Chapter 1: Introduction

(OS Structure, Modes and Services)

Operating System?

- It is a layer of system software that:
  - directly has privileged access to underlined hardware
  - hides hardware complexity
  - manages hardware on behalf of one or more applications.

What is an Operating System?

- **What is an Operating system?**
  - **A program that acts as an intermediate/ interface between a user of a computer and the computer hardware.**
  - **Resource allocator (Managing the resources efficiently)**
  - **Control Program**

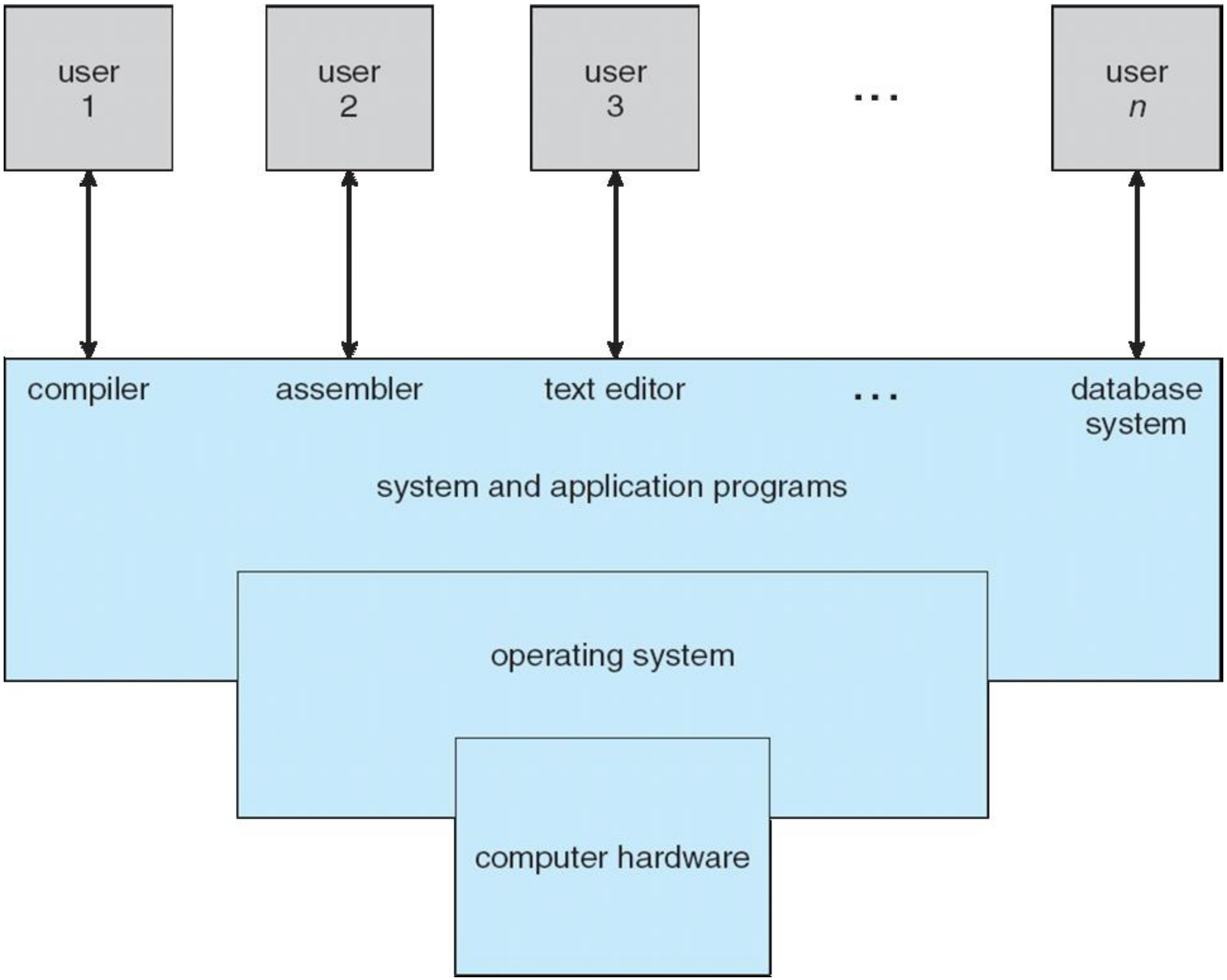
- **Operating system goals:**
  - **Execute user programs and make problem solving easier.**
  - **Make the computer system convenient to use**
  - **Efficiently use available resources**

- An operating system is the one program that is running at all the times on the computer- usually called the kernel.
- Kernel is a program that (allow) let the hardware to recognize and read the program/process.

Computer System Structure

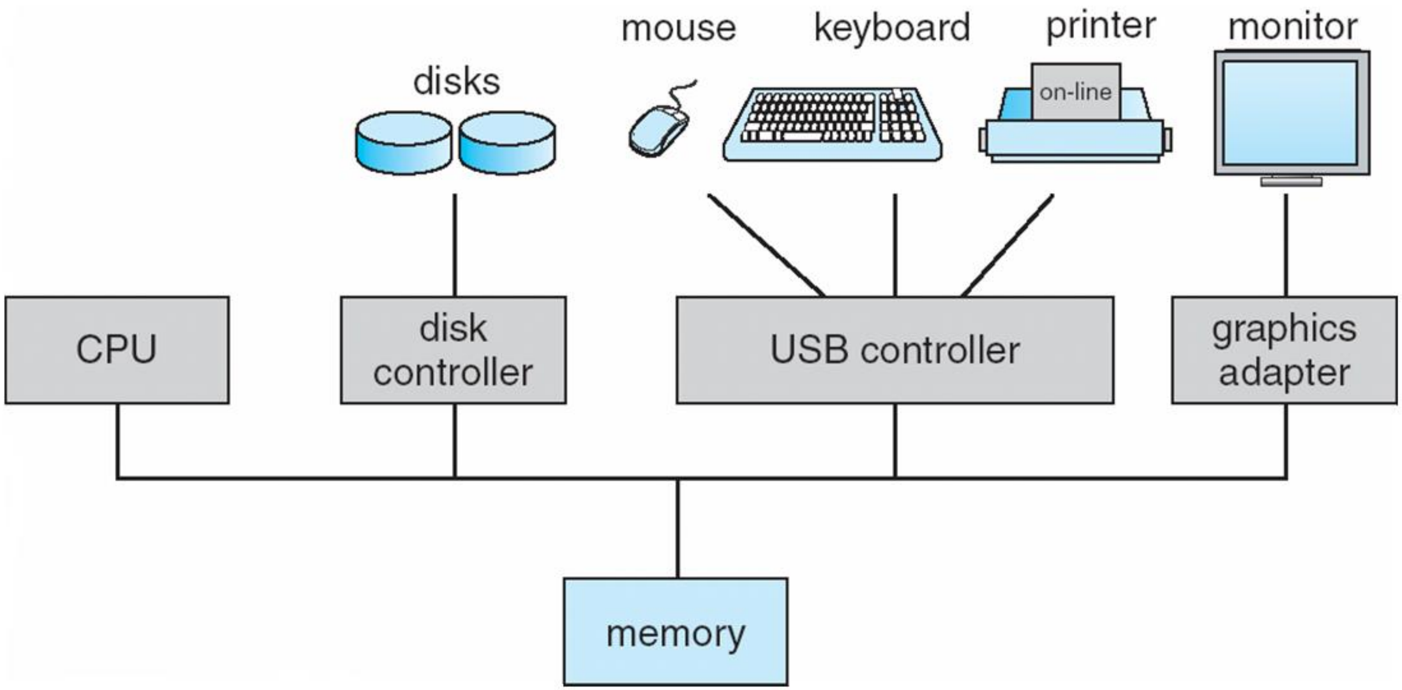
- **Computer system can be divided into four components:**
  - **Hardware** – provides **basic computing resources**
    - CPU, memory, I/O devices
  - **Operating system**
    - **Controls and coordinates use of resources** among various applications and users
  - **System/Application programs** – define the ways in which the system resources are used to solving user problems
    - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers

Four Components of a Computer System



Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



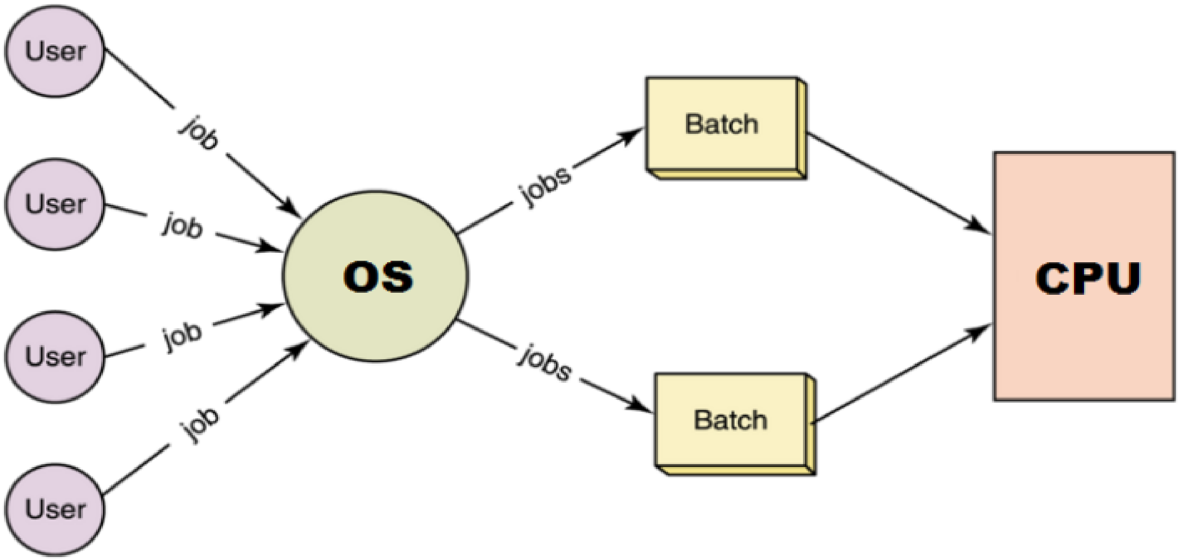
TYPES OF OS

Batch Systems

“Batch operating system. The users of a batch operating system do not interact with the computer directly.

Each user prepares his job on an off-line device like punch cards and submits it to the computer operator.

To speed up processing, jobs with similar needs are batched together and run as a group”.



m]

TYPES OF OS: Batch Systems

- There is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirements.

Disadvantages:

- No interaction between user and computer.
- No mechanism to prioritize the processes

TYPES OF OS: Batch Systems

- The common **input devices** were **card readers and tape drives**.
- The common **output** devices were **line printers, tape drives, and card punches.**

- **https://www.youtube.com/watch?v=sq2SE_GbZ34**

Multiprogrammed OS

Multiprogramming: When 2 or more processes reside in memory at the same time

- Single user processes cannot keep CPU and I/O devices busy at all times
- **Multiprogramming organizes jobs (code and data) so CPU always has one to execute**
- Multiprogramming assumes a **single shared processor.** One job selected and run via **job scheduling**
- Multiprogramming increases CPU utilization.
- **It is mixture of I/O bound and CPU bound processes**

Multiprogrammed OS

- In this the operating system picks up and begins to execute one of the jobs from memory.

- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).

- Jobs in the memory are always less than the number of jobs on disk(Job Pool).

Multiprogrammed OS

- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.

- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.

- In Multiprogramming system, CPU will never be idle and keeps on processing.

Multitasking/Timesharing OS

- **Timesharing (multitasking)** when multiple jobs are executed by the CPU simultaneously by switching between them.
  - There is at least one program is executing in memory ⇨ **process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - **Only one CPU is involved**, but **it switches** from one process to another **so quickly** that it gives the appearance of **executing all of the processes at the same time.**

Multiprocessing OS

- A multiprocessor system consists of several processors that share a common physical memory.
- Multiprocessor system provides higher computing power and speed.
- In multiprocessor system all processors operate under single operating system.

Multiprocessing OS

- Multi-processor systems; that is, they **have multiple CPU.**

- **Also known as parallel systems** or **tightly coupled systems**

- **Such systems have more than one processor** in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

Distributed Systems

- A network is a communication path between two or more systems.
- Each system over the network keeps copy of the data, and this leads to Reliability (Because if one system crashes , data is not lost).

- CLIENT SERVER SYSTEMS
- PEER TO PEER SYSTEMS

Real Time Systems

- **Time bound systems**
- Real time systems are of 2 types:

- **1. Soft Real time Systems:** Process should complete in specific time but May have some delay (Positive delay) and will not harm the system.

Exp: Session expires but can be re-logged in.

- **2. Hard Real Time Systems:** Each process is assigned a specific time instance, and Process must complete in that time otherwise system will crash.

Real Time Embedded Systems

- is a computing environment that **reacts to input within a specific time period.**
- Time Driven
- Task specific
- Exp: Microwave, Washing Machine…

Operating System Views

OS can be explored from 2 view points:

1. User view:

- The goal of the Operating System is to **maximize the work and minimize the effort of the user**.
- Operating System gives an effect to the **user as if the processor is dealing only with the current task**, but in background processor is dealing with several processes.

Operating System Views

OS can be explored from 2 view points:

2. System View:

Operating System is a program involved with the hardware.

- OS is a **resource allocator**
  - Allocates and Manages all resources and their sharing.
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls **execution of programs to prevent errors and improper use of the computer**
  - **It prevents** improper usage, error and handle **deadlock** conditions.

Operating-System Operations

- **OS's are Interrupt driven**. If no process, no I/o devices, No users then OS will sit quietly waiting for some event to occur.

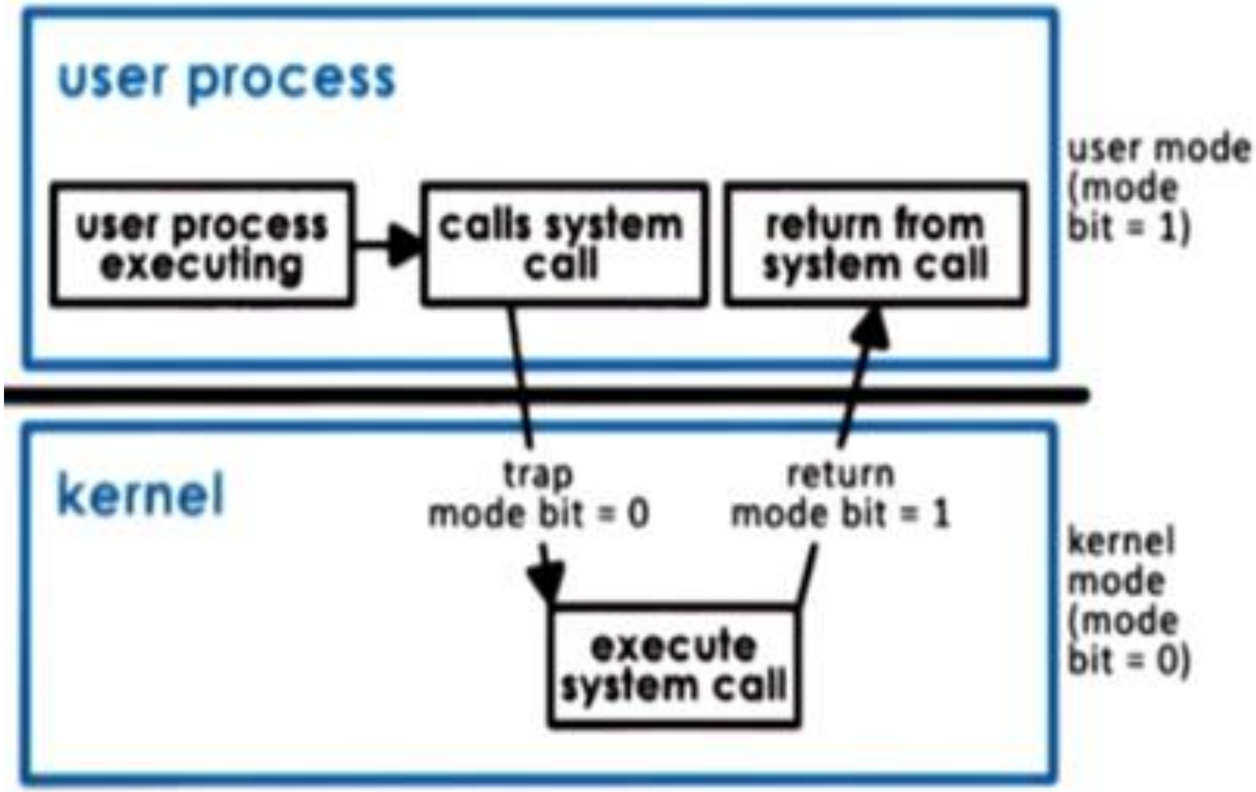
- Program or **software send, Generate Events by using system calls**. Error or request by a software creates **exception** or **trap**
  - E.g. Division by zero

- **To ensure proper execution of OS, we must distinguish between execution of OS code and user defined code.**

Operating-System Operations

- **To protect OS, Dual-mode** operations exist:
  - **User mode (1)** and **kernel mode (0)**
  - A **Mode bit is added to** hardware to indicate mode
    - Provides ability to distinguish when system is running user mode or kernel mode
    - System call changes mode to kernel, return from call resets it to user

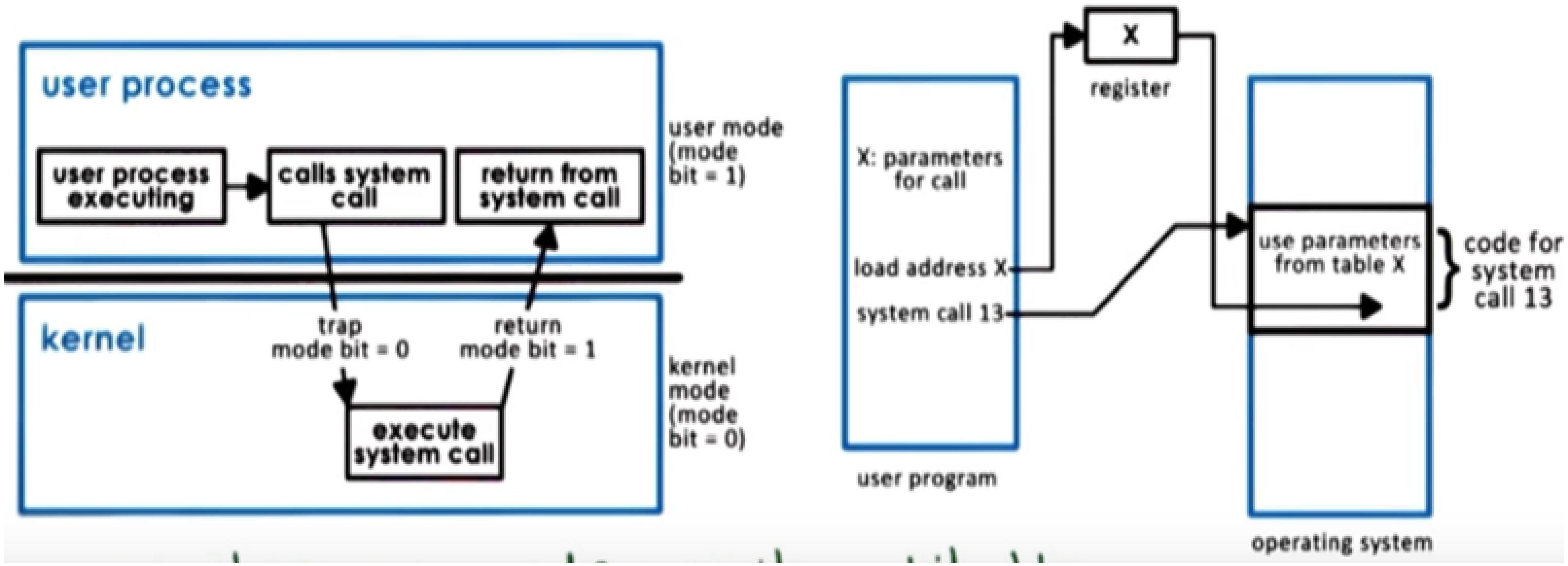
Transition from User to Kernel Mode



To make a system call an application must

- write arguments
- save relevant data at well-defined location
- make system call

Transition from User to Kernel Mode

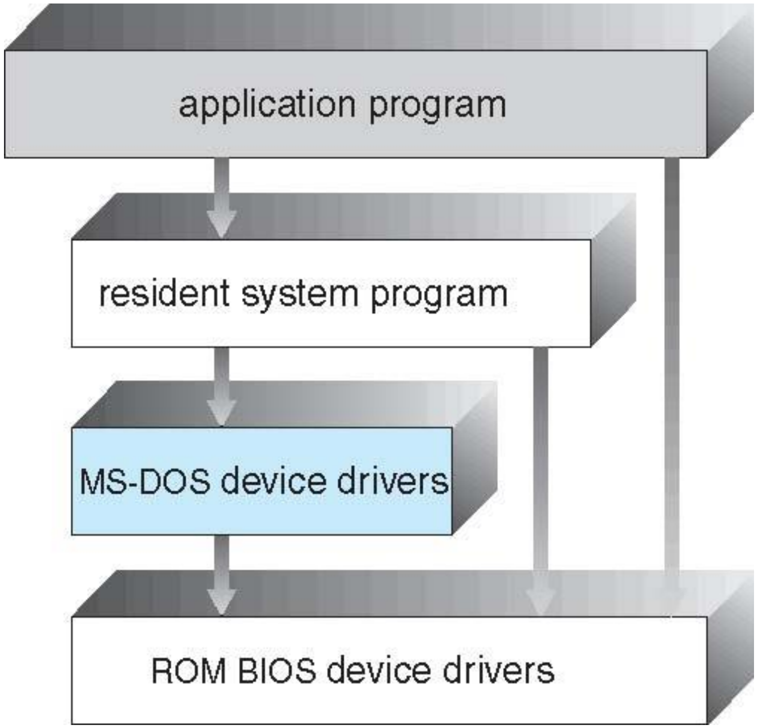


Operating System Structure

- Various ways to structure a system

Simple Structure -- MS-DOS

- 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

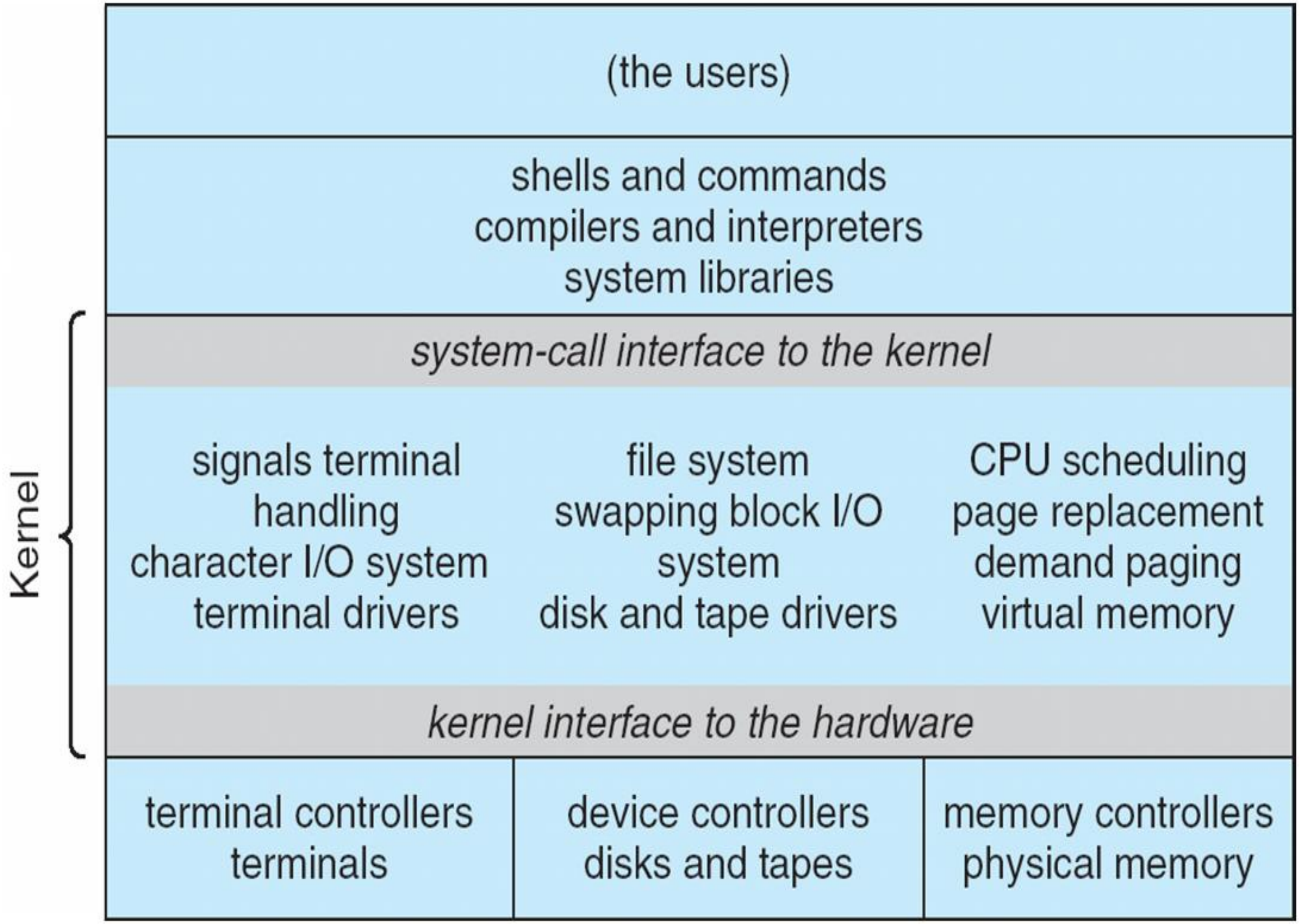


Non Simple Structure -- UNIX

The UNIX OS consists of two parts:

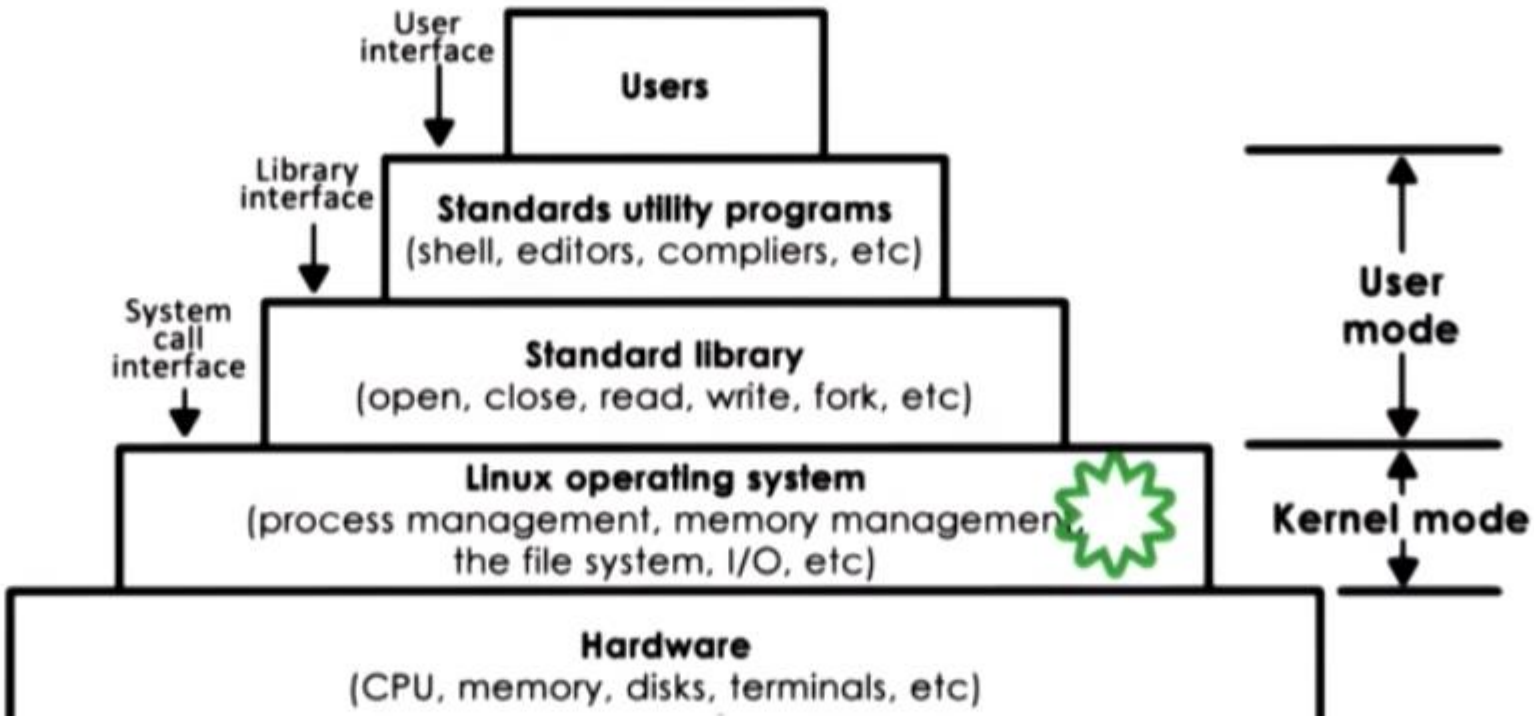
- System programs
- The kernel
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Traditional UNIX System Structure

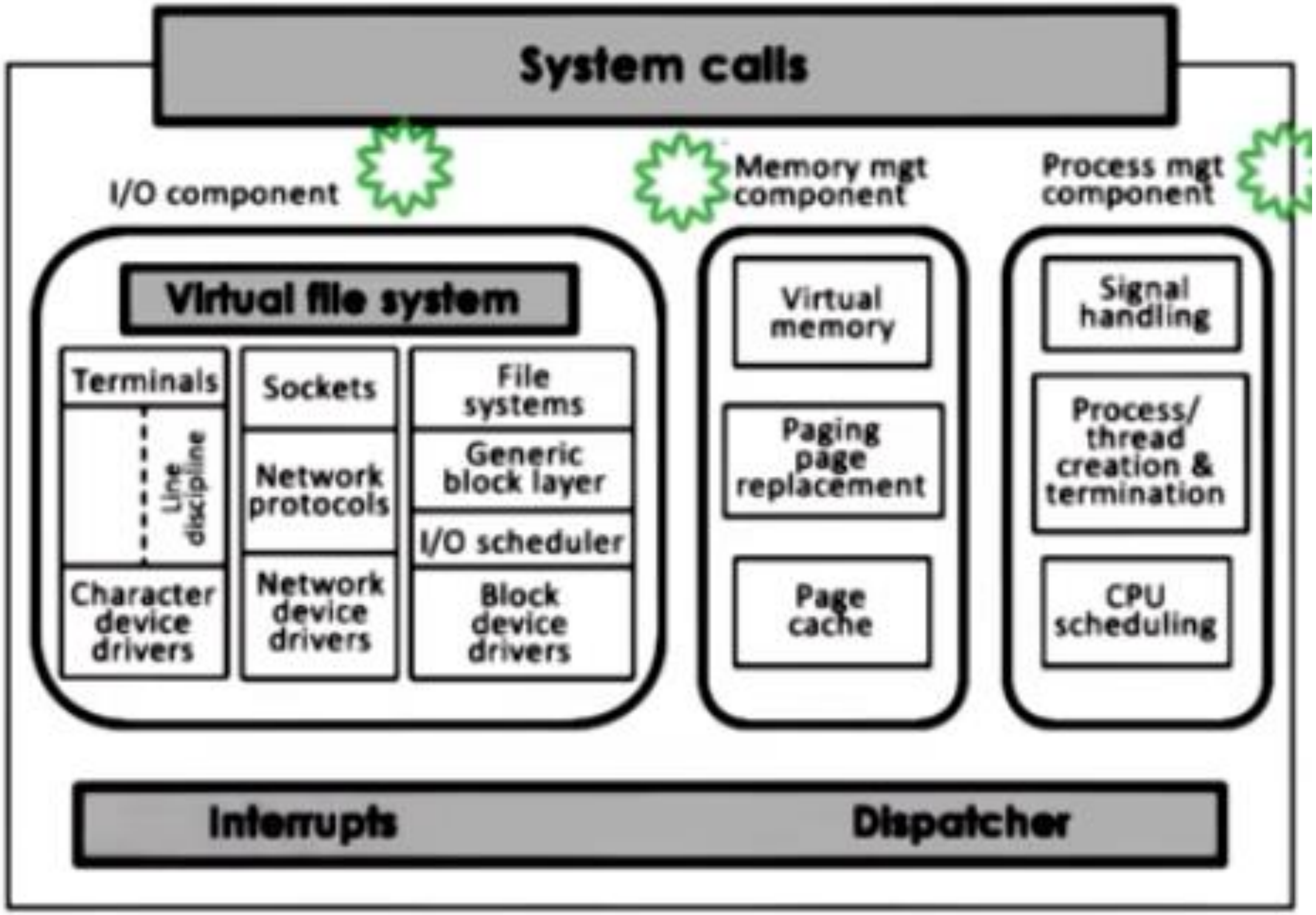


Traditional LINUX System Structure

Linux Architecture

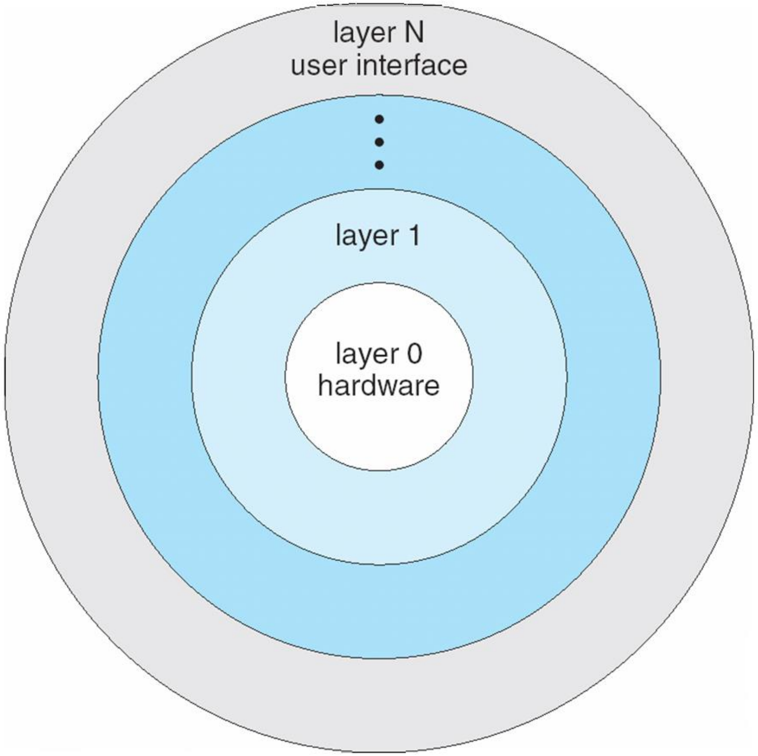


Kernel has many inbuilt modules



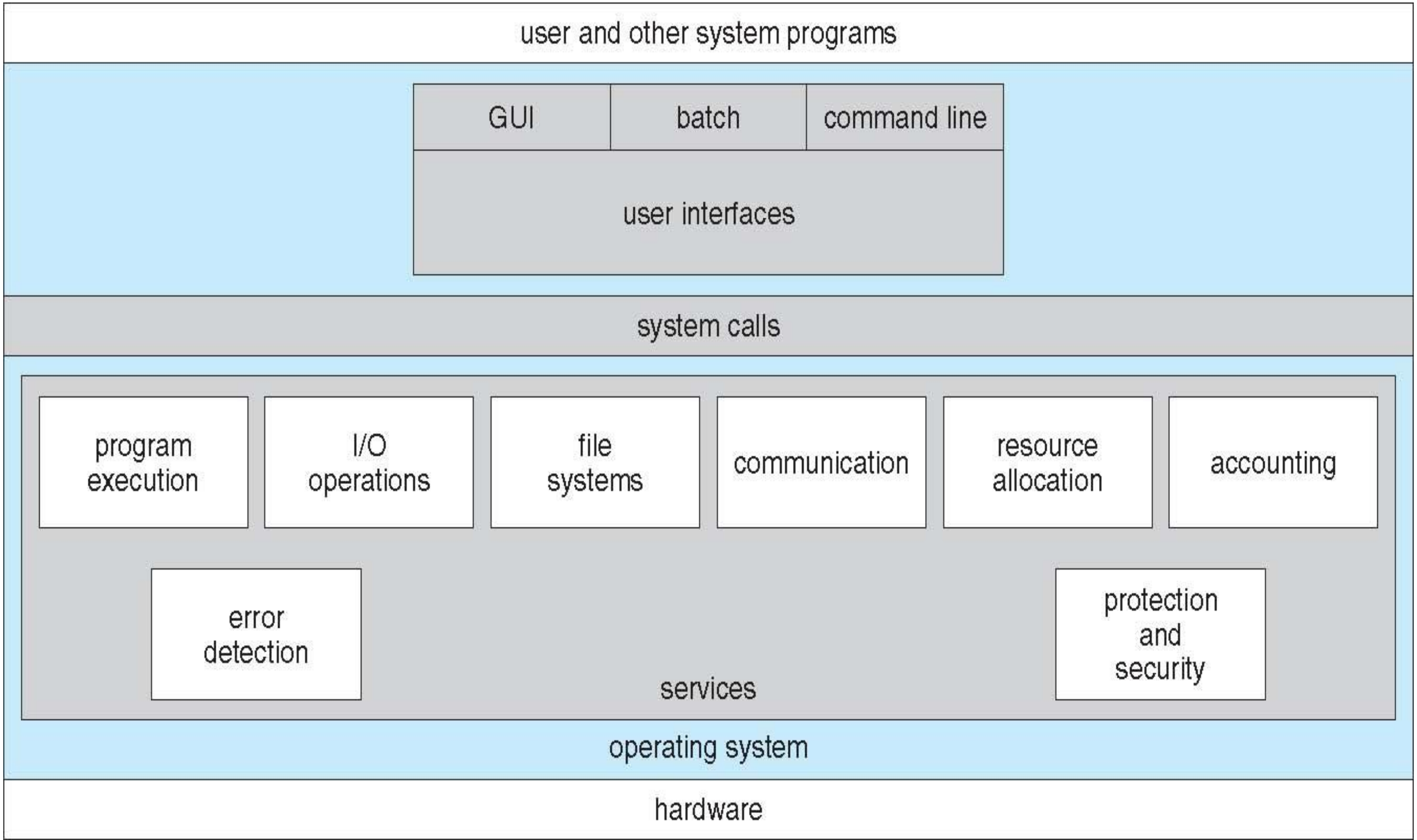
Layered Approach

- 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



Operating System Services

- An operating system provides an **environment for the programs to run.**
- It provides certain services to programs



Operating System Services

- 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).
  - **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** - read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services

- **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 facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

- **Resource allocation –** OS must ensure allocation of resources to all programs running.
  - **Many types of resources** - such as **CPU cycle time, main memory, and file storage**, **I/O devices**

Operating System Services

- **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.**

Kernel

- A kernel is a central component of an operating system.
- It acts as an interface between the user applications and the hardware.
- The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc).
- The main tasks of the kernel are :
  - Process management
  - Device management
  - Memory management
  - Interrupt handling
  - I/O communication
  - File system...etc..

Kernel

- A kernel is the lowest level of software that interfaces with the hardware in your computer.

- It is responsible for interfacing all applications that are running in “user mode” down to the physical hardware, and allowing processes, to get information from each other using inter-process communication (IPC).

Kernel Types

kernels fall into one of three types:

- Monolithic
- Microkernel
- Hybrid

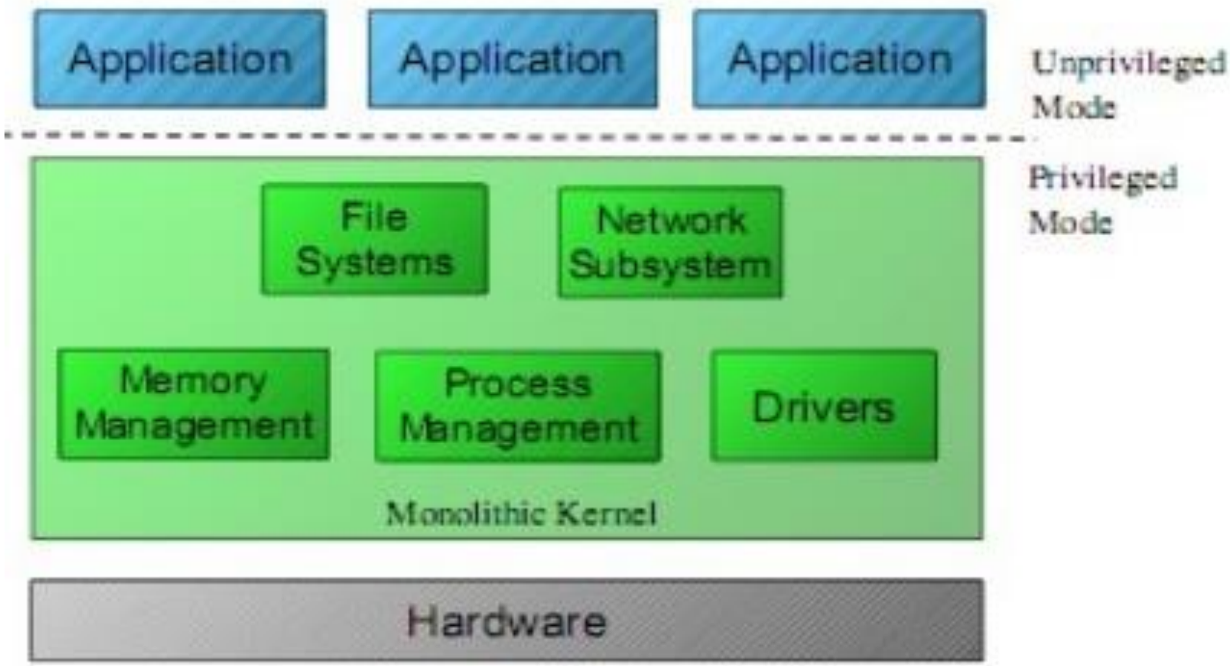
Monolithic Kernel

- A **monolithic kernel** is an operating system architecture where the entire operating system (which includes the device drivers, file system, and the application IPC etc.) is working in kernel space, in supervisor mode.

- Monolithic kernels are able to dynamically load (and unload) executable modules at runtime.

- Examples of operating systems that use a monolithic kernel are - Linux, Solaris, OS-9, DOS, Microsoft Windows (95,98,Me) etc.

Monolithic Kernel



Monolithic Kernel

Pros:

- More direct access to hardware for programs
- Easier for processes to communicate between each other
- If your device is supported, it should work with no additional installations
- Processes react faster because there isn’t a queue for processor time.

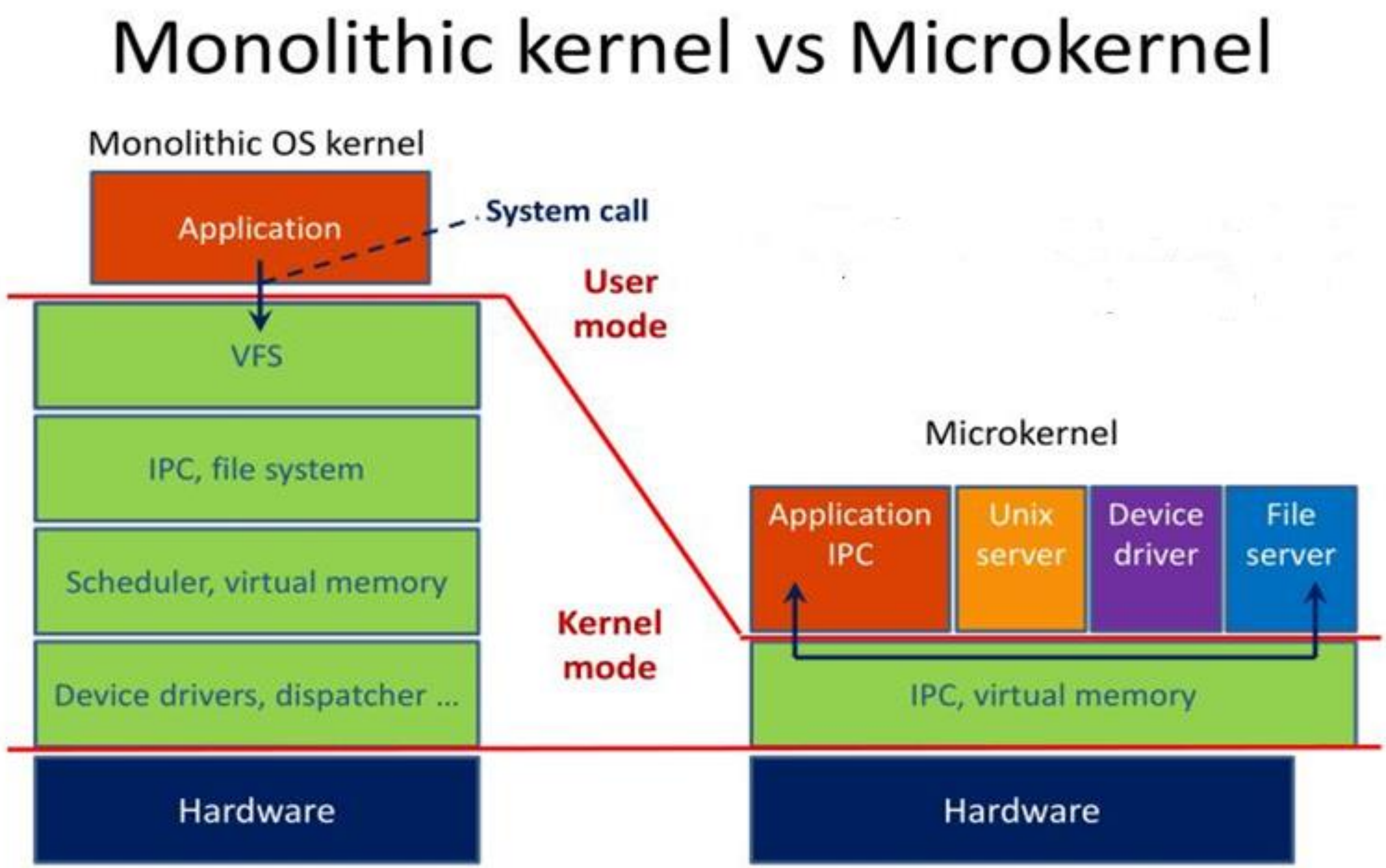
Cons:

- Large install footprint
- Large memory is needed
- Less secure because everything runs in supervisor mode

MicroKernel

- In a Microkernel architecture, the core functionality is isolated from system services and device drivers.
- This architecture allows some basic services like device driver management, file system etc. to run in user space.
- This reduces the kernel code size and also increases the security and stability of OS as we have minimum code running in kernel.
- Examples of operating systems that use a microkernel are - QNX, Integrity, PikeOS, Symbian, L4Linux, Singularity, K42, Mac OS X, HURD, Minix, and Coyotos.

Types of Kernel



MicroKernel

Pros

- Portability
- Small install footprint
- Small memory
- Security

Cons

- Hardware is more abstracted through drivers
- Hardware may react slower because drivers are in user mode
- Processes have to wait in a queue to get information
- Processes can’t get access to other processes without waiting

HybridKernel

- Hybrid kernels have the ability to pick and choose what they want to run in user mode and what they want to run in supervisor mode.
- Device drivers and file system I/O will be run in user mode while IPC and server calls will be kept in the supervisor mode.
- This require more work of the hardware manufacturer because all of the driver responsibility is up to them.

Hybrid Kernel

Pros

- Developer can pick and choose what runs in user mode and what runs in supervisor mode
- Smaller install than monolithic kernel
- More flexible than other models

Cons

- Processes have to wait in a queue to get information
- Processes can't get access to other processes without waiting
- Device drivers need to be managed by user

Interrupts

- An interrupt is a signal from a device attached to a computer or from a program within the computer that causes the main program that operates the computer (the operating system) to stop and figure out what to do next.

- Interrupts can be of following type:

  - Generated by Hardware (Hardware Interrupt)

  - Generated by Software (Software Interrupt)

Hardware Interrupt

1. Hardware interrupts are used by **devices** to communicate that they **require attention from the operating system**.
2. Hardware interrupts by sending signal to CPU **via system bus.**
3. Hardware interrupts are referenced by an ***interrupt number*.**
4. These numbers are mapped with hardware that created the interrupt. This enables the system to monitor/understand **which device created the interrupt and when it occurred**.

Software Interrupt/ Trap

- Interrupt generated by executing a instruction.
- Software interrupts by a special operation called a System Call or Monitor Call.

Exp: 1. cout in C++ is a kind of interrupt because it would make a system to print something.

2. division by zero

System Calls

- Allow user-level processes to request services of the operating system.
- It provides a way in which program talks to the operating system.
- It is a call to the kernel in order to execute a specific function that controls a device or executes a instruction.

Why system calls are required?

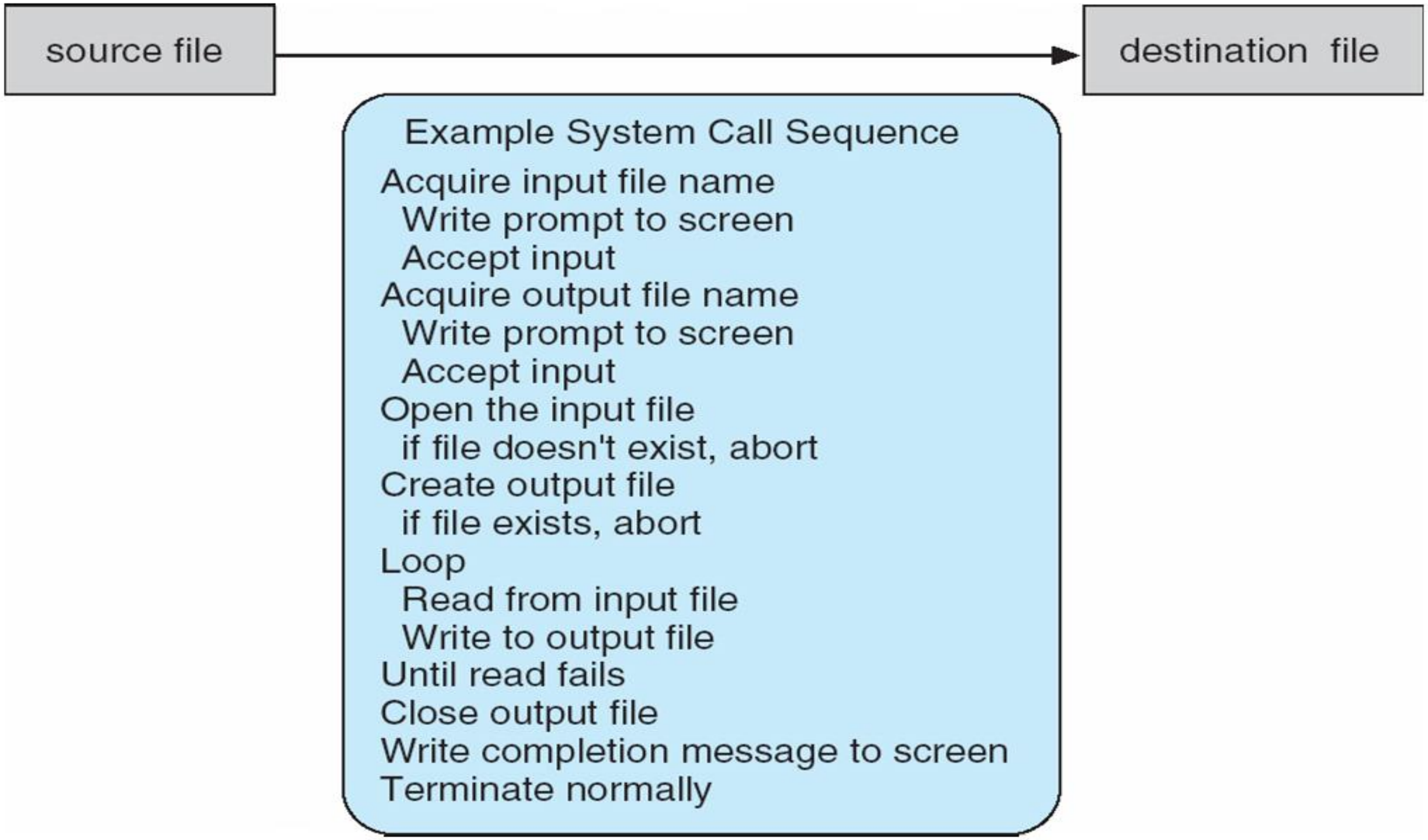
- **It is a request to the operating system to perform some activity**.
- A system call looks like a procedure call

Accessing System Calls

- System calls are accessed by programs via a high-level **Application Program Interface (API)** (provides run time environment)
- System calls are implemented via API, API’s interact with kernel of OS.

Example of System Calls

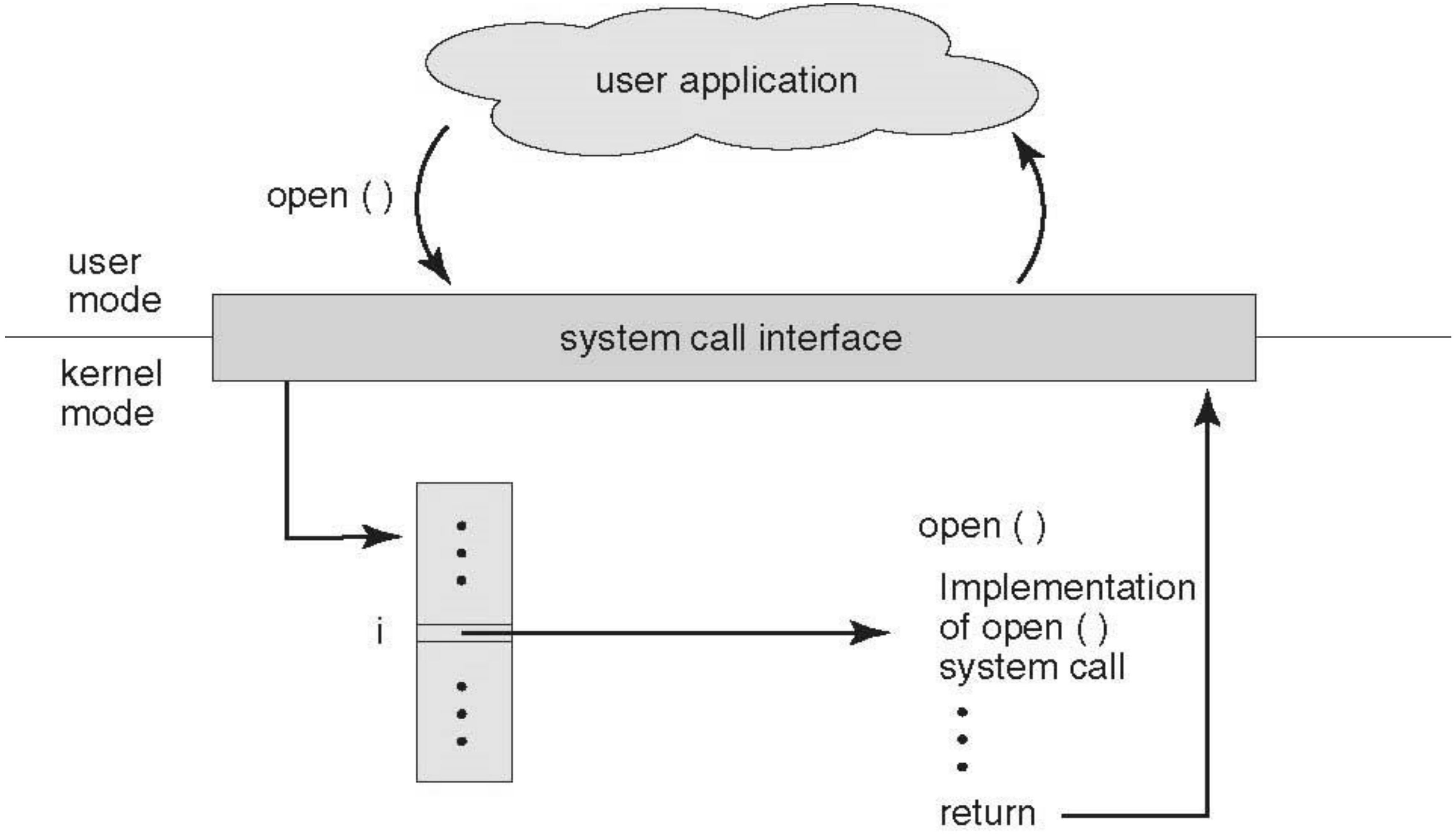
- System call sequence to copy the contents of one file to another file



System Call Implementation

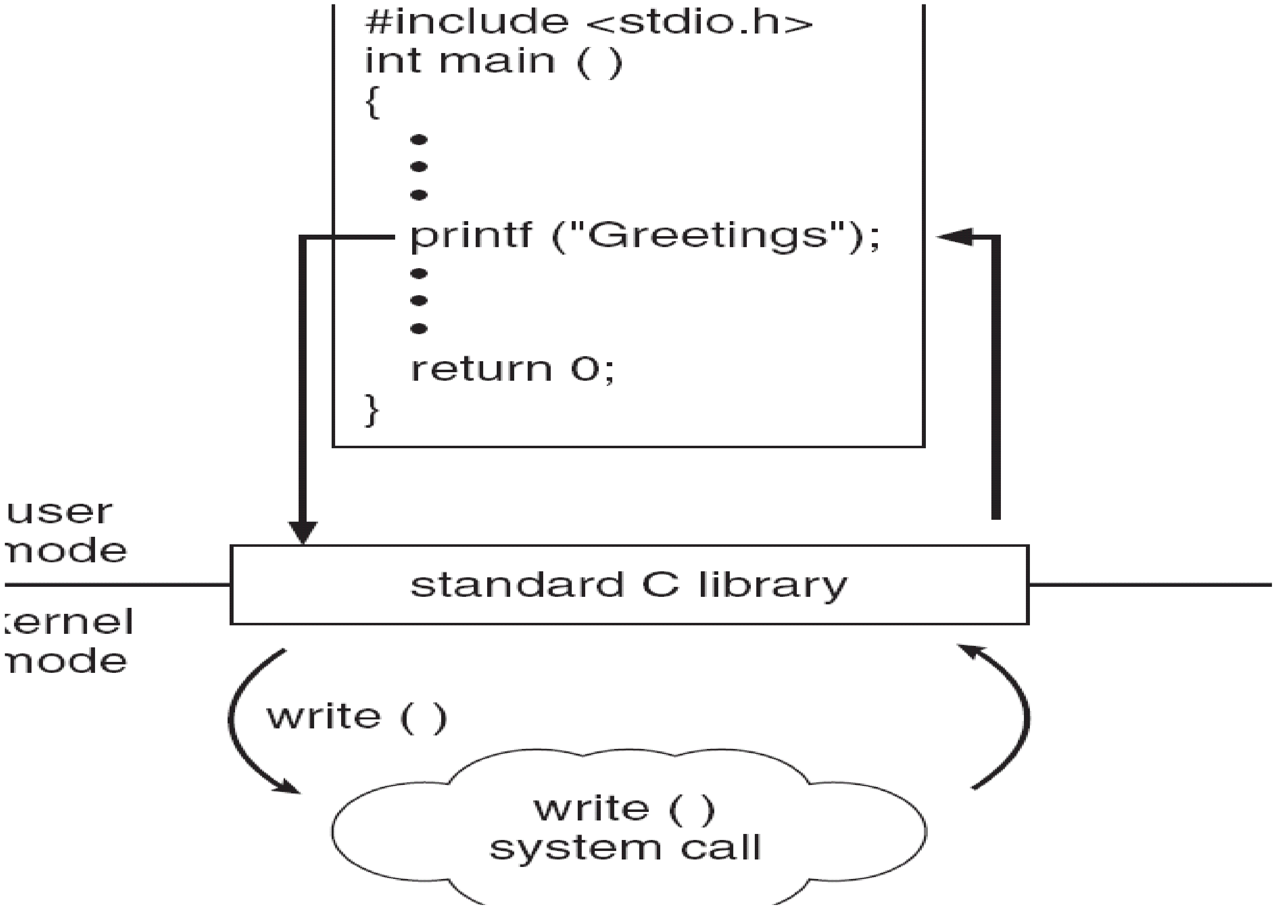
- A number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call with a return value.
- The caller needs to know nothing about how the 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

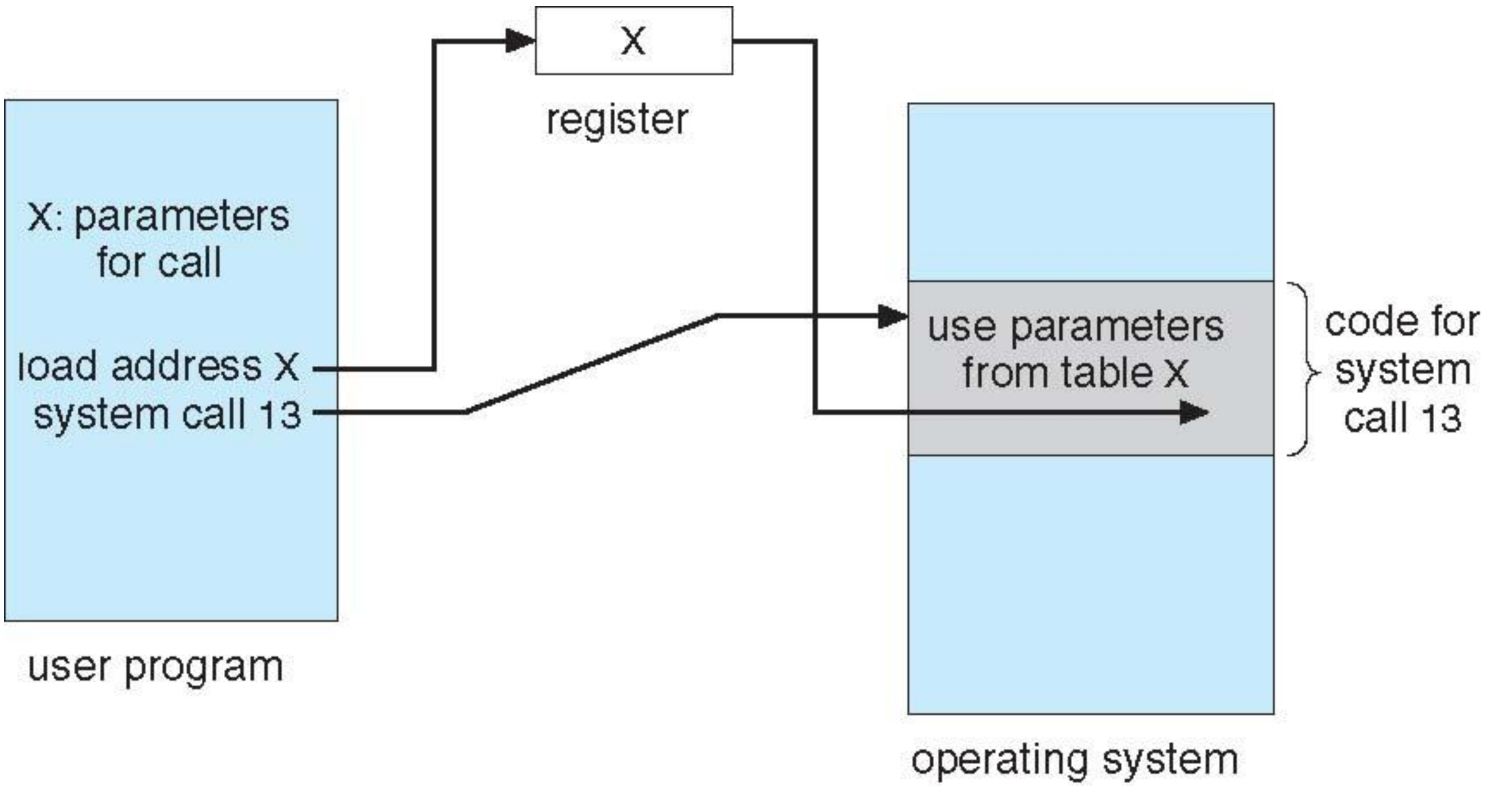
Passing Parameters to System Calls:

- Information required for a system call vary according to OS and call.

- **Three general methods used to pass parameters to the OS**

  1. **Pass the parameters in *registers***
     - When parameters are < 6.
  2. **Parameters stored in a *block,* or table**, in memory, and address of block passed as a parameter in a register. (6 or more)
     - This approach taken by Linux and Solaris
  3. **Parameters placed**, or *pushed,* **onto the *stack*** by the program and *popped* off the stack by the operating system.

Parameter Passing via Table



Types of System Calls
5 Categories

- Process Control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory

- File Management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Types of System Calls (Cont.)

- Device Management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information Maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

Storage Structure and Hierarchy