BITS, PILANI – K. K. BIRLA GOA CAMPUS

Operating Systems

by

Mrs. Shubhangi Gawali

Dept. of CS and IS

What is an Operating System?

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

OS runs all the time in the system.

Main objectives/goals of an OS

- Convenience
- Efficiency
- Ability to evolve

What if no Operating System?

- We need a mechanism to
  - Load the program into memory
  - Run the program in processor
  - Store the result in persistent storage and
  - Unload the program to release memory [for the next program to use]

Role of an OS

- OS as a Resource manager
- Resources:
  - Processor cycles
  - Main memory
  - Secondary memory
  - i/o devices
  - File system

Services provided by an OS

- Process management activities
- Memory management activities
- Files management activities
- Protection and security

Process management activities

- Program development and execution
- Multitasking
- Inter-process communication
- deadlock handling

Memory Management Activities

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes and data to move into and out of memory
- Allocating and deallocating memory space as needed

File Management Activities

- Creating and deleting files and directories
- Support primitives to manipulate files and directories
- Mapping files onto secondary storage
- Backup files onto stable (non-volatile) storage media

Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS.
- Security – defense of the system against internal and external attacks.
- User identities (user IDs, security IDs) include name and associated number, one per user.
- Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file

Computer Startup

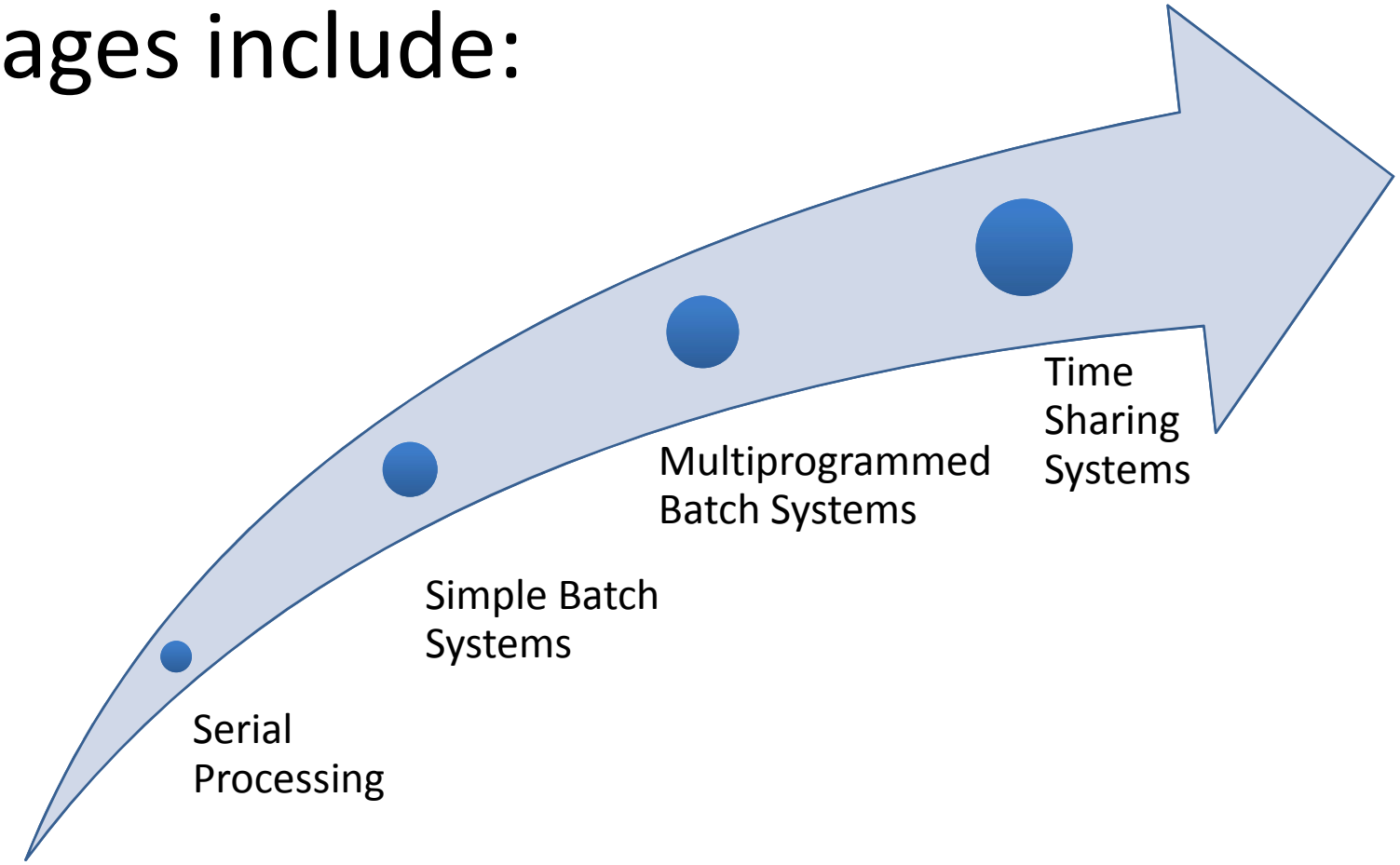
- **bootstrap program** is loaded at power-up or reboot
- Typically stored in ROM or EPROM, generally known as firmware
- Initializes all aspects of system
- Loads operating system in MM and starts execution
- Interrupt driven

Evolution of Operating Systems

- Reasons
  - Hardware upgrades
  - New types of hardware
  - New services
  - Fixes

Evolution of Operating Systems

- Stages include:



Serial Processing

Earliest Computers:

- No operating system
  - programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in “series”

Problems:

- Scheduling:
  - most installations used a hardcopy sign-up sheet to reserve computer time
    - time allocations could run short or long, resulting in wasted computer time
- Setup time :
  - aconsiderable amount of time was spent just on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization
- Monitor
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them together and places them on an input device
  - program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

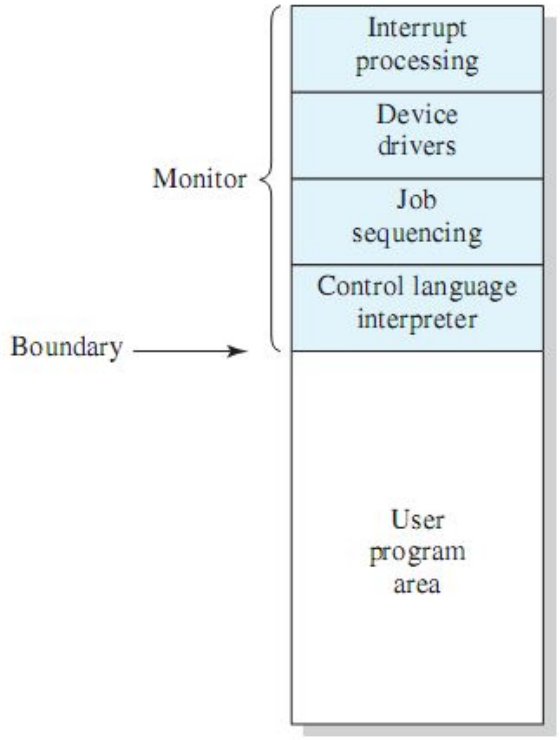


Figure 2.3 **Memory Layout for a Resident Monitor**

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "*control is passed to a job*" means processor is fetching and executing instructions in a user program
- "*control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Desirable hardware features

- Memory protection
- Timer
- Privileged instructions
- Interrupts

Modes of Operation

User Mode

- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

Kernel Mode

- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor

  - Sacrifices:
    - some main memory is now given over to the monitor
    - some processor time is consumed by the monitor

  - Despite overhead, the simple batch system improves utilization of the computer

Multiprogrammed batch system

- Processor is often idle
  - even with automatic job sequencing
  - I/O devices are slow compared to processor

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Batch Multiprogramming vs. Time Sharing

- Maximize the processor utilization
- Minimize the response time

New problems for the OS

Timesharing and multiprogramming raised new problems for the OS.

- Protection of memory and file system
- Resource contention

Developments lead to modern OS

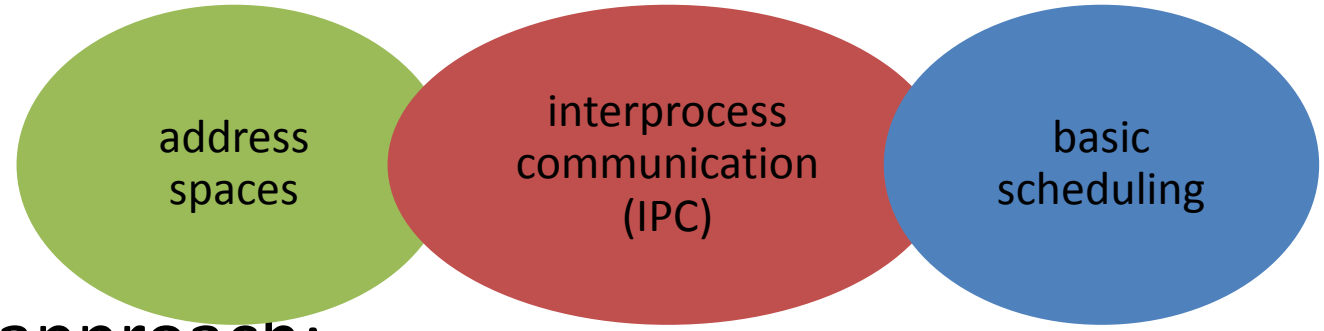
- Hardware drivers
  - Multiprocessor systems,
  - greatly increased processor speed,
  - high-speed network attachments, and
  - increasing size and variety of memory storage devices.
- Application arena:
  - multimedia applications,
  - Internet and Web access, and
  - client/server computing .
- Security:
  - sophisticated attacks such as viruses, worms, and hacking techniques, have had a profound impact on OS design.

Different Architectural Approaches

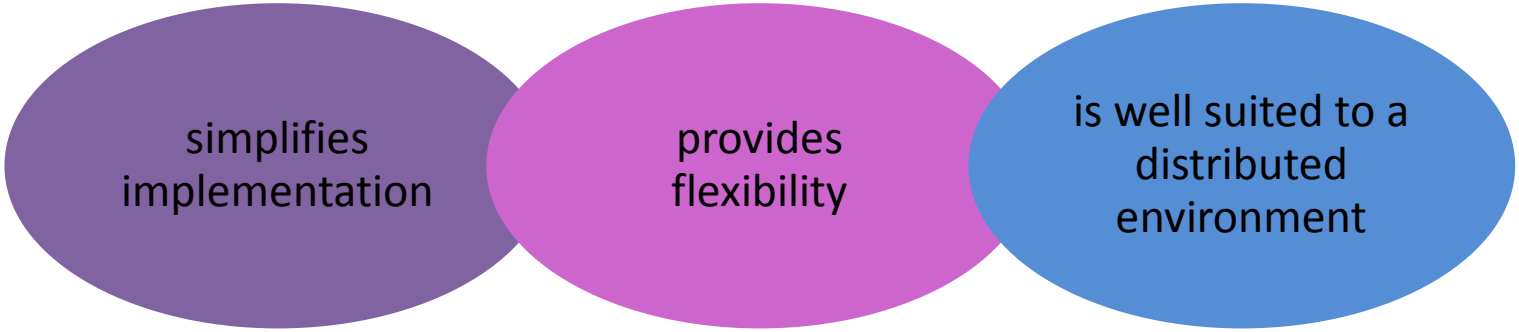
- Microkernel architecture
- Multithreading
- Symmetric multiprocessing
- Distributed operating systems
- Clustered system

Microkernel Architecture

- Assigns only a few essential functions to the kernel:



– The approach:



Multithreading

- Technique in which a process, executing an application, is divided into threads that can run concurrently

Thread

- dispatchable unit of work
- includes a processor context and its own data area to enable subroutine branching
- executes sequentially and is interruptible

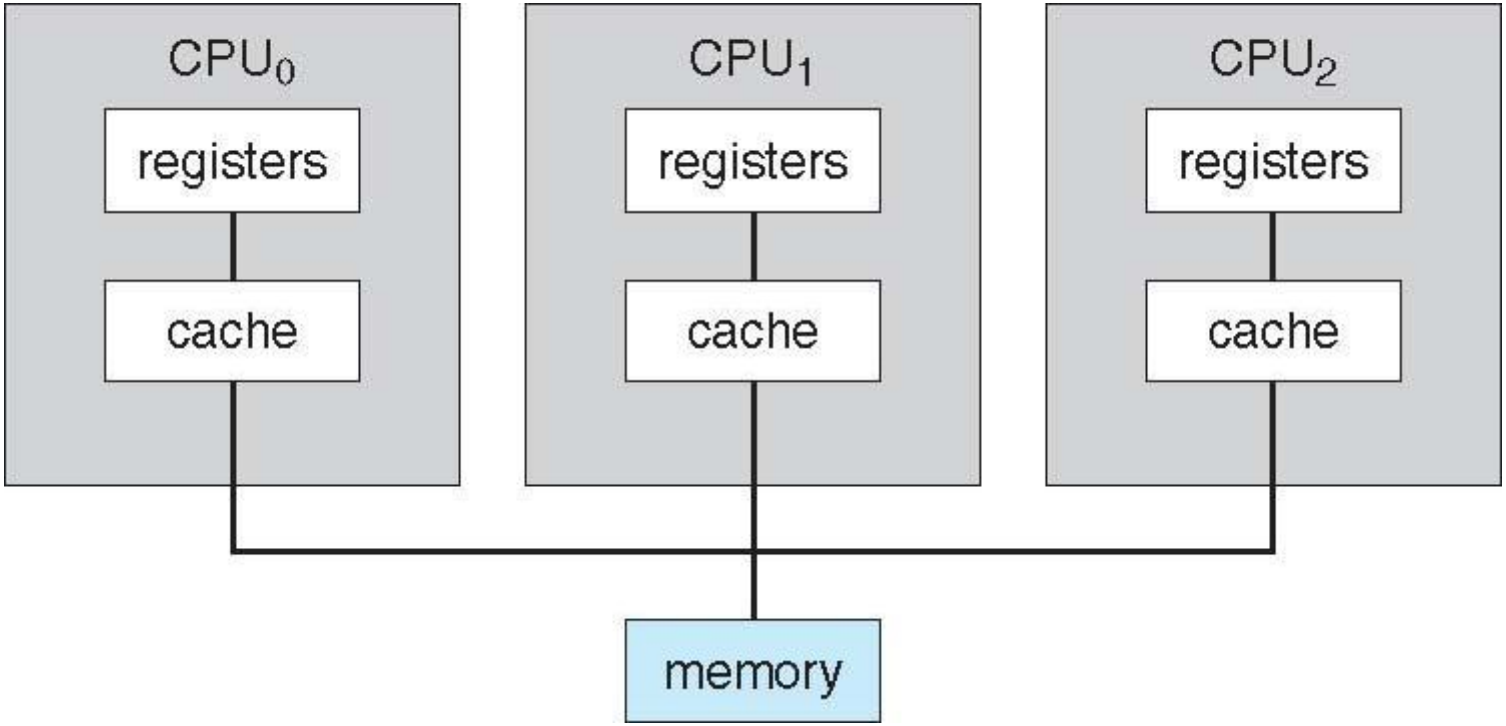
Process

- a collection of one or more threads and associated system resources
- programmer has greater control over the modularity of the application and the timing of application related events

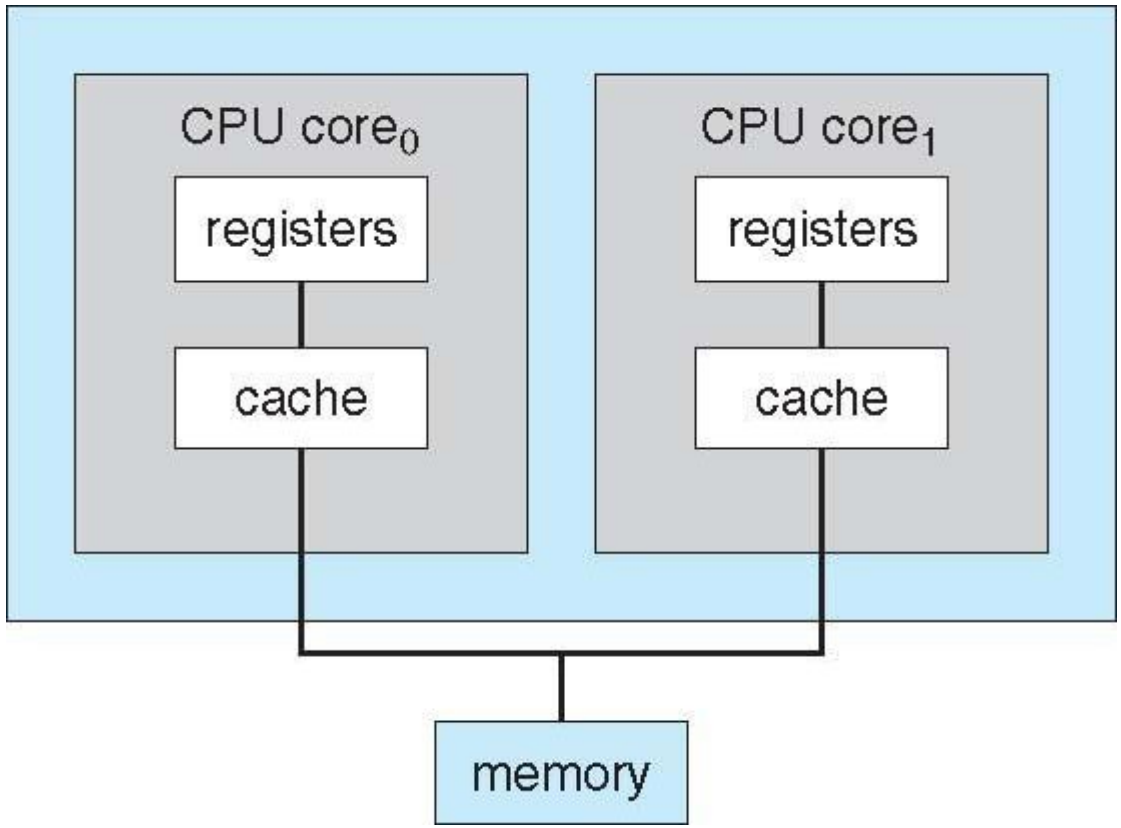
Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- Several processes can run in parallel
- Multiple processors are transparent to the user
    - these processors share same main memory and I/O facilities
    - all processors can perform the same functions
- The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors

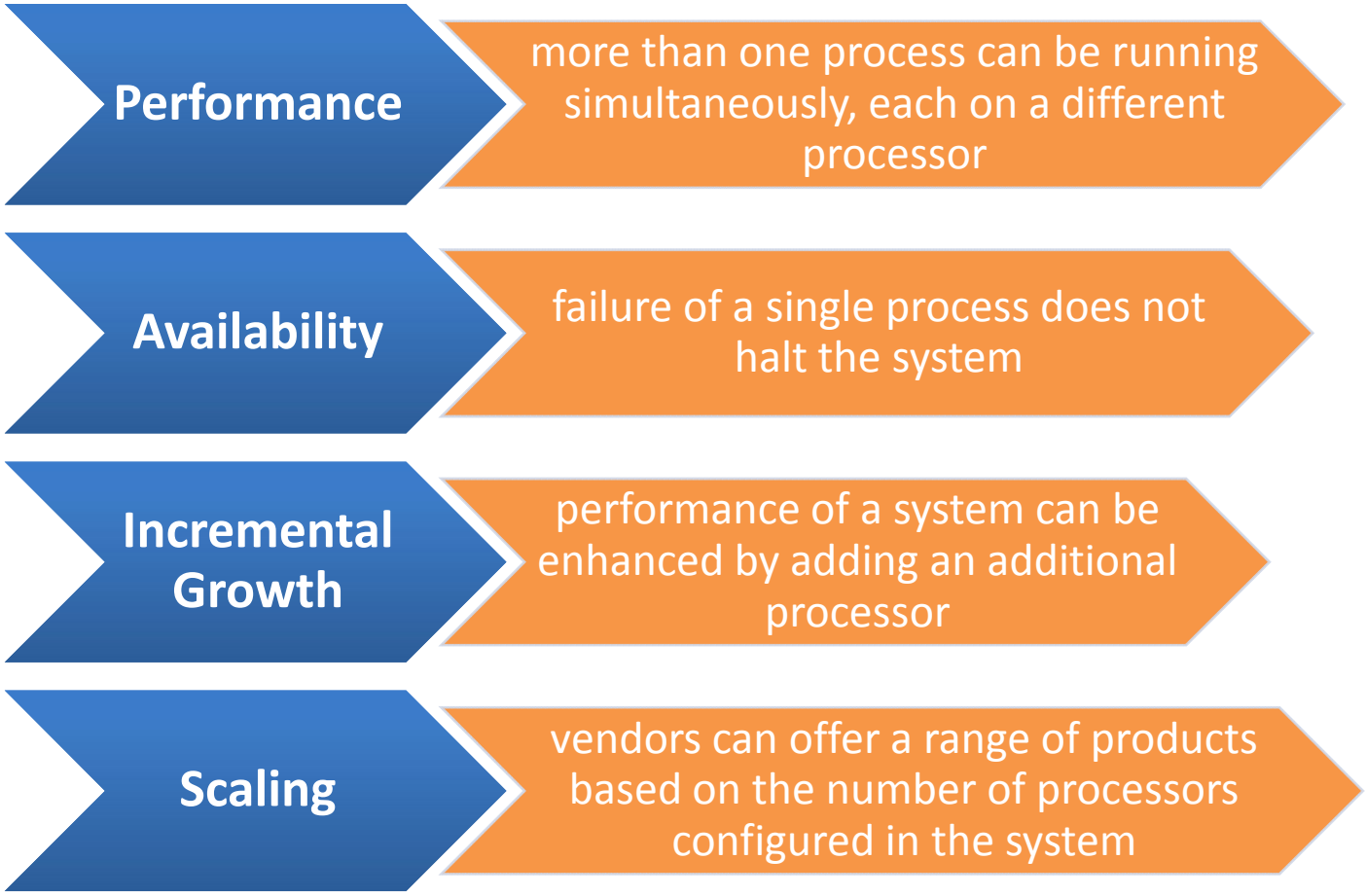
Symmetric Multiprocessing Architecture



A Dual-Core Design



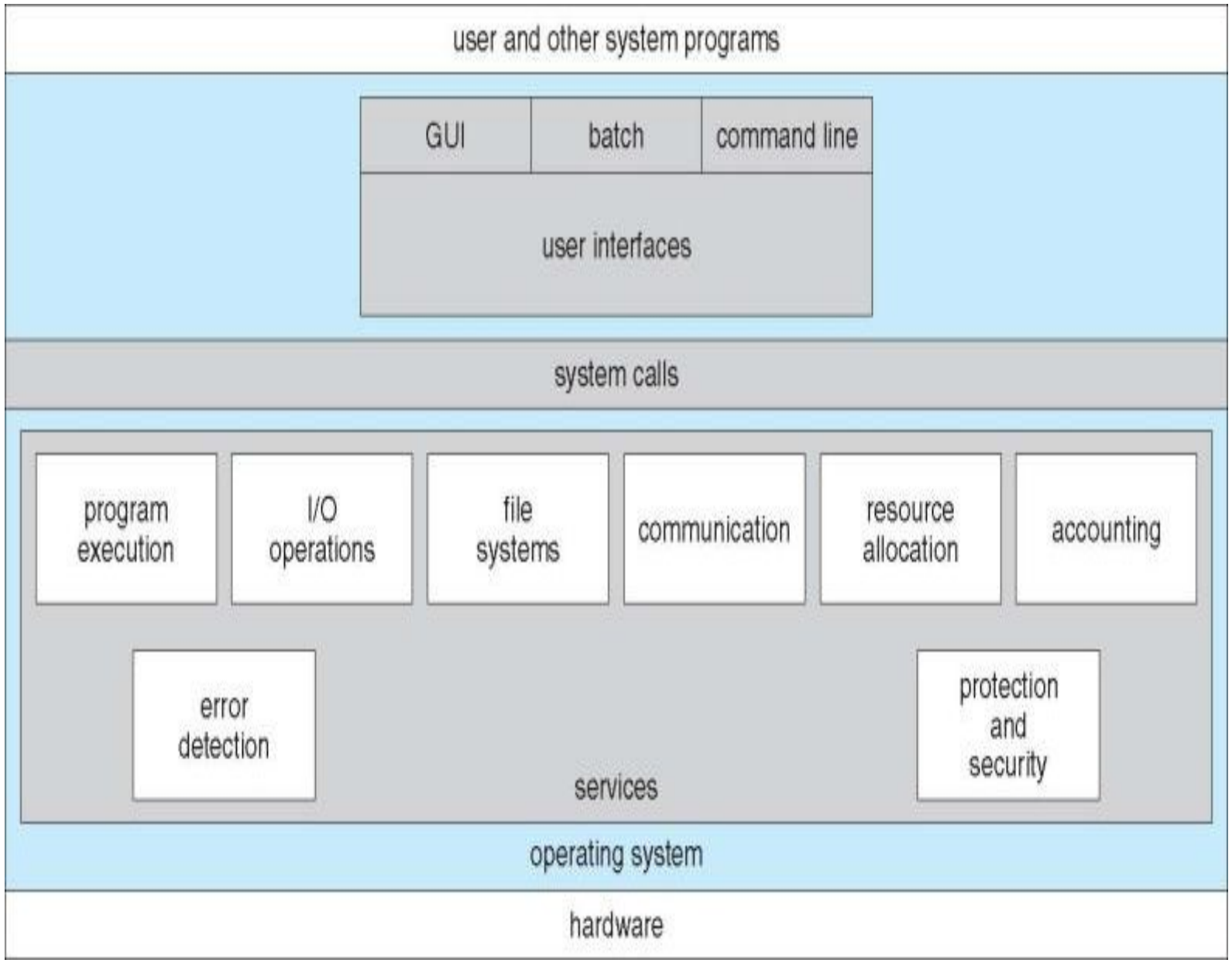
SMP Advantages



- Distributed Systems
  - Distribute the computation among several physical processors
  - *Loosely coupled system*
    - Each processor has its own local memory; Processors communicate with one another through various communications lines, such as high-speed buses or telephone lines
  - Client server and Peer to Peer system
  - Enables Parallelism but speed up is not the goal
  - Advantages
    - Resources Sharing, Computation speed up – load sharing, Reliability, Communications
- Clustered Systems
  - Usually performed to provide high availability
  - Asymmetric cluster: one machine will be in hot stand by mode
  - Symmetric cluster: 2 or more hosts are running applications and they are monitoring each other. Mode is more efficient

- Real-time Systems
  - Timeliness is equally important as produced results
  - Hard Real-time Systems
  - Soft Real-time Systems
  - Example: QNX, RTLINUX,VXWORKS
- Embedded Operating Systems
  - OS embed on the System itself
  - Fast, Application specific
  - Examples : Processor in washing machines, mobile phones
- Hand held Systems
  - Power consumption and weight must be low
  - Memory ranges from 512KB to 8MB
  - Speed of the processor can not be very high because of the power consumption

A View of Operating System Services



Operating System Services

- User interface - (UI)
  - 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
- 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.

Operating System Services (Cont)

- 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.
  - 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:
  - this 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 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
- 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.

User Operating System Interface -CLI

- 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
- Sometimes commands built-in, sometimes just names of programs
- If the latter, adding new features doesn’t require shell modification

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)

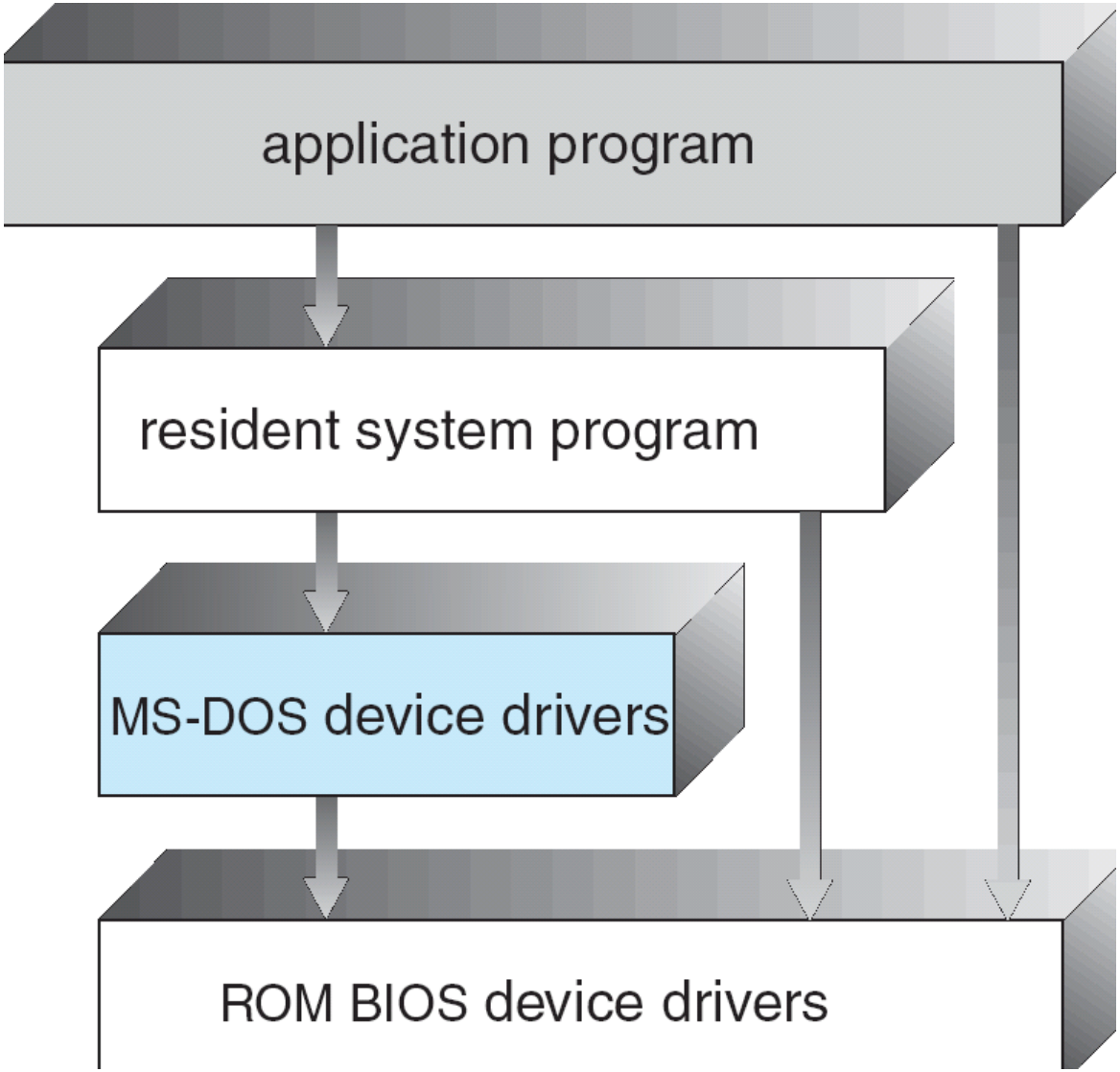
Many systems now include both CLI and GUI interfaces

- Microsoft Windows is GUI with CLI —command sh ll
- Appl Mac OS X as —”Aqua” GUI int rfac with UNIX k rn l underneath and shells available
- Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

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

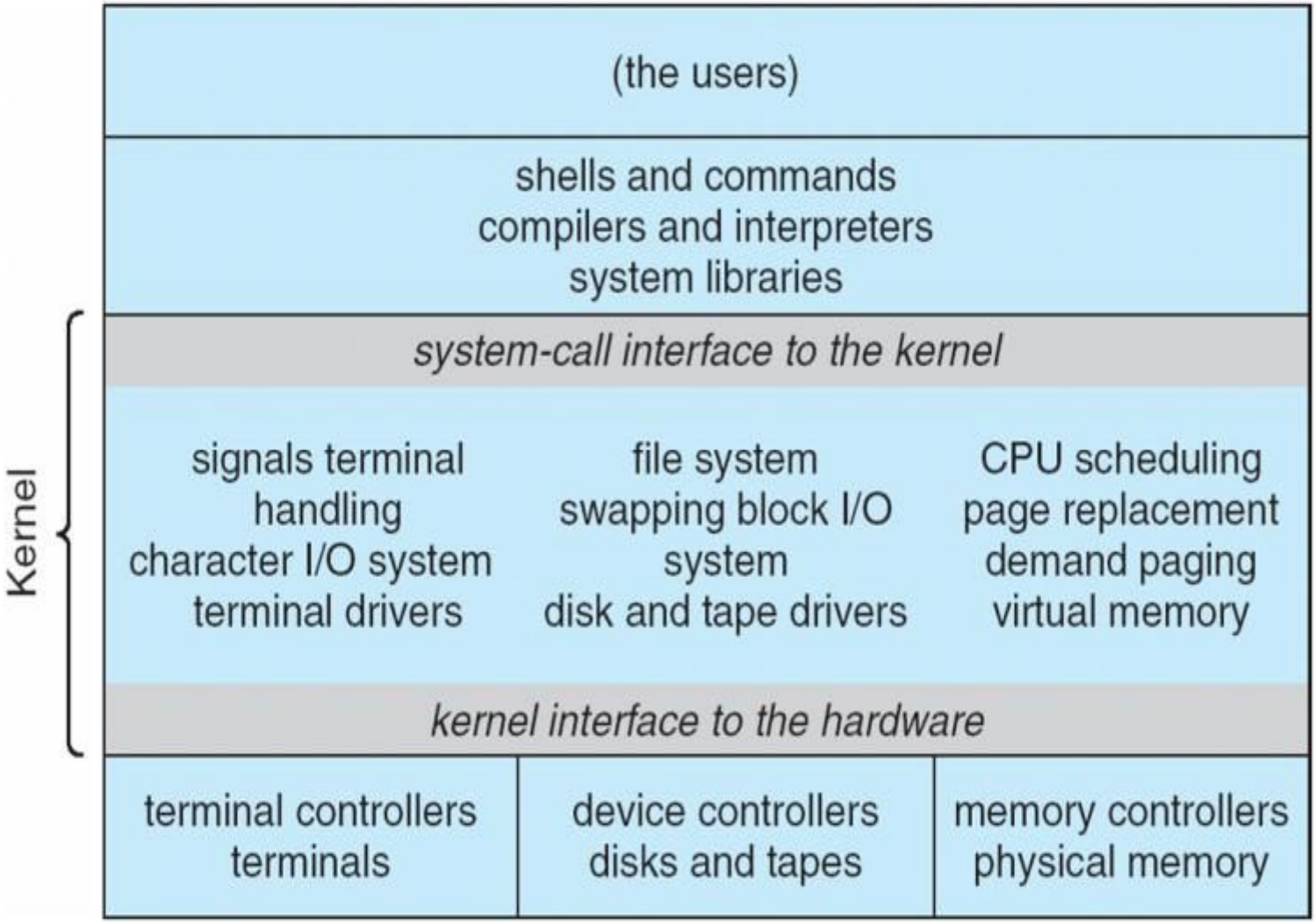


Layered Approach

- 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

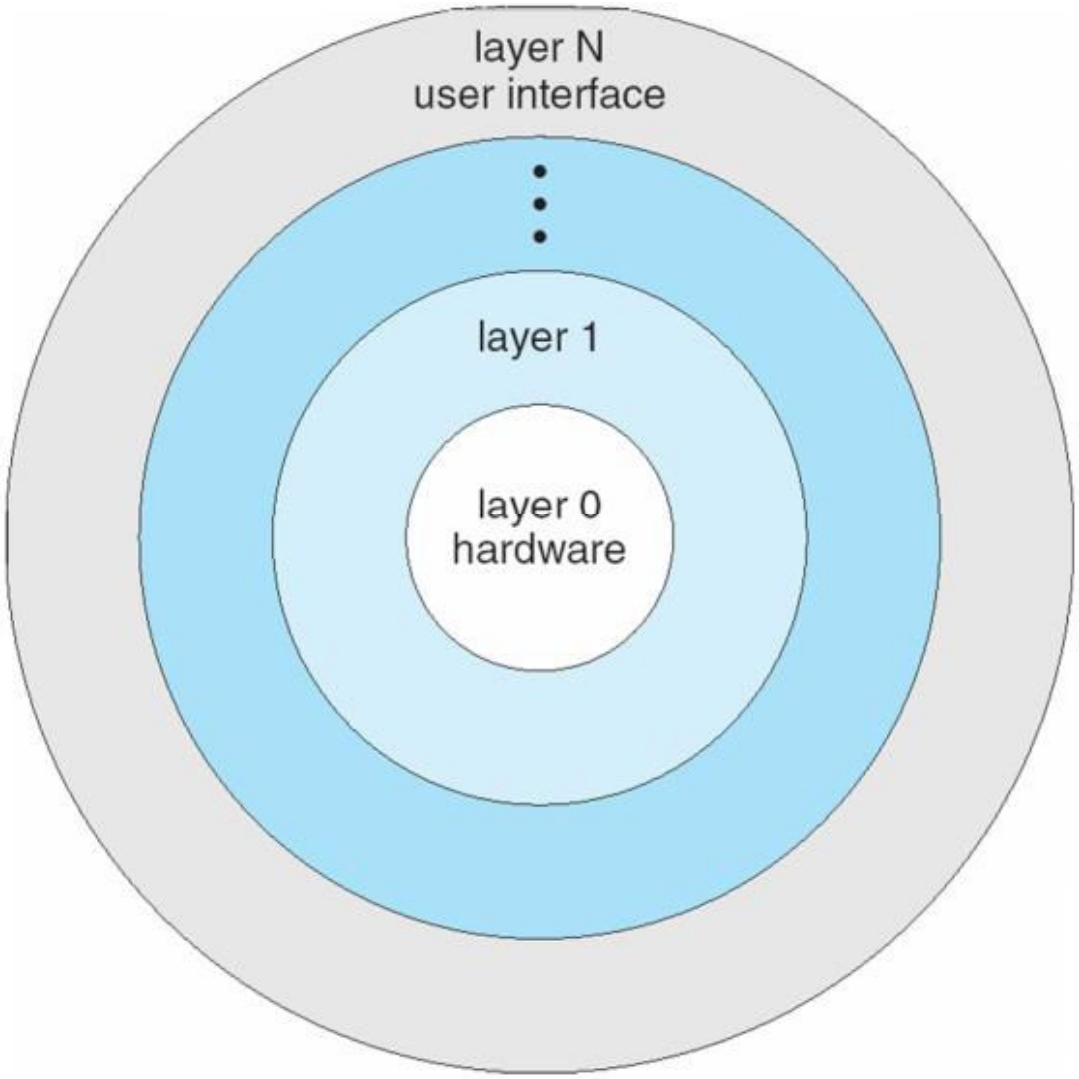
Traditional UNIX System Structure



UNIX

- 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 flexible

System Calls

General-System Architecture

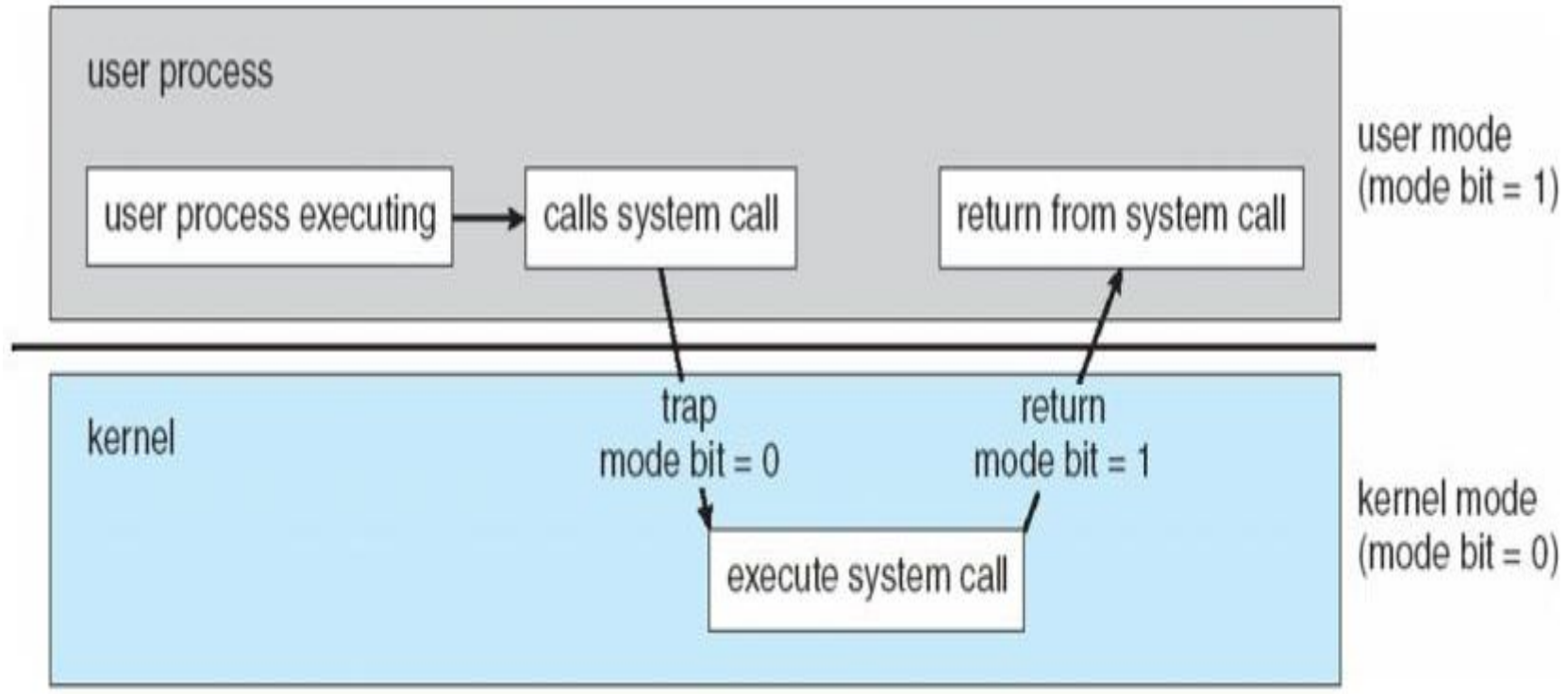
Why we need protection?

- In single task / programming environment
  - To protect OS from incorrect program
- In multiprogramming / multitasking environment
  - To protect OS and other programs from incorrect program
- Given the I/O instructions are privileged, how does the user program perform I/O?

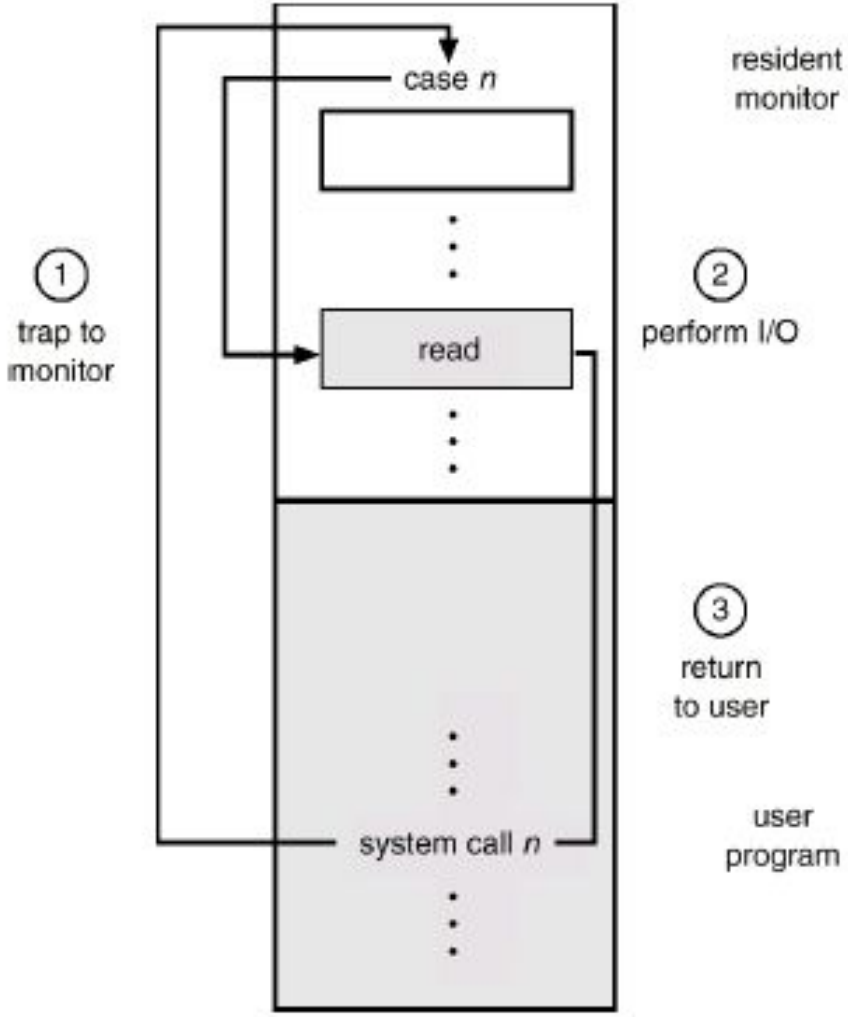
Operating-System Operations

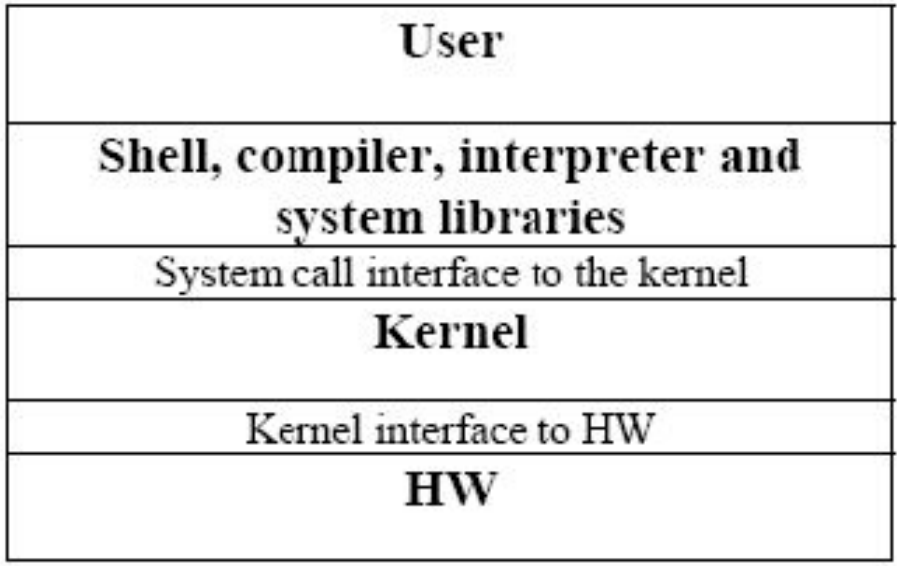
- Dual-mode operation allows OS to protect itself and other system components
- User mode and kernel mode
- Mode bit provided by hardware

Transition from User to Kernel Mode



Use of A System Call to Perform I/O





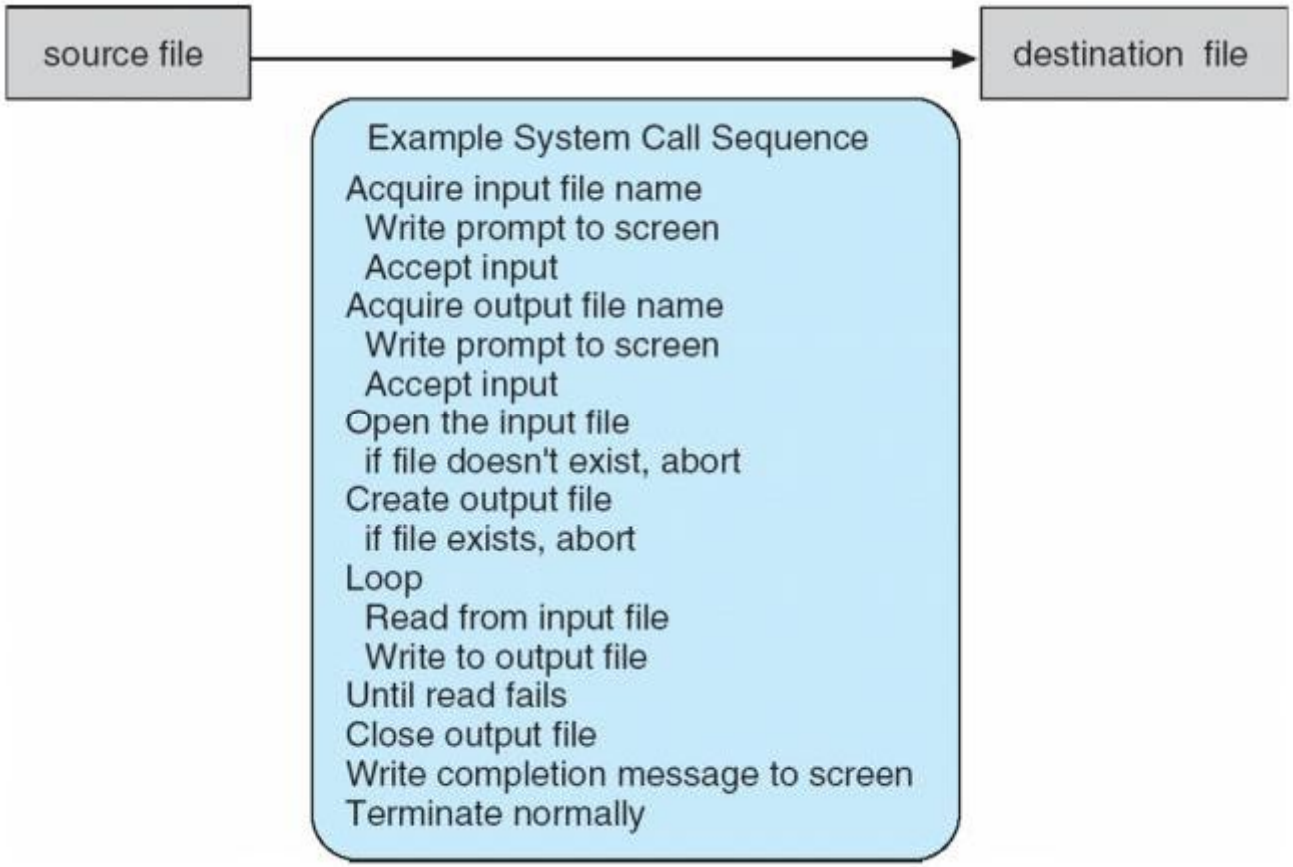
How to communicate ?

How to communicate between two modes?

- System calls have always been the means through which user space programs can access kernel services. (Typically written in a high-level language (C or C++)
- System call is the only legal entry point to the kernel
- System call provides an abstract hardware interface for user space
- Application need not worry about type of disk, media and file system in use
- System call ensures system stability and security.
- Kernel can keep track of application's activity

Example of System Calls

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



System Calls

- System call – the method used by a process to request action by the operating system provide the interface between a running program and the operating system

- It is the mechanism used by an application program to request service from the operating system

- Often use a special machine code instruction (i.e. software interrupt or trap) which causes the processor to transfer control to a specific location in the interrupt vector (kernel code)

- Usually takes the form of a trap to a specific location in the interrupt vector

System Call Implementation

- The process fills the registers with the appropriate values and calls a special instruction which jumps to a previously defined location in the kernel (under Intel CPUs, this is done by means of interrupt 0x80)

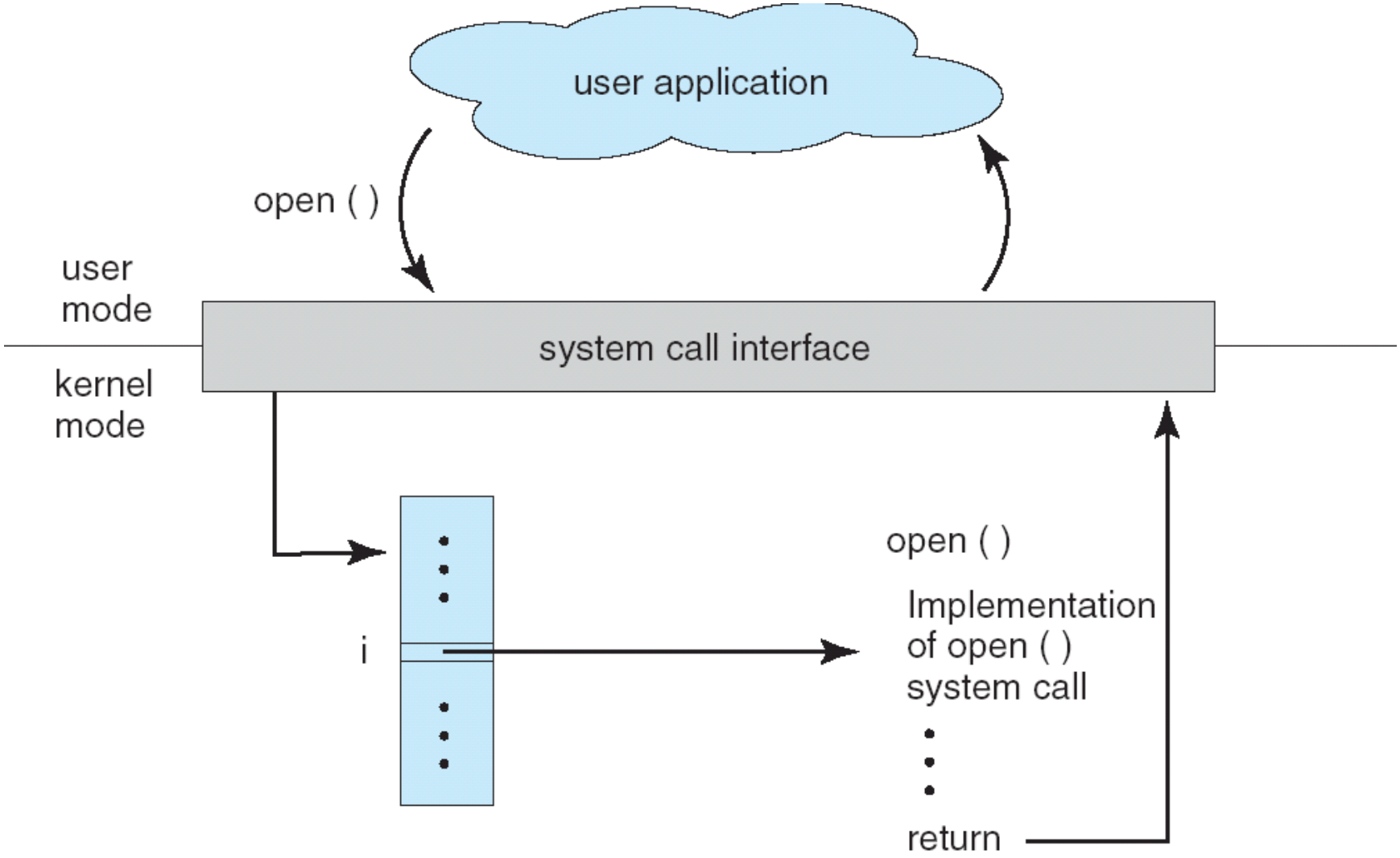
- Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to monitor mode.

- The monitor mode verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

System Call Implementation

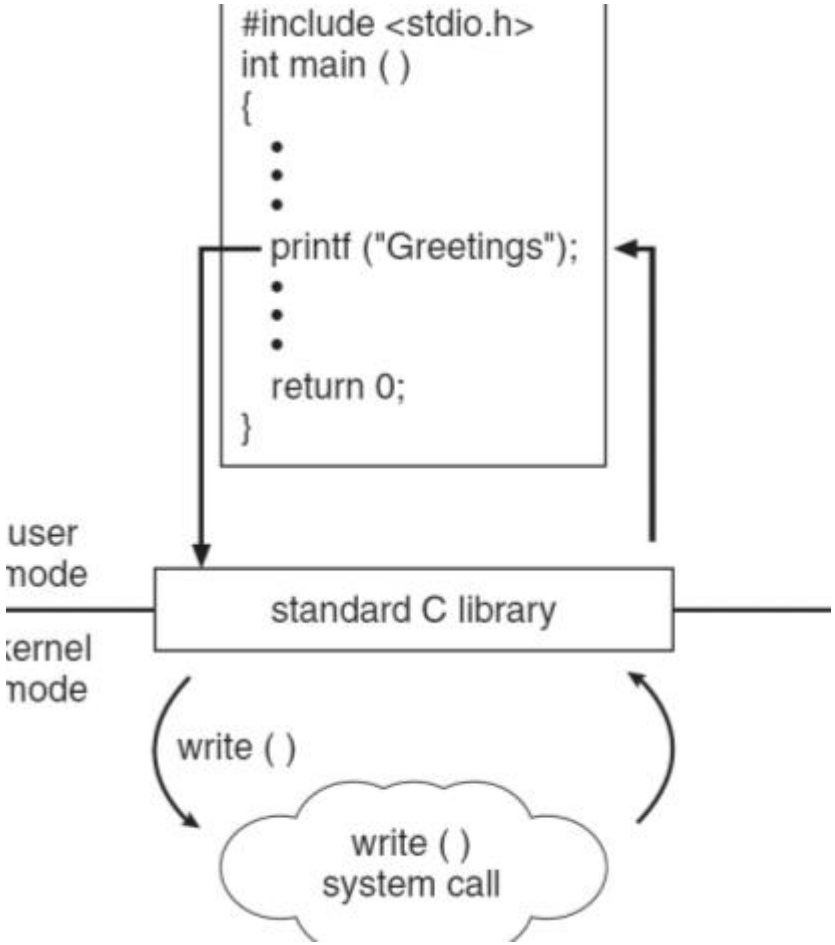
- Typically, a number associated with each system call.
- System-call interface maintains a table indexed according to these numbers.
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values.
- The caller need not know any thing 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 .
- Managed by run-time support library (set of functions built into libraries included with compiler)

API –System Call –OS Relationship



Standard C Library Example

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



System Call Parameter Passing

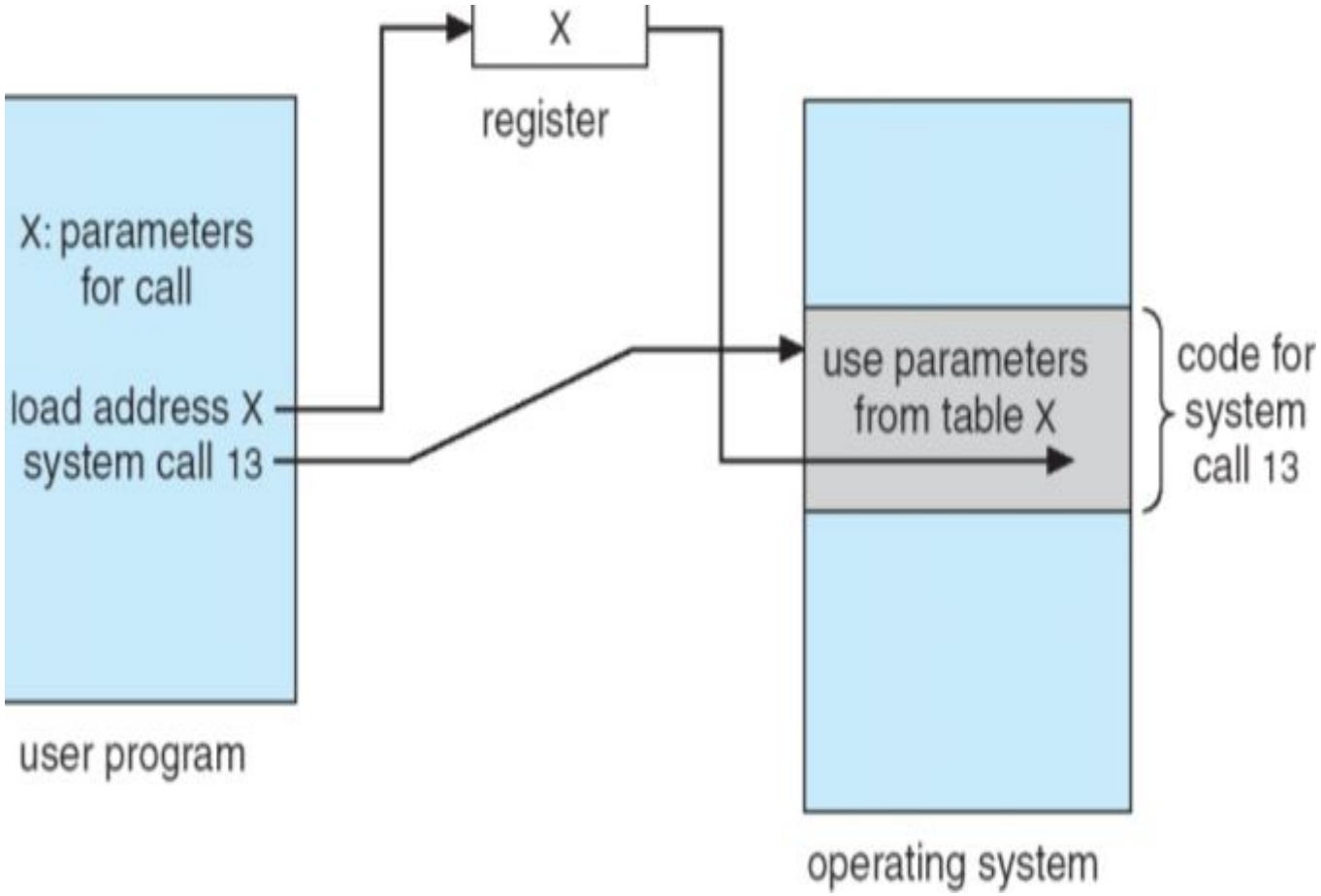
- Often, more information is required than simply identity of desired system call.
- Exact type and amount of information vary according to OS and call.
- Most of the system calls require one or more parameter to be passed to them
- Parameters are stored in registers EBX, ECX, EDX, ESI and EDI (if the parameters are less than six)
- If number of parameters are more than five (very rare) a single register is used to hold a pointer to user space where all parameters exist.

Methods to pass parameters to OS

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 is 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



New System calls – Design considerations

- What is the purpose of new system call?
  - If possible it should have only one purpose
- What are the new system call's arguments, return value and error codes?
- Is it portable and robust?
- Verifying system call parameters
  - Ensure it is valid and legal
  - Important to check the validity of pointers a user gives
  - Before following a pointer into user space, the system must ensure
    - The pointer points to a region of memory in user space
    - The pointer points to a region of memory in the process's address space
    - If reading, memory is marked readable. If writing memory is marked writable.

Difference between System call and function call

- System call typically accessed via function calls.
- System call involves **context switching** (from user to kernel and back) where as function call does not.
- Takes much longer time than function calls.
- Avoiding excessive system calls might be a wise strategy for programs that need to be tightly optimized.
- Most of the system calls return a value (error if failed) where as it is not necessary for subroutine.
- If an error (return value –1) use perror (“Message”)
- to print the error.

Pros and Cons of system calls

- Pros
  - Simple to implement and easy to use
  - In Linux it is very fast
- Cons
  - Need a system call number, which needs to be officially assigned to you during the developmental kernel series.
  - Once stabilized, the interface can not change with out breaking user space application
  - Each architecture needs to separately register the system call and support it.
  - For simple exchanges of information, a system call is overkill (overheads because of cache miss and context switch)

Types of System Calls

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

System Calls for process control

- fork()
- wait(), waitpid()
- execl(), execlp(), execv(), execvp()
- exit()
- Signal - signal(sig, handler), kill(sig, pid), alarm(), pause()
- getpid(), getppid()
- nice()

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()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()