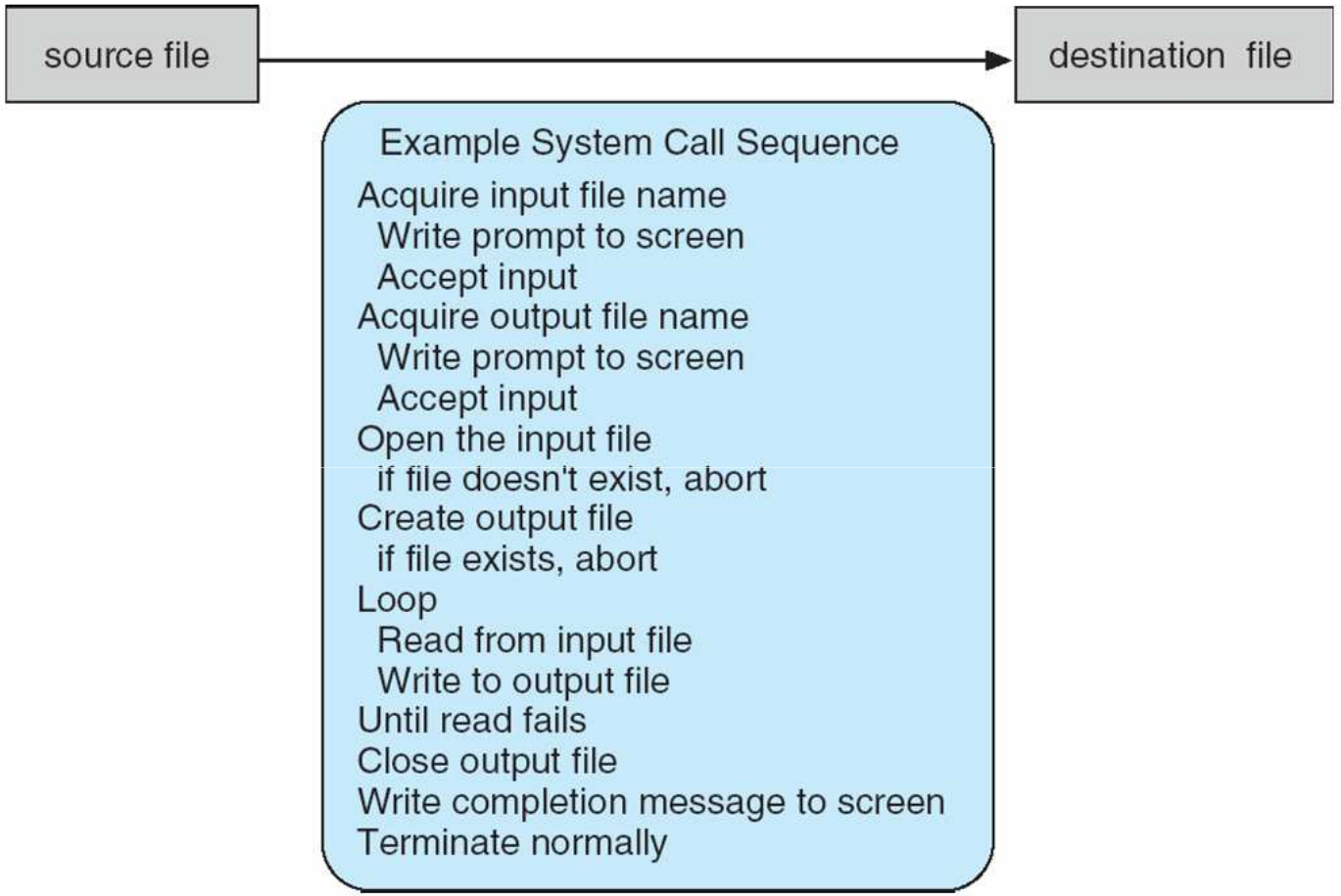
System Calls



- 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
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?
  - Portability
  - Usability (API functions are typically easier to use than system calls)

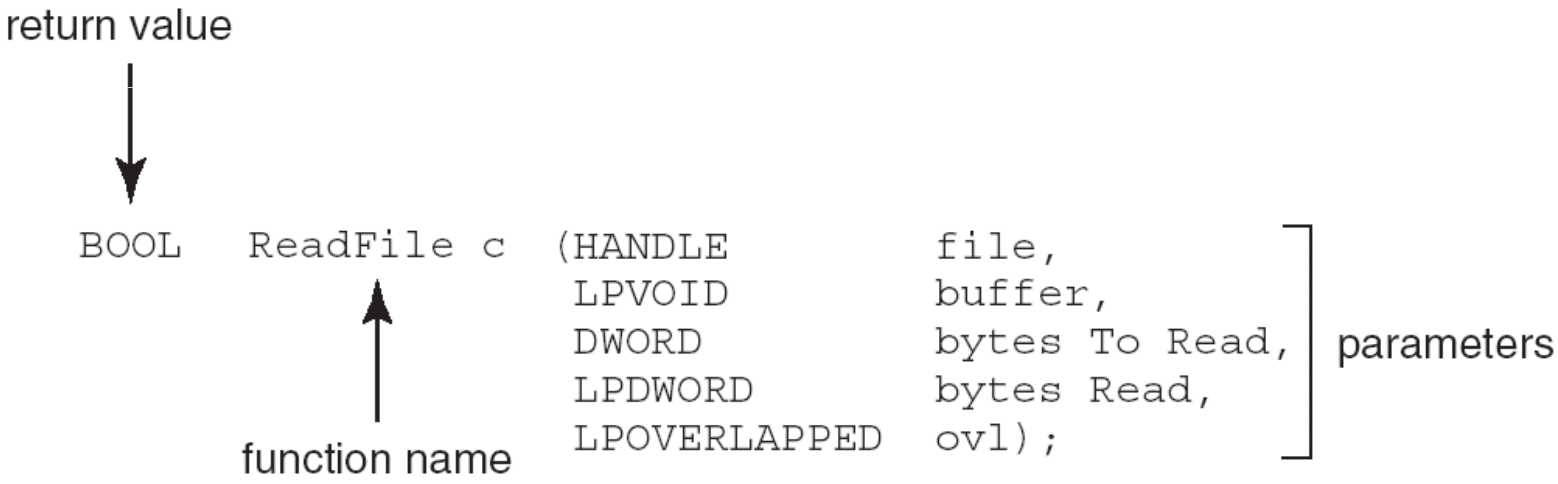
Example of System Calls

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



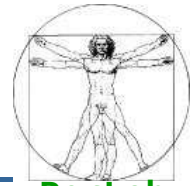
Example of Standard API

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



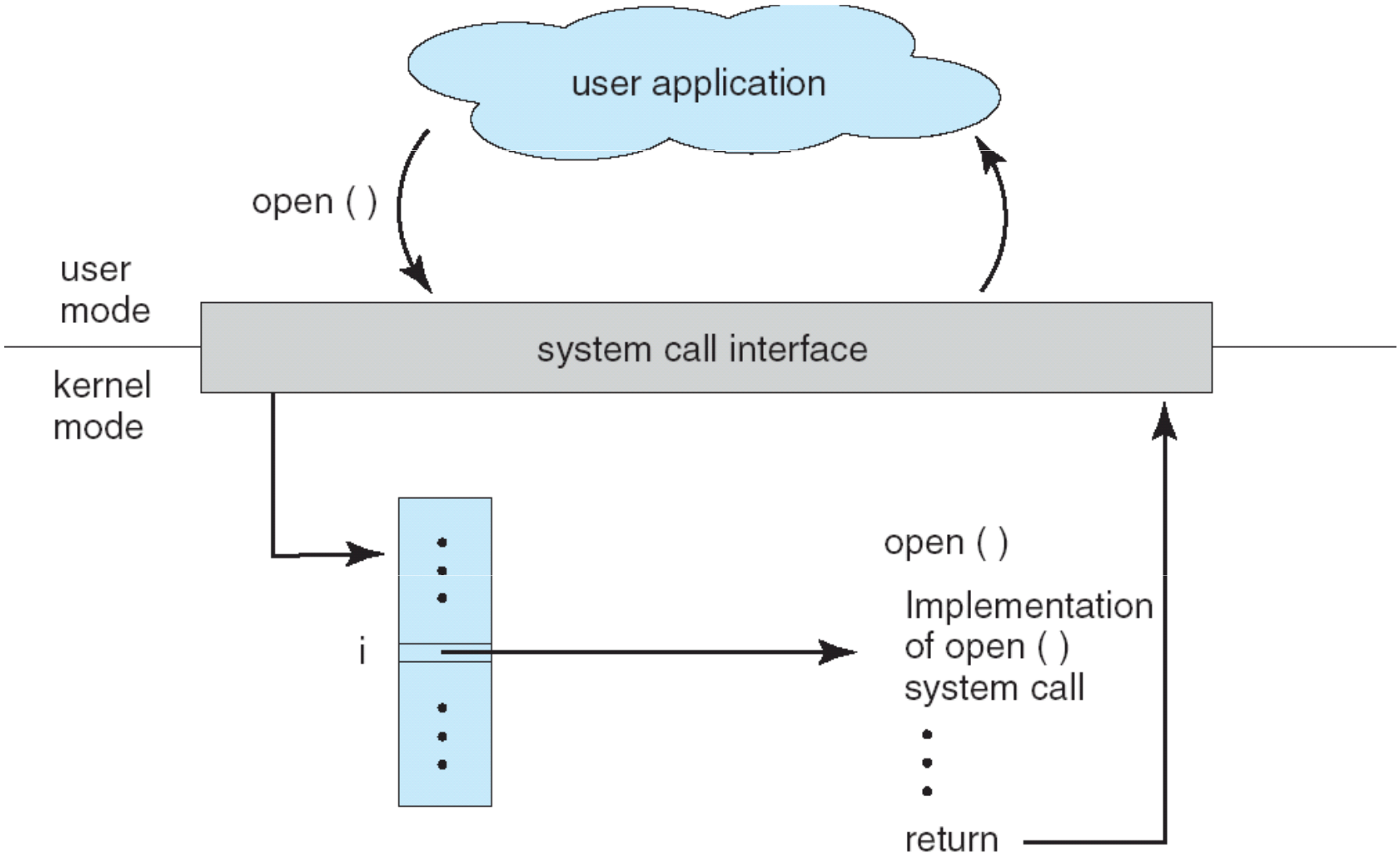
- Description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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

System Call Implementation

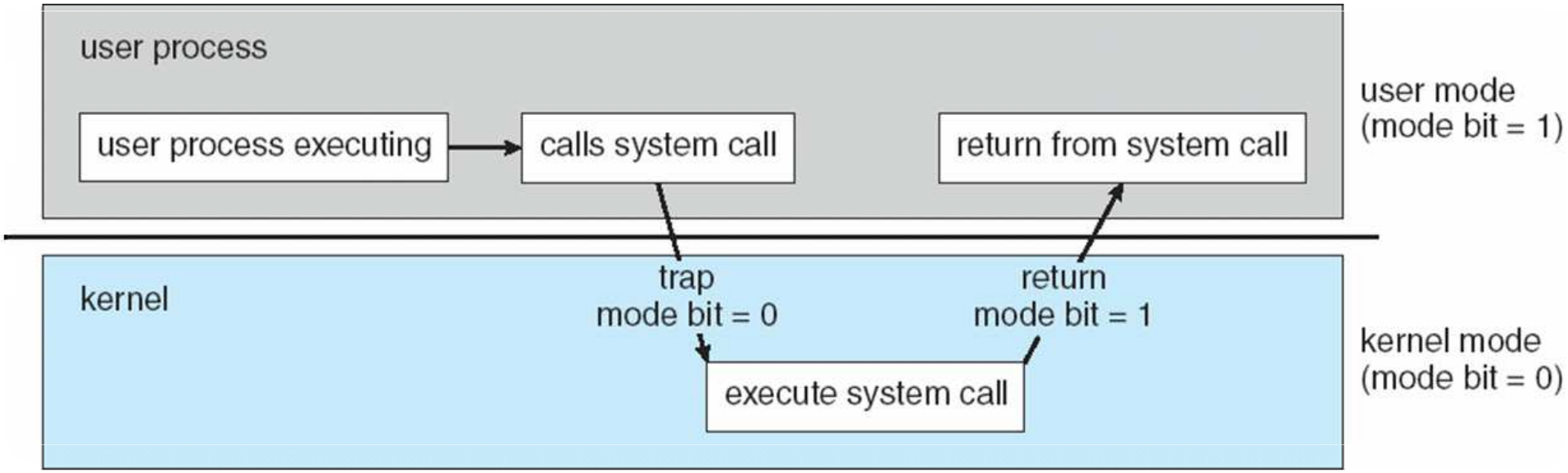


- Typically, a number associated with each system call
  - The compiler maintains a table of system calls
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller needs know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface are 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

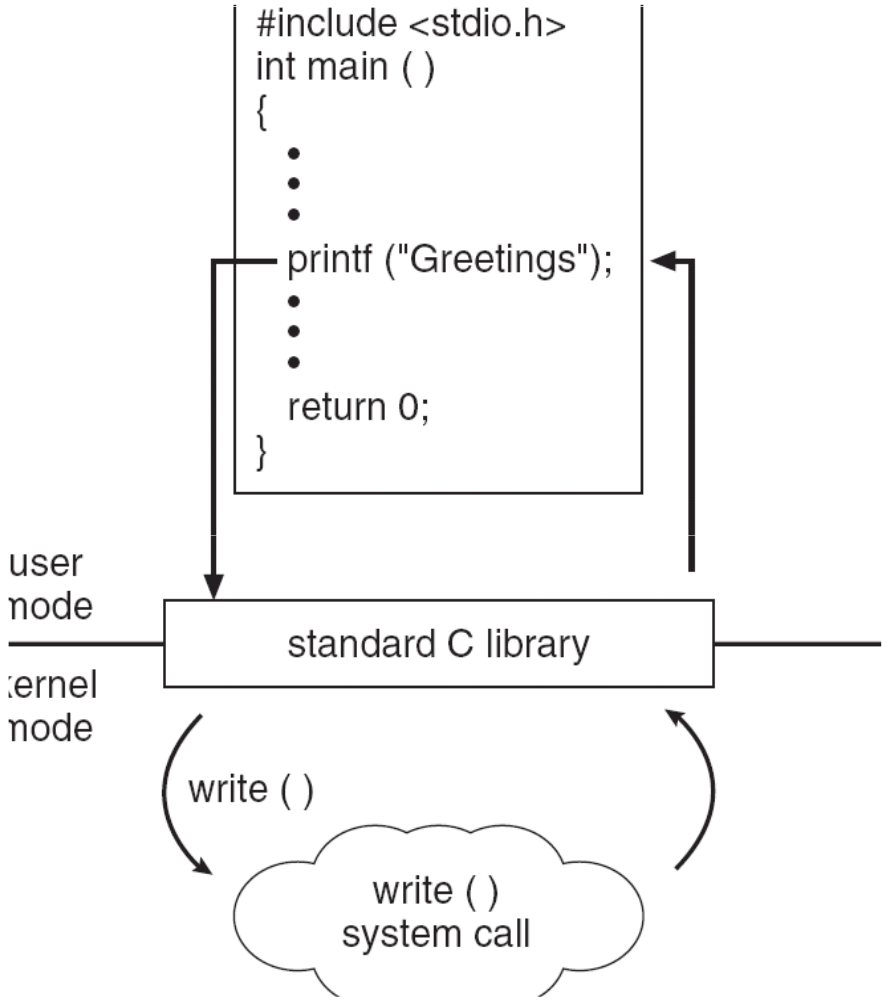


Transition from User to Kernel Mode

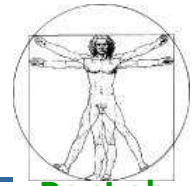


Standard C Library Example

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

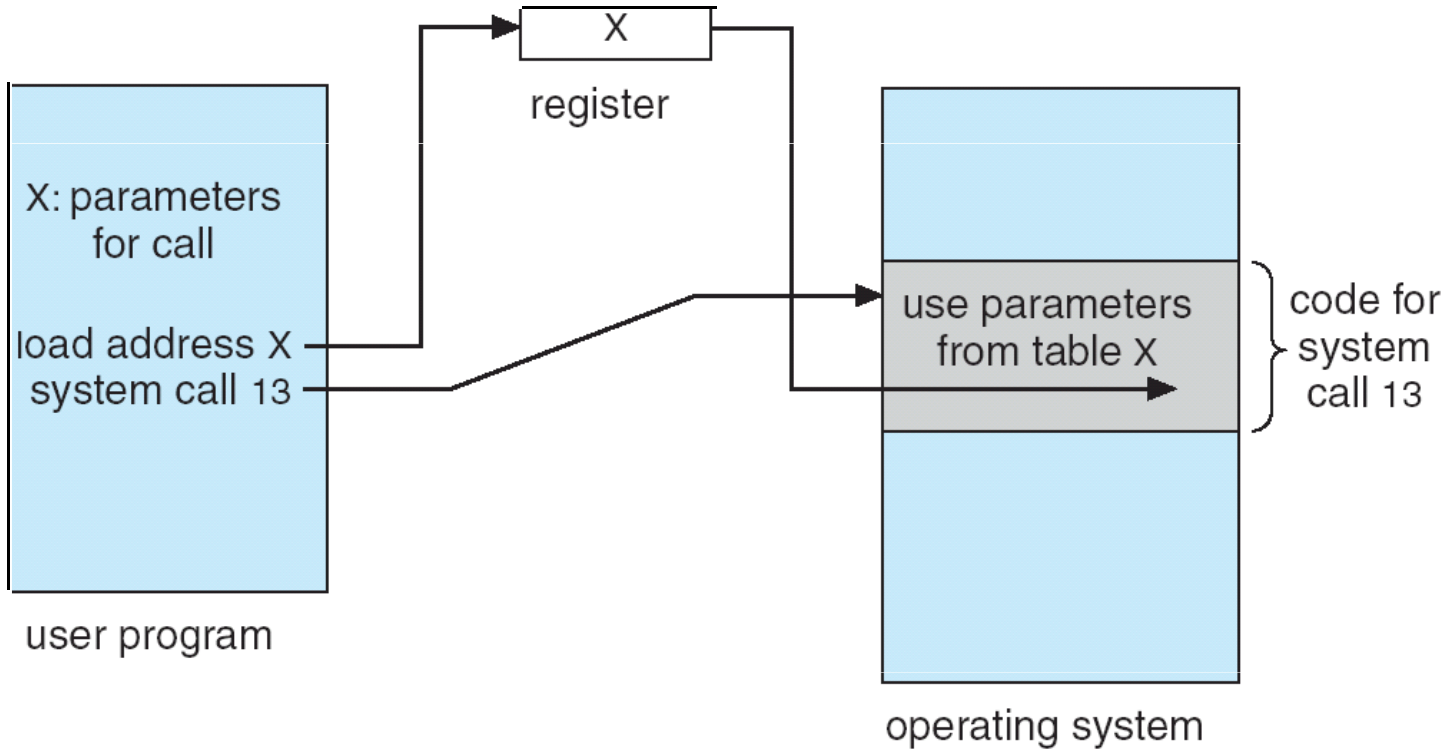


System Call Parameter Passing

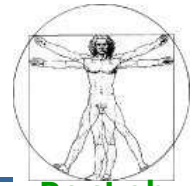


- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters 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



Types of System Calls



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

Process control

- create_process()
- end(), abort()
- load()
- execute()
- get_process_attribute(), set_process_attribute()
- wait(time), wait(event)
- signal(event)

File Manipulation

- create_file(), delete_process()
- open(), close()
- read(), write(),
- get_file_attributes(), set_file_attributes()

Device Management

- request_device(), release_device()
- read(), write()
- get_device_attribute(), set_device_attribute()
- logical_attach_device(), logical_detach_device()

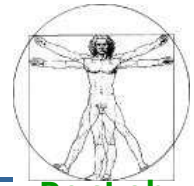
System Calls (5)

- get_time(), get_date()
- set_time(), set_date()
- get_process_attribute(), set_process_attribute()
- get_file_attribute(), set_file_attribute()
- get_device_attribute(), set_device_attribute()

Communications

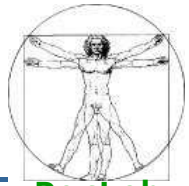
- create_connection(),
- delete_connection()
- send(msg), receive(msg)

Examples of Windows/Unix System Calls



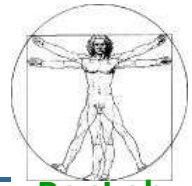
	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information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

Overview



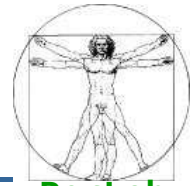
- Preliminary Concepts
- Services
- System Calls
- **System Programs**
- Internal Structure
- System Boot

System Programs



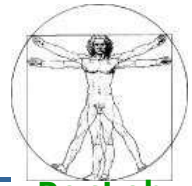
- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most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

System Programs



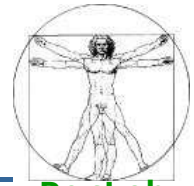
- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management
  - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - 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

System Programs (cont'd)



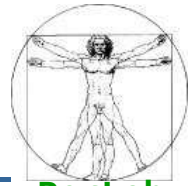
- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support
  - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution
  - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications
  - 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

Overview



- Preliminary Concepts
- Services
- System Calls
- System Programs
- **Internal Structure**
- System Boot

Common System Components

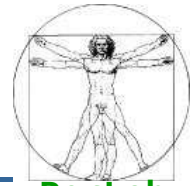


- User Interface
- Process Manager
- Memory Manager
- File Manager
- I/O System Manager
- Secondary Memory Manager
- Networking
- Protection System

Process Manager

- The process manager is responsible for the following activities
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication

Memory Manager



- Memory is a large array of words or bytes, each with its own address.
- It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device.
- The memory manager is responsible for the following activities
  - Keep track of which parts of memory are currently being used and by whom.
  - Allocate and deallocate memory space as needed.

File Manager

- A file is a collection of related information defined by its creator.
- The file manager is responsible for the following activities:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

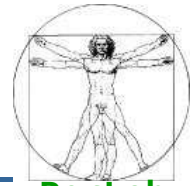
I/O System Manager

- The I/O system consists of:
  - A general device-driver interface
  - Drivers for specific hardware devices
  - A buffer-caching system

Secondary-Storage Manager

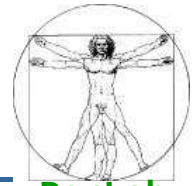
- Main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently
- The computer system must provide *secondary storage* as a permanent storage system.
  - Typically Disks
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

Networking (Distributed Systems)



- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol.*
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Protection System



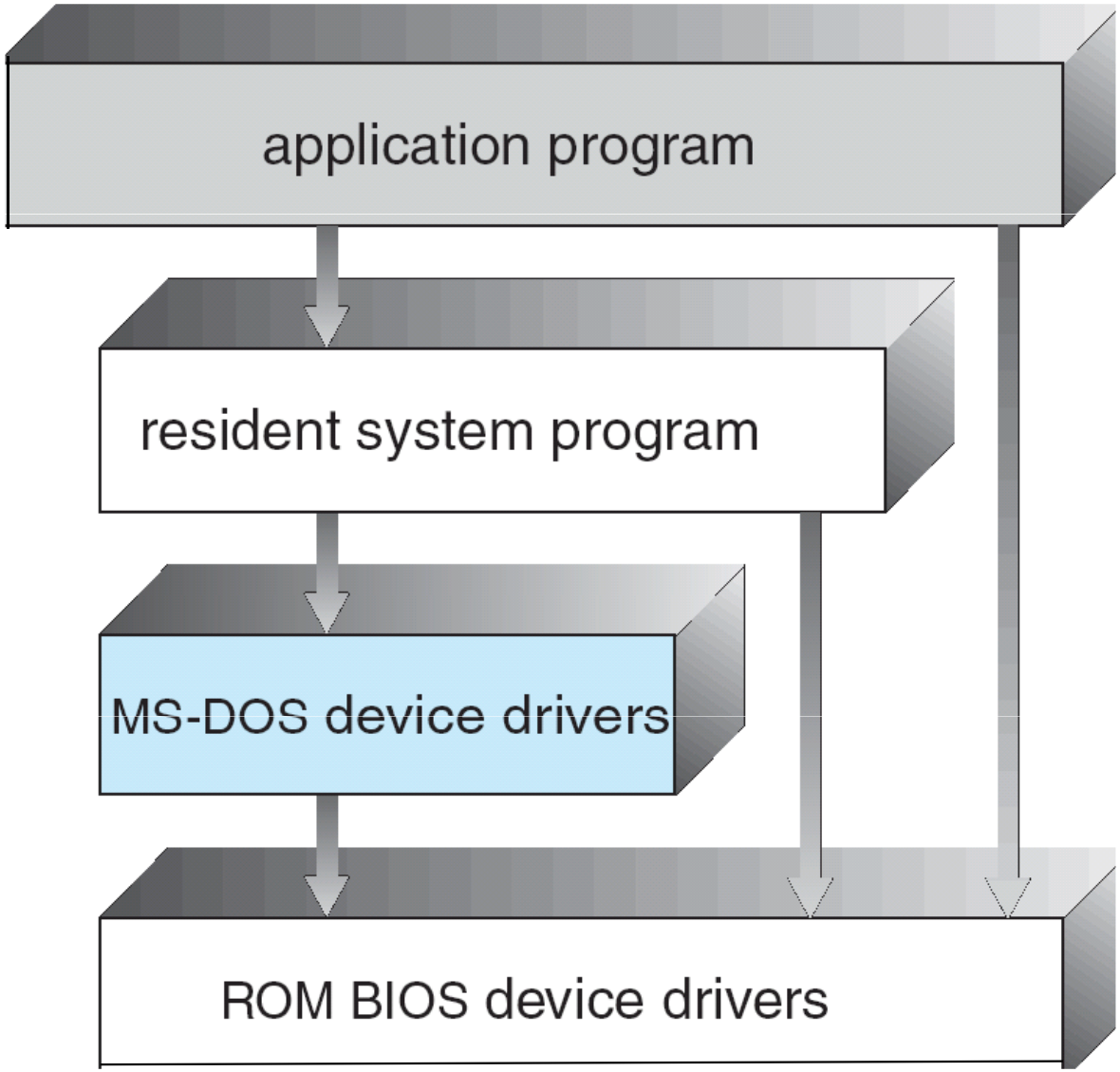
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.

- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

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

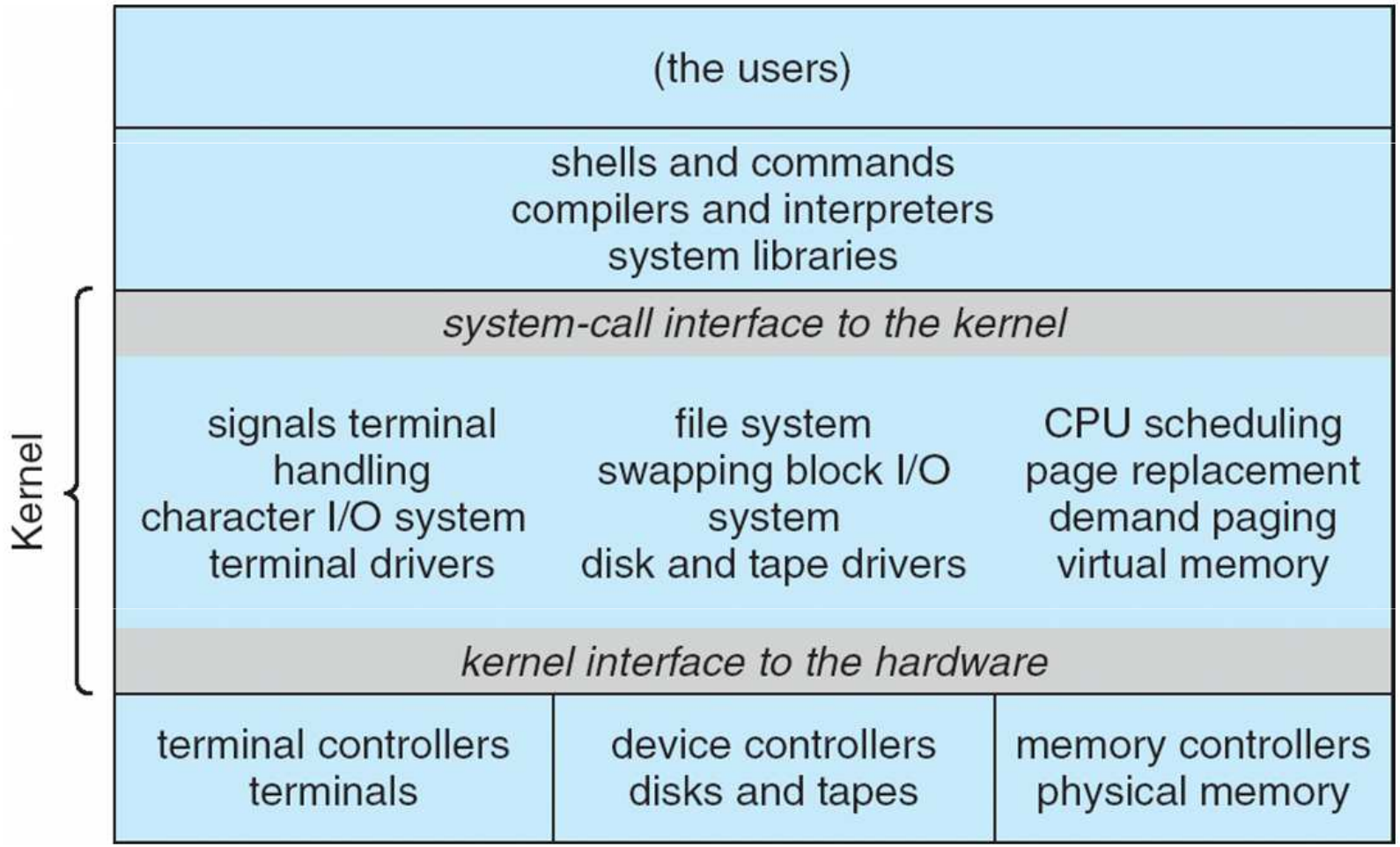
MS-DOS Layer Structure



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.
- Each layer uses functions (operations) and services of only lower-level layers

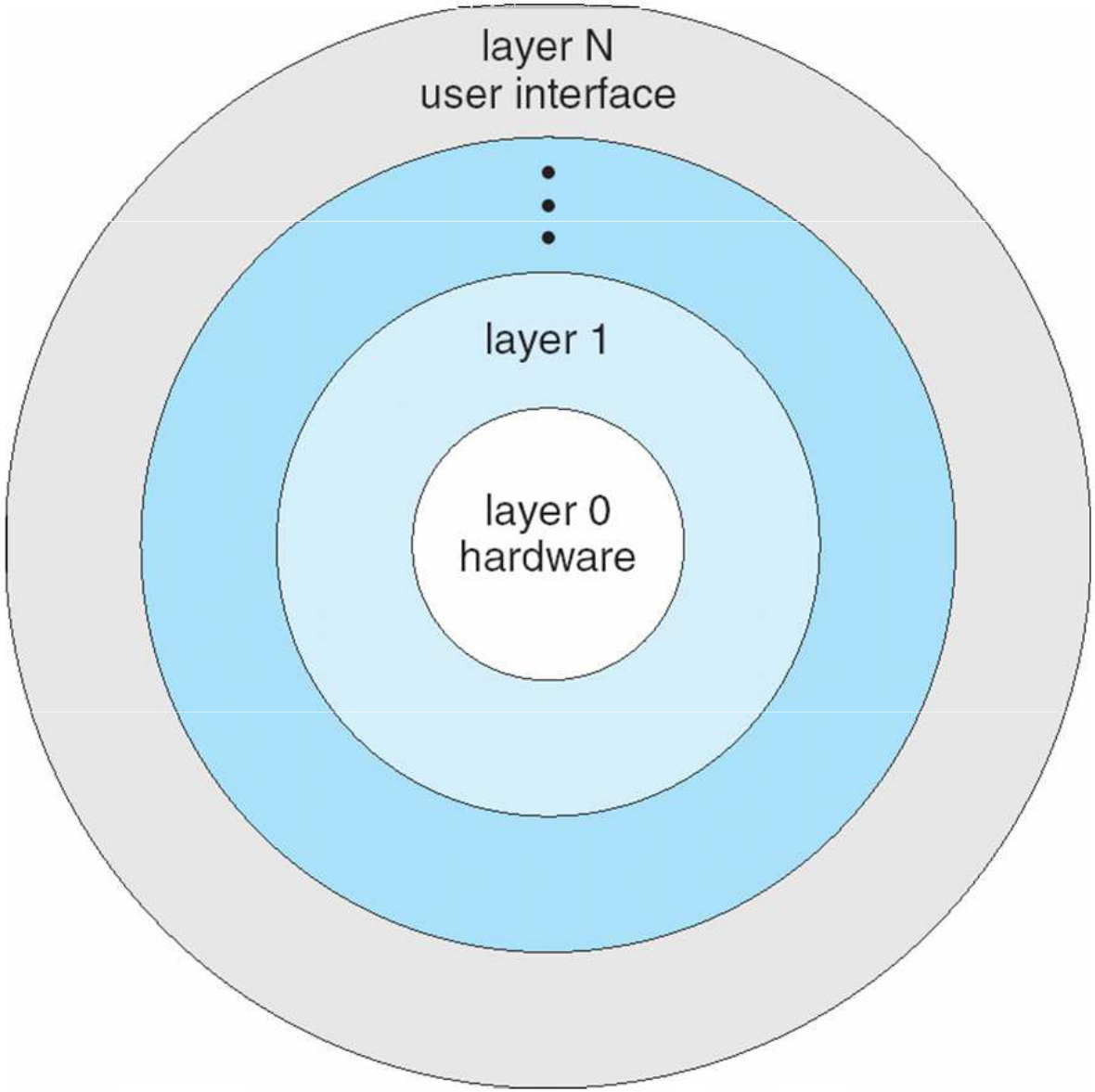
Traditional UNIX System Structure



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



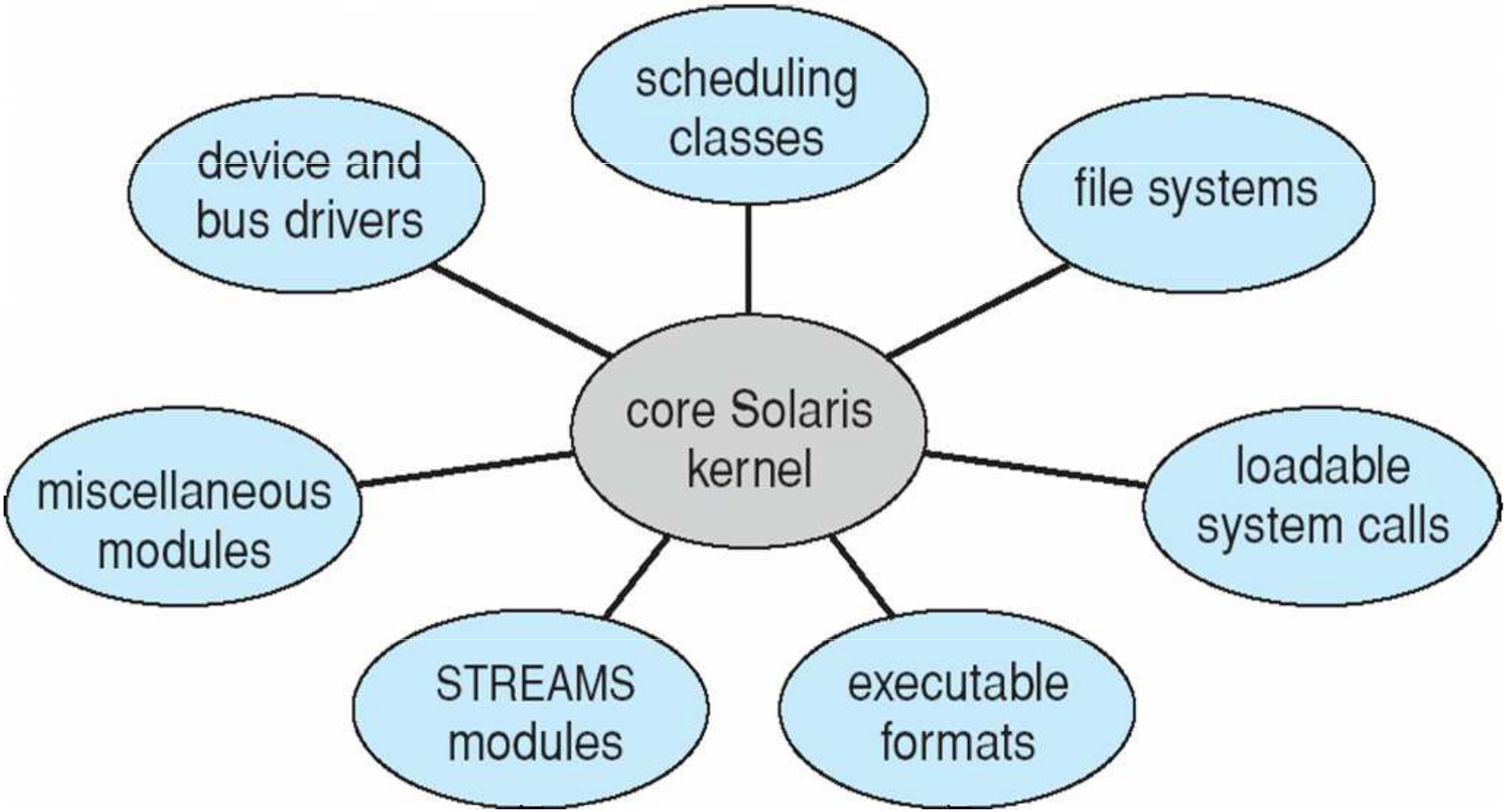
Microkernel System Structure

- 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

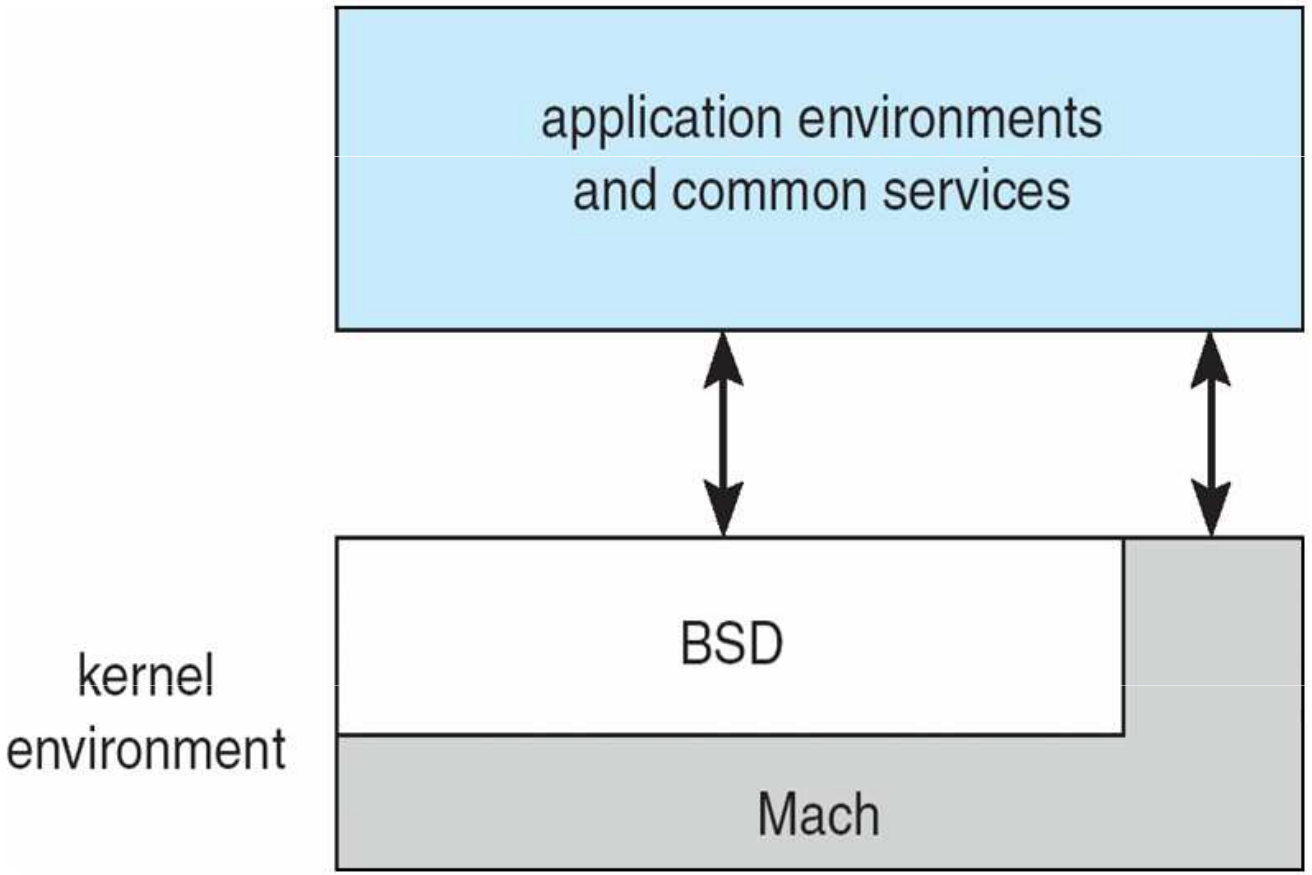
Modules

- 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 flexibility

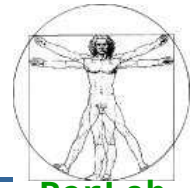
Solaris Modular Approach



Mac OS X Structure

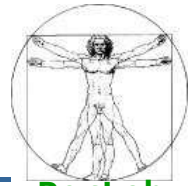


Mac OS X Structure (cont'd)



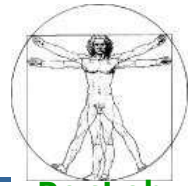
- Mach Micro-kernel is responsible for
  - CPU scheduling
  - Memory management
  - Inter-process communication (IPC)
  - Remote procedure calls (RPC)
- BSD kernel provides
  - CLI User Interface
  - File manipulation services
  - Networking services
  - POSIX API (including Pthreads)

Overview



- Preliminary Concepts
- Services
- System Calls
- System Programs
- Internal Structure
- **System Boot**

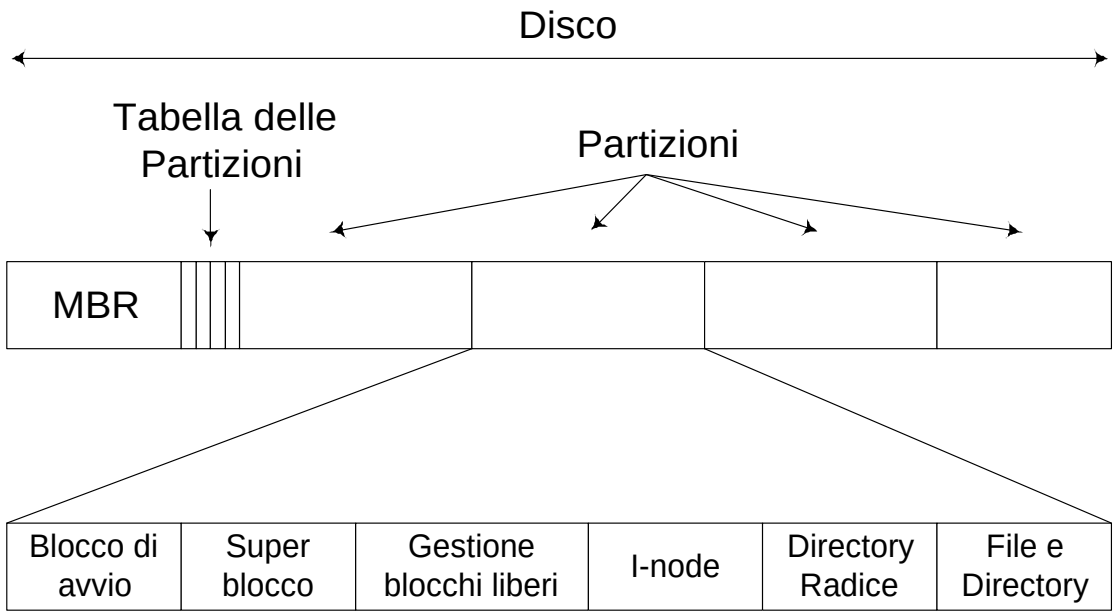
System Boot



- Operating system must be made available to hardware so hardware can start it
- **Bootstrap loader**
  - locates the kernel, loads it into memory, and starts it
  - When power initialized on system, execution starts at a fixed memory location
    - firmware used to store the initial boot code (ROM does not need to be initialized and is immune to viruses)
    - Small computers store the entire operating system in ROM (EPROM)
- Two-Step bootstrap
  - **boot block** at fixed location loads bootstrap loader

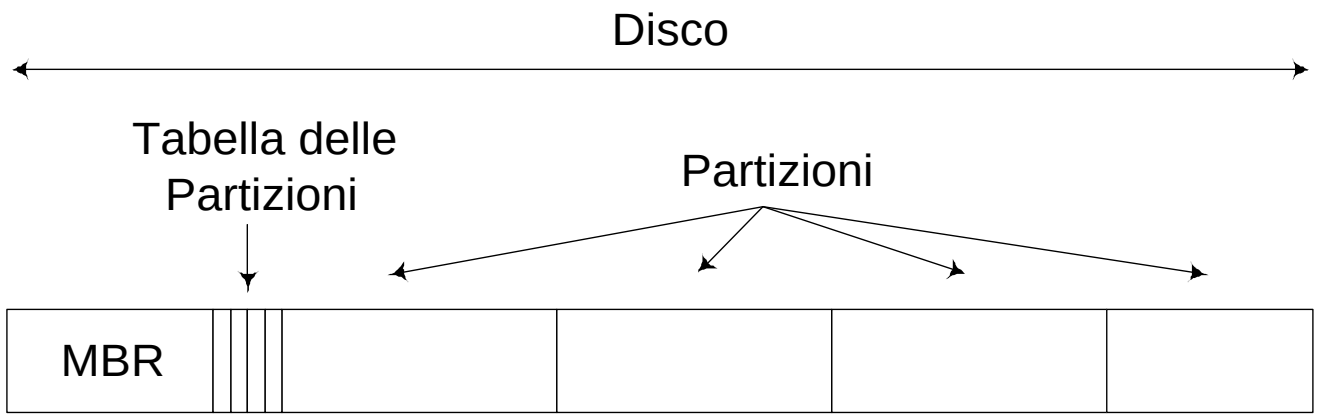
Disk Organization

- Il disco può essere suddiviso in partizioni ognuna contenente un proprio file system
- Il partizionamento del disco avviene mediante la formattazione di alto livello



Disk Organization

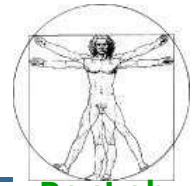
- MBR (Master Boot Record)
  - Contiene programma di avvio
  - La fine del MBR contiene la tabella delle partizioni
- Tabella delle partizioni
  - Contiene punto di inizio e fine di ogni partizione
  - Una sola partizione è marcata come attiva
  - E' la partizione da cui verrà caricato il SO



Disk Organization

- Blocco di avvio
  - Contiene semplice eseguito in fase di bootstrap e serve a caricare il kernel
  - Ogni partizione contiene il Blocco di Avvio anche se non contiene il SO (potrebbe contenerne uno)
- Superblocco
  - Contiene informazioni sul file system
    - Numero magico che identifica il FS
    - Numero di blocchi del FS
    - …
- Gestione per lo spazio libero
  - Strutture dati per la gestione dei blocchi liberi
- I-node
  - Nei SO che utilizzano gli i-node questi sono raggruppati in unaparte del disco
- Directory radice
- File e directory

Two-Step Bootstrap (Windows 2000)



- Esecuzione del programma di avvio in ROM
  - Diagnosi
  - Caricamento del MBR
- Esecuzione del codice di avvio contenuto nel MBR
  - Localizza la partizione attiva dalla tabella delle partizioni
  - Legge il primo blocco (blocco di avvio) e lo esegue
- Esecuzione del codice nel Blocco di Avvio
  - Localizza il kernel nella partizione attiva
  - Carica in memoria il kernel
  - Cede il controllo al kernel

Questions?