K. J. Somaiya College of Engineering, Mumbai
(A constituent College of Somaiya Vidyavihar University)

Operating System

Module 1. Introduction to System software

Dr. Prasanna Shete
Dept. of Computer Engineering
prasannashete@somaiya.edu
Mobile/WhatsApp 9960452937

Module 1: Introduction to System software

- Concept, introduction to various system programs such as assemblers, loaders, linkers, macro processors, compilers, interpreters, operating systems, device drivers
- Operating System Objectives and Functions,
- The Evolution of Operating Systems
- Operating system structures
- OS Design Considerations for Multiprocessor and Multicore architectures
- System Calls
- Linux Kernel and Shell
- System boot

Operating System

- Entity that makes a computer system operational
- Interface between user and computer applications and hardware
- *OS is a system software that controls execution of programs and acts as an interface between applications and computer hardware*

Operating System

- Responsible for managing resources
- Functions in same way as ordinary computer software
  - It is a program that is executed by the processor

- Operating system relinquishes control of the processor and must depend on the processor to allow it to regain control

OS as Resource Manager

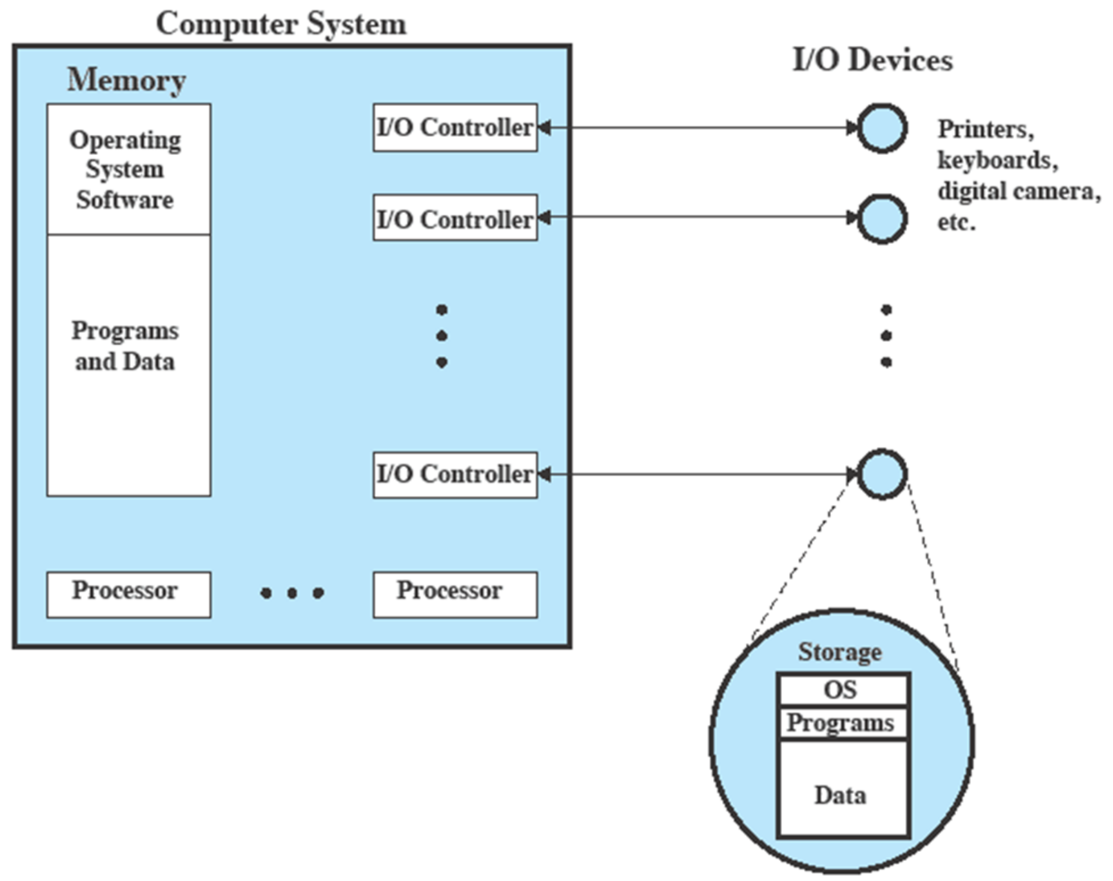


Figure 2.2 The Operating System as Resource Manager

As a resource manager, the OS

- **Manages Hardware Resources**
- **Manages Software Resources**
- **Ensures Fair Resource Distribution**

OS as a User/Computer Interface

The operating system acts as a bridge between users and the computer hardware.

It provides:

- **Command-Line Interfaces (CLI)**
- **Graphical User Interfaces (GUI)**
- **APIs (Application Programming Interfaces)**

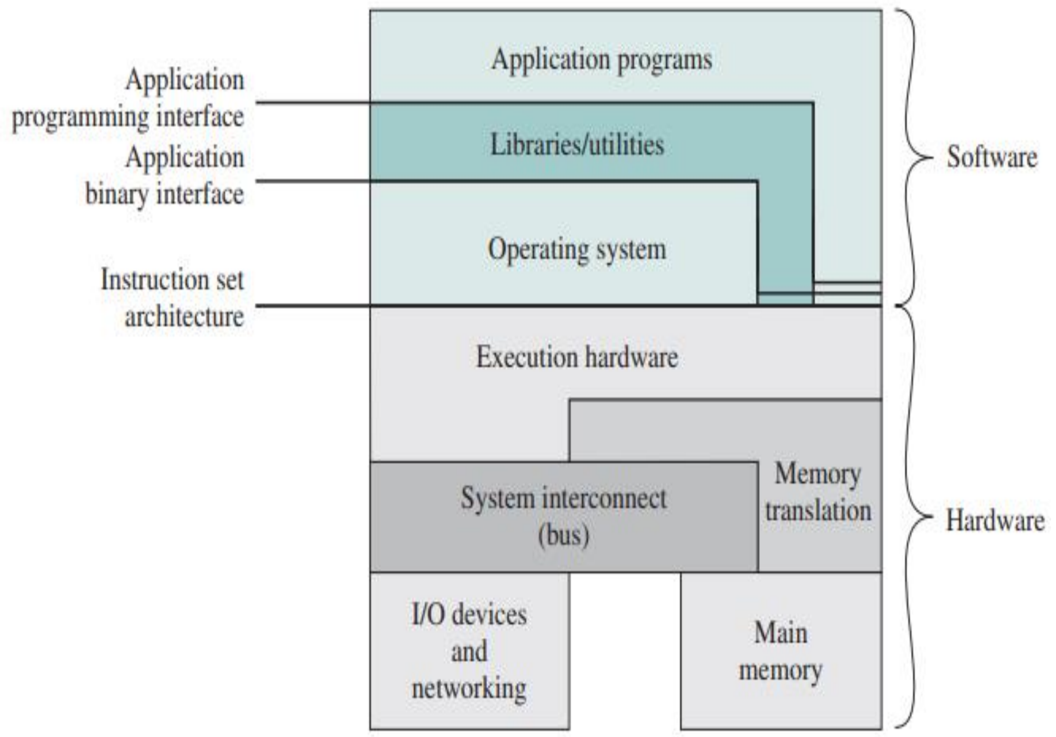

Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

Figure 2.1 Computer Hardware and Software Structure

Functions of Operating System

- Memory Management
- Process Management
- Device Management
- File Management
- I/O Management
- User Interface
- Security

Kernel, System call, CPU modes

- Kernel: part of OS that is residing in the main memory
  - Contains most frequently used OS functions

- System Call:
  - Method by which a program makes request to the OS
  - Standard functions available for direct interaction with the OS

- CPU Modes:
  - Kernel mode: mode for OS; all instructions are allowed
  - User mode: for user programs; I/O and certain privileged instructions are not allowed

Evolution of OS

- **Reasons ??**
  - Hardware upgrades, new types of hardware
  - New services
  - Fixes

1. Serial Processing
2. Simple Batch Systems
3. Multi-programmed Batch Systems
4. Time–Sharing Systems

1. Serial Processing

- No operating system
  - Programmer directly interacted with the computer hardware
- Computer System run from a "console" with display lights, toggle switches, input device, and printer
  - Error indicated by lights; examined by programmer through processor registers and main memory
- Limitations: Scheduling, setup time

1. Serial Processing…

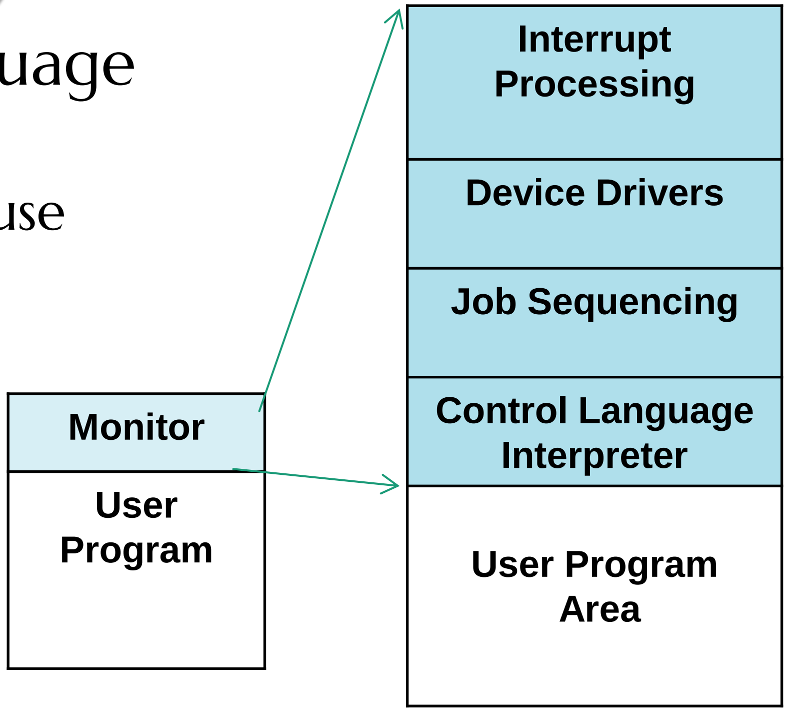
- Scheduling
  - Reservation of machine time using hardcopy signup sheet; reservation in terms of multiples of time-slots
  - wasteful if computer idle or forceful termination of program before getting output (solution of problem)
- Setup time
  - Program execution- loading the compiler, loading HLL code into memory, saving compiled code (object code) and then linking and loading
  - If error occurred, user has to go back to beginning and redo the setup sequence
  - Considerable amount of time wasted in just setting up the program to run

2. Simple Batch system

- No direct access of user with the machine required
- Used software called “Monitor”- OS
- User submits the jobs on cards/tape to computer operator
- Operator Batches the jobs together sequentially
- Places the entire batch on input device for the “Monitor”
- Monitor: Software that controls the sequence of events
    - Executes the Batch jobs
    - Program branches back (returns control) to monitor when finished
    - Monitor automatically loads next program

2. Simple batch system..

- Used Job Control Language (JCL)
  - Special type of programming language
    - Provides instruction to the monitor
    - What compiler to use; what data to use



2. Simple batch system Features:

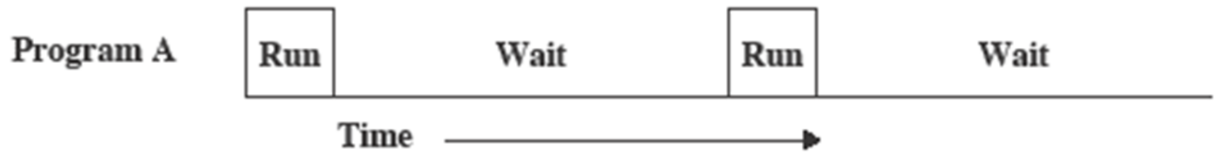
- Timer
  - Prevents a job from monopolizing the system
- Privileged instructions
  - Certain machine level instructions can only be executed by the monitor
- Interrupts
  - Early computer models did not have this capability
- Simple file management
- Memory protection

2. Simple batch system

- Memory protection
    - Does not allow the memory area containing the monitor to be altered
- User program executes in user mode
    - Certain instructions may not be executed
- Monitor executes in system mode
  - Kernel mode
    - Privileged instructions are executed
    - Protected areas of memory may be accessed

3. Multi-programmed Batch Systems

- Uniprogramming

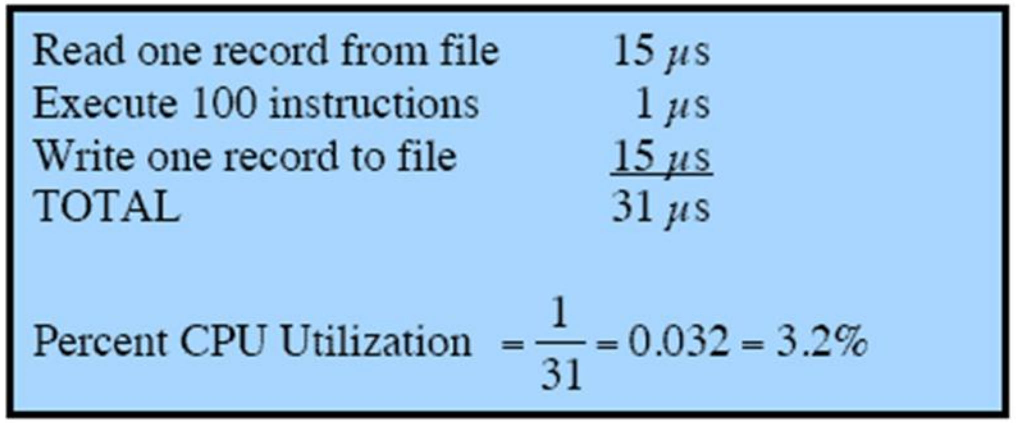


(a) Uniprogramming

- I/O devices slower as compared to processor
  - Processor must wait for I/O instruction to complete before proceeding
- Processor spends large amount of time waiting for I/O devices to finish transferring data to & from file

3. Multi-programmed Batch Systems

- Uniprogramming: System Utilization Example



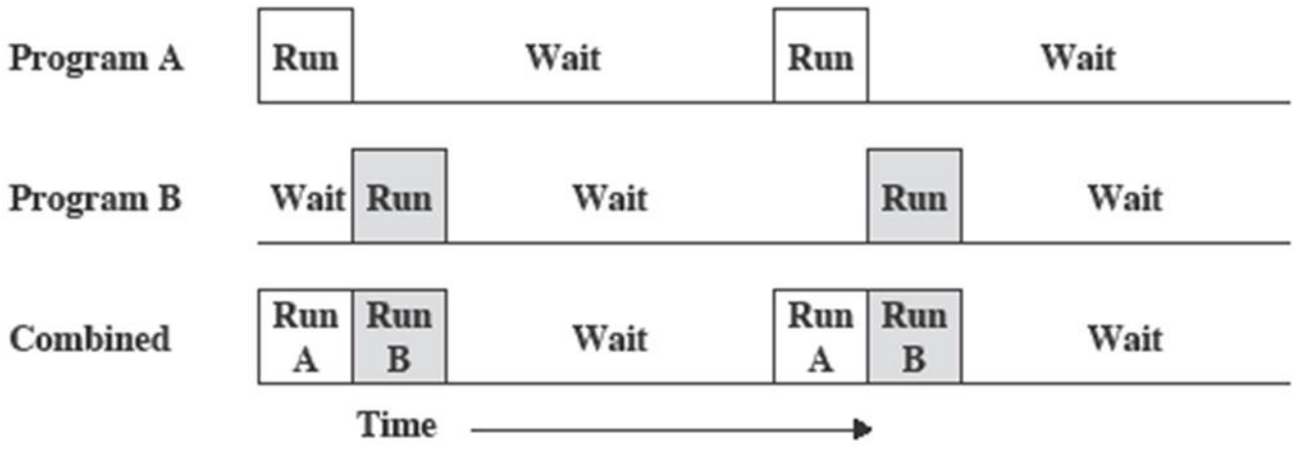
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

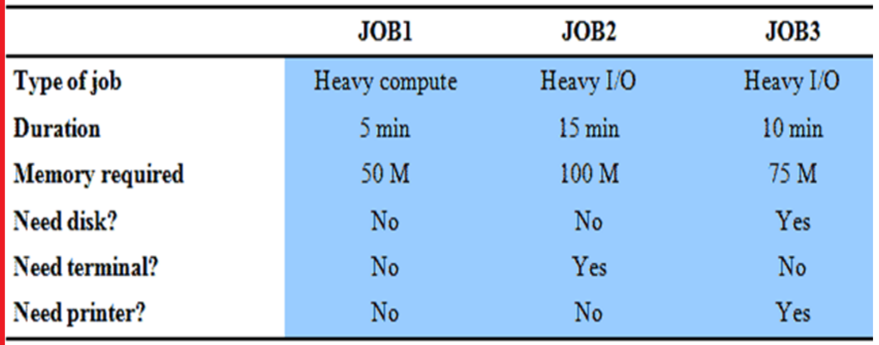
3. Multi-programmed Batch Systems

- Multiprogramming
  - Enough space for storing OS (or resident monitor) and multiple user programs in memory then,
  - If one job is waiting for I/O, the processor can switch to the other job (which is not likely waiting for I/O) →multiprogramming or multitasking

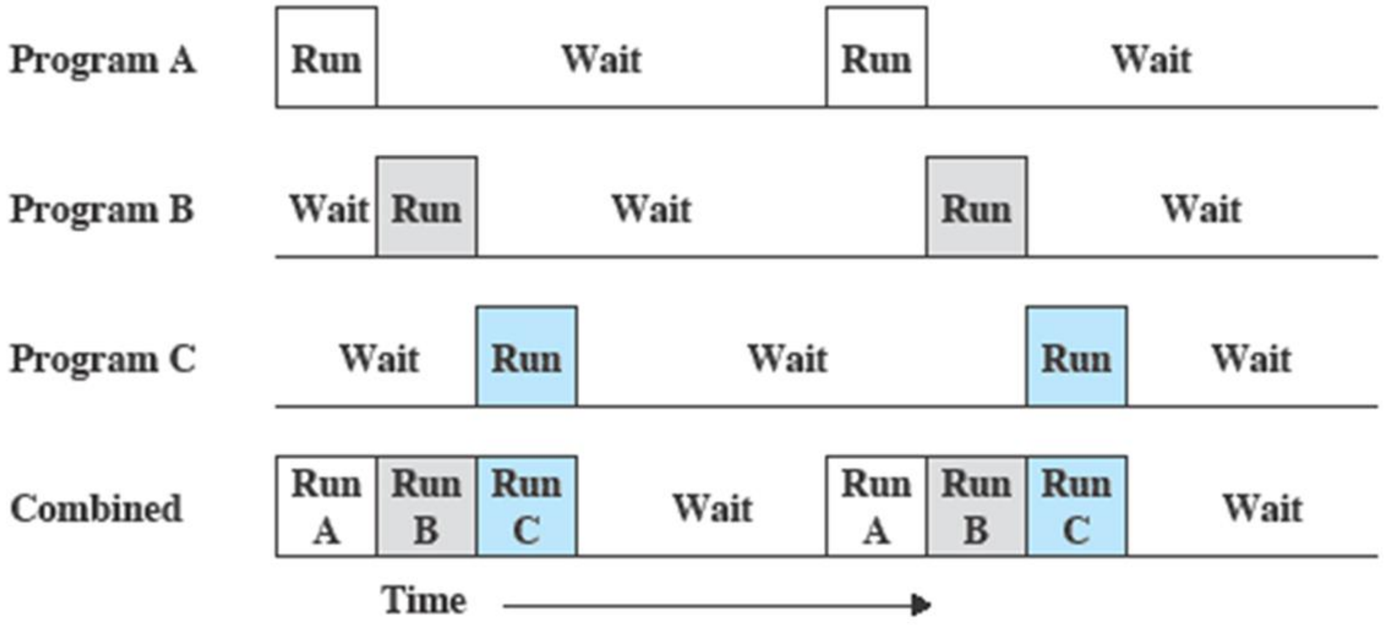


(b) Multiprogramming with two programs

3. Multi-programmed Batch Systems



	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes



(c) Multiprogramming with three programs

3. Multi-programmed Batch Systems

- With uniprocessing:
  - Job A takes 5 mins
  - Job B takes 20 mins and
  - Job C takes 30 mins - to complete from time of submission

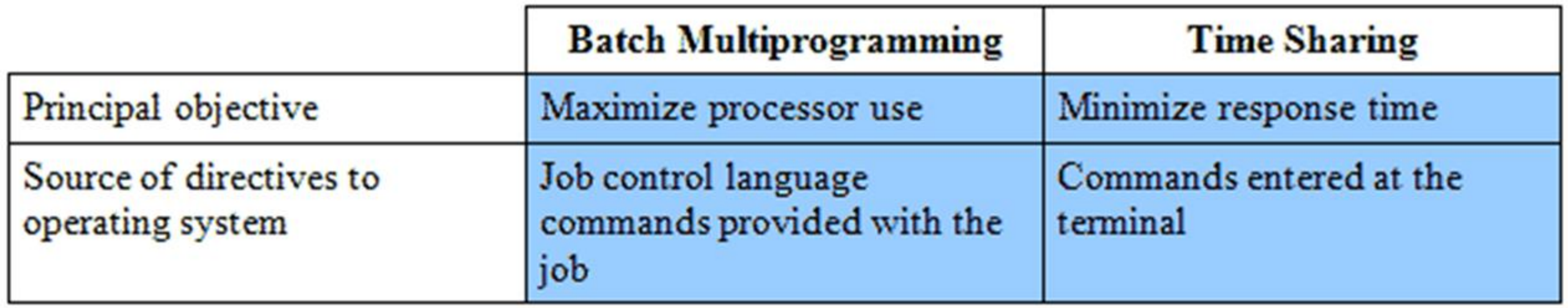
- With multiprocessing:
  - Job A takes 5 mins
  - Job B takes --? mins and
  - Job C takes --? mins to complete from time of submission

4. Time–Sharing Systems

- Many jobs require direct interaction with computer system
  - Transaction processing (interactive mode essential)
- Multiple users simultaneously access the system through terminals (e.g Mainframes)
- OS interleaves the execution of each user program in short burst (quantum) of computation
  - Processor's time is shared among multiple users -time sharing
    - If '*n*' users in the system; each user will have 1/*n* of effective computer time
- Multiprogramming to handle multiple interactive jobs

Batch Multiprogramming versus Time Sharing

Table 2.3 Batch Multiprogramming versus Time Sharing



	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

OS Design Structures

- Monolithic Systems
- Layered Systems
- Virtual Machines
- Client-Server Systems

OS Structure: Monolithic Systems

- No structure; Big mess
- OS written as collection of procedures
  - each can call any of the other procedures, whenever needed
- Each procedure has a well-defined interface in terms of parameters and results
  - each procedure is free to call any other, if the latter provides some useful computation needed by the former
- Program execution: For construction of object program, compile all individual procedures and then bind them together into single object file using system '**Linker**'
- No information hiding- every procedure is visible to every other procedure

OS Structure: Monolithic Systems

- Working:

1. User program executes 'kernel call' (i.e. calls the kernel)
2. OS determines which system call needs to be carried out
3. OS identifies the service procedure and calls it
4. Control is returned to user program

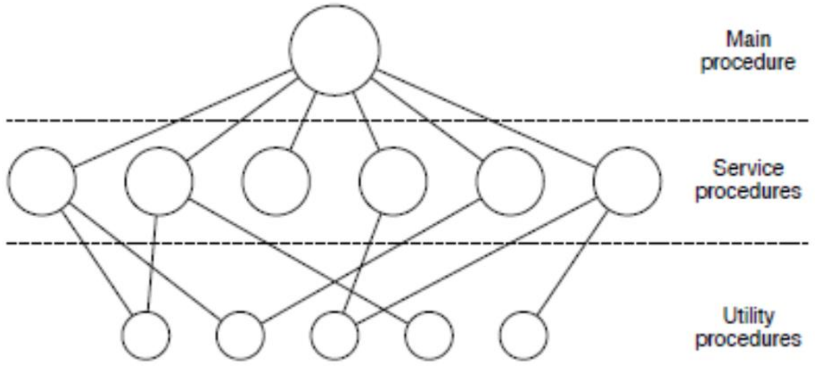


Figure 1-24. A simple structuring model for a monolithic system.

- Basic structure of such OS consists of:

I. A main program that invokes the requested service procedure
II. A set of service procedures that carry out the system call
III. A set of utility procedures that help the service procedures

- For each system call there is 1 service procedure
- Utility procedures perform things that are needed by several service procedures; such as- fetching of data from user programs etc.

OS Structure: Layered Systems

- OS organised as hierarchy of layers; each one constructed upon one below it
- “THE” system (Technische Hogeschool Eindhover, 1968)
- Simple batch system; consist 6 layers

Layer No.	Name
5	The Operator
4	User programs
3	I/O management
2	Operator-process communication
1	Memory and Drum management
0	Process allocation and multiprogramming

OS Structure: Layered Systems

- Layer 0 -Process allocation and multiprogramming
  - Allocation of processor, switching between processes on interrupts/expiry of timers
    - Provides basic multiprogramming of CPU
    - Above this layer, system consists sequential processes, each programmed for multiple processes running on single processor
- Layer 1 -Memory and Drum management
  - Responsible for memory mgmt- allocates space for processes in main memory and on 512 K word drum [used for holding parts of processes (pages) which had no room in main memory]
- Layer 2 -Operator-process communication
  - Handles communication between process and operator console
    - Above this layer each process has its own console

OS Structure: Layered Systems

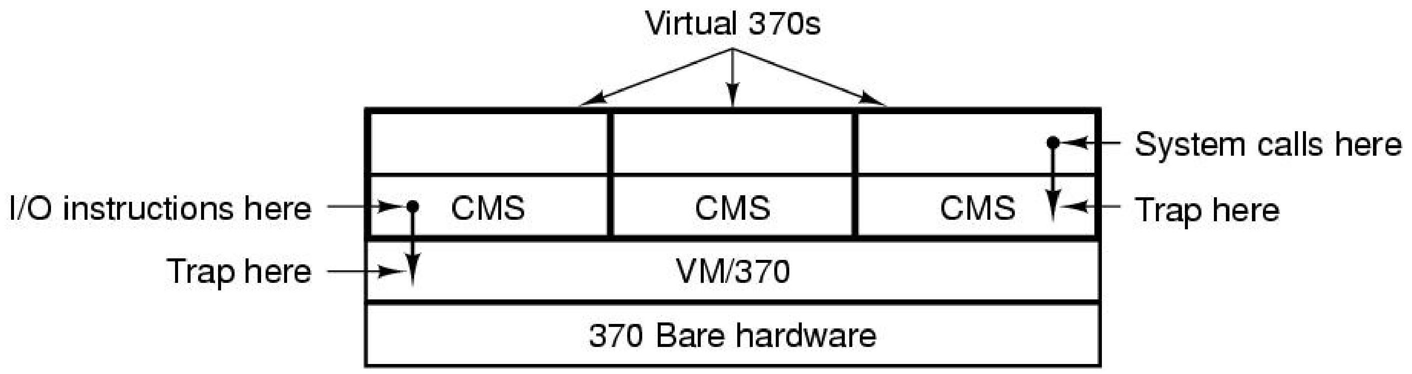
- Layer 3 -I/O management
  - Management of I/O devices and buffering the information streams to and from them
  - Above this layer each process can deal with abstract I/O devices instead of real devices
- Layer 4 -User programs
  - Layer where user programs reside; user programs need not have to worry about process, memory, console or I/O management
- Layer 5 -The Operator
  - Location of system operator process

- Layering concept was generalized in OS named MULTICS
  - OS organized as series of concentric rings; inner layers more privileged than outer ones
  - TRAP instruction

OS Structure: Virtual Machines

- Virtual Machines are basically time shared systems
    - Originally called as CP/CMS (conversational monitor system)- renamed as VM/370
- Time sharing systems provide Multiprogramming and extended machine with more convenient interface than bare hardware
- VMM (virtual machine monitor)- Heart of system; runs on bare hardware and does the multiprogramming, providing several virtual machines to upper layer
- These VMs are not extended machines, with files and other features; rather they are exact copies of the bare hardware including kernel/user mode, i/o, interrupts and everything that a real machine has
- Each VM is identical to true hardware, and each can run any OS on bare h/w→ Different VMs can run different OS

OS Structure: Virtual Machines



- CMS program executes a system call; the call is trapped to the OS in its own VM and not to VM/370
  - Just as it would if it were running on real machine instead of virtual
- CMS then issues the normal hardware instruction for reading its virtual disk or whatever resource is needed to carry out the call
  - The i/o instructions are trapped by VM/370, which then performs them as part of its simulation of real hardware

Client-Server Systems

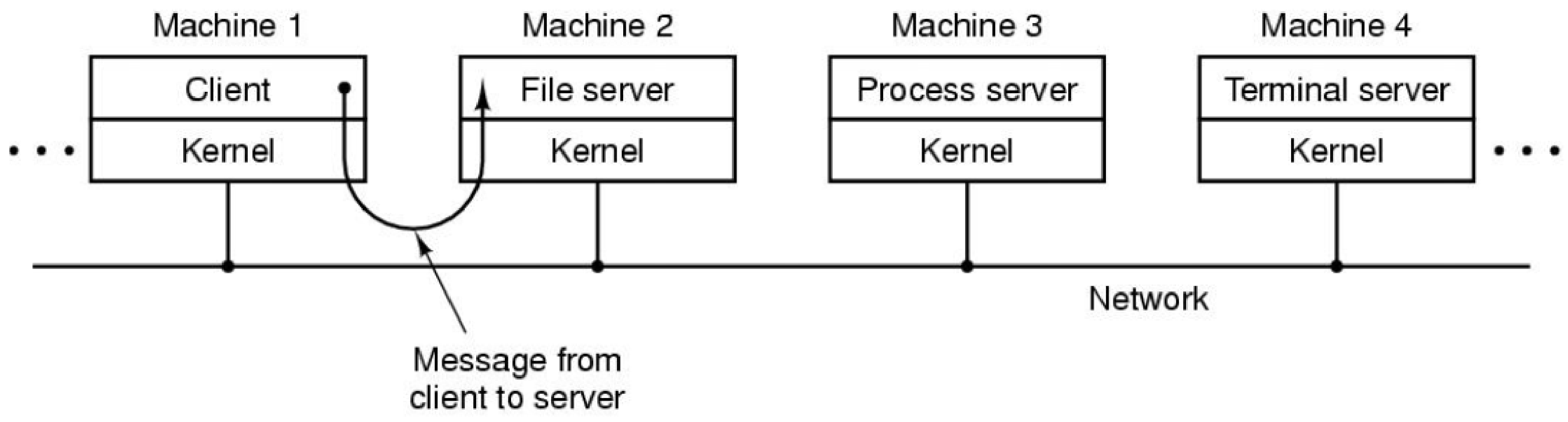
- Principle: Move most of the functionalities to higher layers; remove as much as possible from OS→ minimal kernel
    - Implement most of OS functions as user programs
- To request a service, user process (called client process) sends the request to a server process
- Server process does the work and sends back the result to client
    - E.g Reading a block of files- client process requests the File server
- OS is splitted into parts: each handling one facet of the system such as: File server, process server..etc
    - →Easier to manage

Client-Server Systems

- All Servers run in user-mode and not in kernel mode- they do not have direct access to the h/w
  - If bug in file server is triggered, the file service may clash but will not bring whole system down

- Client-server approach can be readily adopted in distributed systems

Client-Server Systems



- When a client process communicates with a server process by sending msgs, it is not aware whether the msg is handled locally in its own machine or it is sent across the network to a server on a remote machine

Modern Operating Systems

- Modifications and enhancement of architectures and new organization of OS due to demands of users/applications

- Different approaches & design elements of OS:
  - Microkernel Architecture
  - Multithreading
  - Symmetric Multiprocessing
  - Distributed OS
  - Object oriented design

Modern Operating Systems: Microkernel Architecture

- Monolithic OS has large kernel with all functionalities included
- Microkernel architecture assigns only few essential functions to the kernel
  - Address space
  - Inter-process communication (IPC)
  - Basic scheduling
- Other services provided by processes called '**servers**', that run in user mode and are treated like any other application by the µkernel
- Decoupling between kernel and server development → simplifies implementation, provides flexibility and well suited for a distributed environment

Modern Operating Systems: Microkernel Architecture

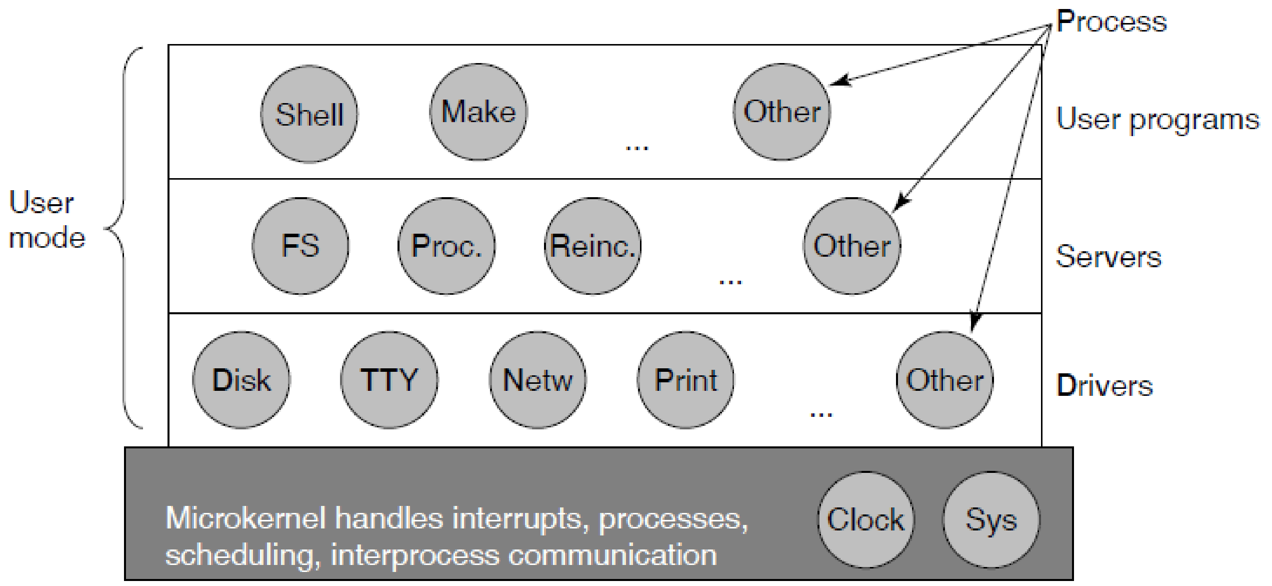


Figure 1-26. Simplified structure of the MINIX system.

Modern Operating Systems: Multithreading

- Technique in which process executing an application is divided into threads that can run concurrently
- Thread- dispatchable unit of work
  - Includes processor context (PC & SP) and its own data area for a stack
  - Executes sequentially and is interruptable so that process can turn to another thread
- Process: collection of one or more threads and associated system resources (e.g. memory with code and data, open files and devices)
- Advantages:
- Programmer has great control over the modularity of the application and timing of application related events
- Multithreading useful for applications that perform number of independent tasks that need not be serialized
  - e.g. Database server that listens to and processes number of client requests

Symmetric Multiprocessing

- Computers with Multiple processors
- SMP – standalone computer system with following characteristics:
  1. Multiple processors
  2. Processors share same main memory and i/o; interconnected by communication bus /other connection
  3. All processors can perform same functions
- Existence of multiple processors is transparent to users
  - OS takes care of scheduling of threads or processes on individual processors and synchronization among processors
- OS of SMP:
  - Schedules processes or threads across all the processors
  - Provides tools and functions to exploit parallelism in SMP system

Distributed OS

- Used in cluster computers/ multi-computers
- Illusion of single main memory and secondary memory space;
- + Unified access facilities such as distributed file systems

Object oriented Design

- Provides discipline for adding modular extensions to small kernel
- OO structure enables programmers to customize the OS without disrupting system integrity

Evolution of Operating System

- Mainframe System
    - Batch Systems
    - Multiprogrammed Systems
    - Time Sharing Systems
- Desktop Systems
    - Multiprocessor Systems
        - Symmetric Multiprocessor
        - Asymmetric Multiprocessor
    - Distributed Systems
        - Client Server Systems
        - Peer to Peer Systems
    - Clustered Systems
        - Asymmetric clustering
        - Symmetric clustering
    - Real Systems
        - Hard Real Time System
        - Soft Real Time System
- HandHeld Systems

Evolution of OS

1. **Early Systems (1940s-1950s) - Batch Processing Systems**:
   - These systems processed one job at a time in a batch mode.
   - Jobs were collected and executed sequentially without user interaction.
   - Examples: IBM 701, UNIVAC.
2. **Simple Batch Systems (1950s-1960s) - Resident Monitor:**
   - An early form of the OS that resided in memory to handle job sequencing.
   - Jobs were processed in batches with minimal manual intervention.
   - Examples: IBM 1401.
3. **Single-User, Single-Tasking Systems (1960s-1970s):**
4. **Multiprogramming Systems (1960s):**
   - Allowed multiple programs to be loaded into memory and executed concurrently by the CPU.
   - Examples: IBM System/360, CTSS (Compatible Time-Sharing System).
5. **Time-Sharing Systems (1960s-1970s):**
   - Examples: MULTICS (Multiplexed Information and Computing Service), Unix

Evolution of OS (continued)

6. **Single-User, Multiprocessing Systems (1970s-1980s):**
   - Examples: Early versions of macOS (Classic Mac OS), MS-DOS with TSR (Terminate and Stay Resident) programs.
7. **Multi-User, Multiprocessing Systems (1970s-1980s):**
   - Examples: Unix, VMS (Virtual Memory System).
8. **Personal Computer Operating Systems (1980s-1990s) - Single-User, Multiprocessing:**
   - Graphical User Interfaces (GUIs) gained popularity, making computers more accessible.
   - Examples: Windows 95, Mac OS.

Evolution of OS (continued)

9. **Network Operating Systems (1980s-1990s)**:
    - Enabled file sharing, printer sharing, and inter-process communication over networks.
    - Examples: Novell NetWare, Windows NT, early Unix-based systems with networking extensions.
10. **Distributed Operating Systems (1990s):**
    - Managed independent computers as a single coherent system.
    - Examples: Amoeba, Plan 9, early versions of Linux with distributed capabilities.
11. **Modern Operating Systems (2000s-Present) - Multi-User, Multiprocessing Systems:**
    - Support advanced multitasking, multi-user environments, and extensive networking.
    - Emphasize security, stability, and user-friendly interfaces.
    - Examples: Modern Windows, macOS, Linux distributions, Android, iOS.

Evolution of OS (continued)

12. **Mobile Operating Systems (2000s-Present)**:
    - Designed for mobile devices with touch interfaces, efficient power management, and connectivity.
    - Emphasize app ecosystems and seamless user experiences.
    - Examples: iOS, Android.
13. **Cloud and Virtualization (2010s-Present)**:
    - Examples: VMware, Hyper-V, Kubernetes, cloud-based OS like Google Chrome OS.

Evolution of Operating System

- Within the broad family of operating systems, there are generally seven types, categorized based on the types of computers they control and the sort of applications they support.
- The categories are
  - Single User Single Task
  - Single User Multi-Tasking
  - Multi-user
  - Distributed
  - Multiprocessing
  - Parallel
  - RTOS

Single User Single Task

Designed for one user to perform one task at a time.

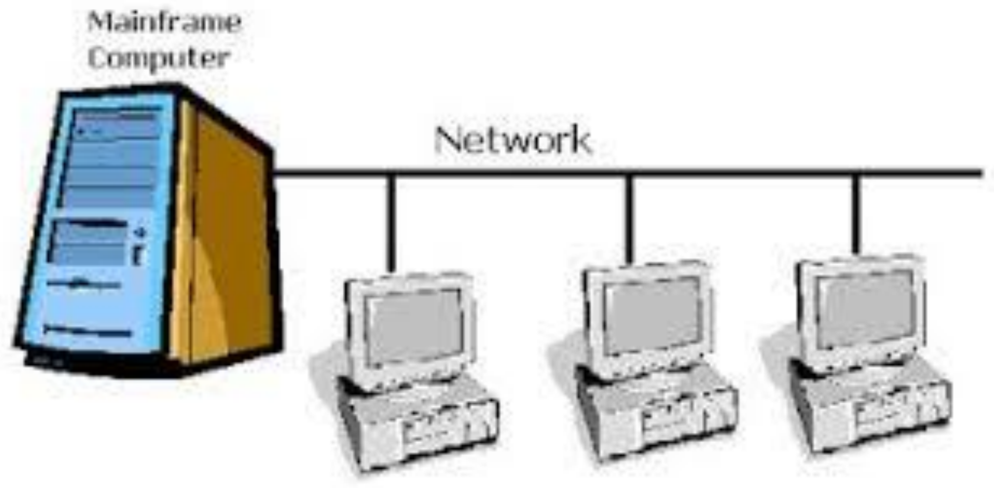
- **Example**
  - **MS-DOS**: The user can only execute one program at a time, such as typing in Notepad or executing a command in the terminal.
  - **Early Mobile phones** - There can only be one user using the mobile and that person is only using one of its applications at a time.

Single User Multi-tasking

- Allows one user to run multiple applications simultaneously.
- **Example**:
  - **Microsoft Windows**: A user can browse the internet, write a document in MS Word, and play music at the same time.
  - **MacOS**: Similarly supports multitasking for a single user.

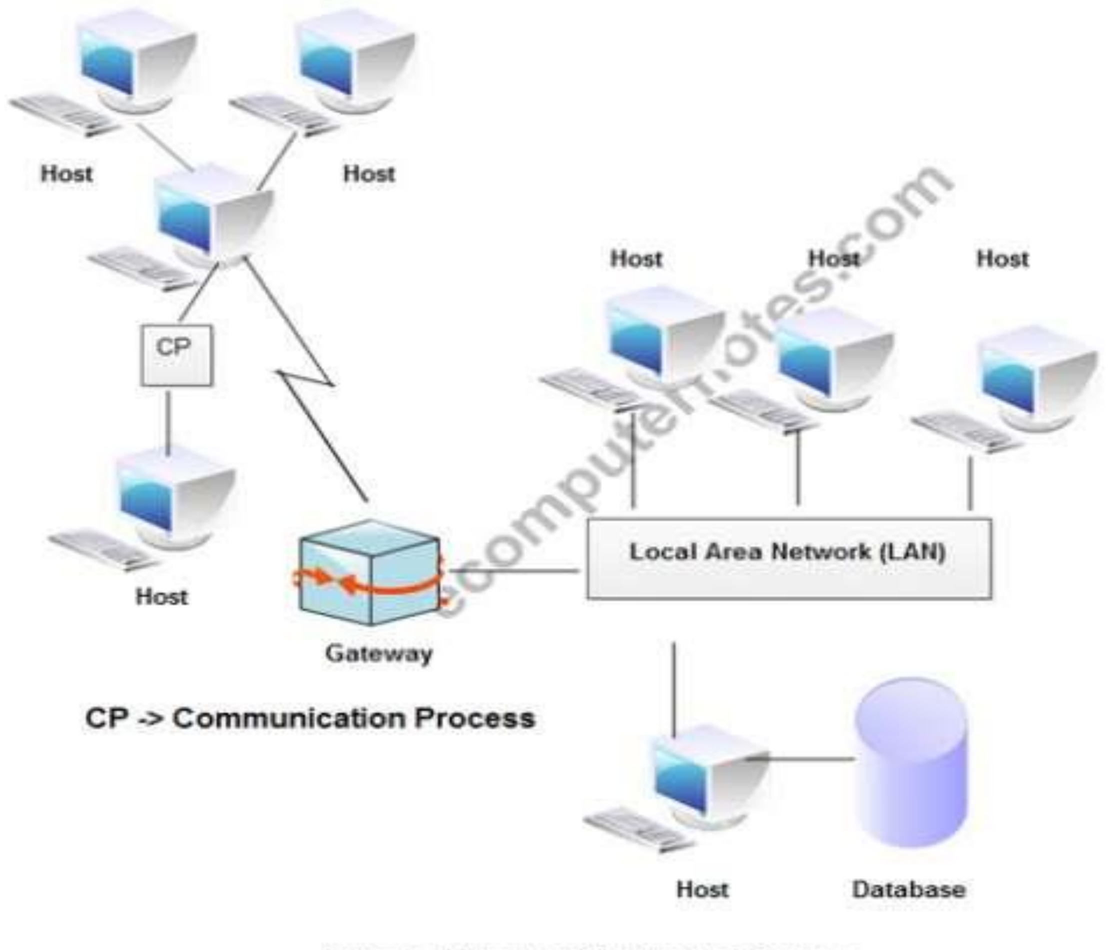
Multi-user

- Enables multiple users to access a single system's resources simultaneously.
- **Example**:
  - **Unix/Linux Servers**: Multiple users can log in via terminals and execute tasks like programming or file management concurrently.



Distributed

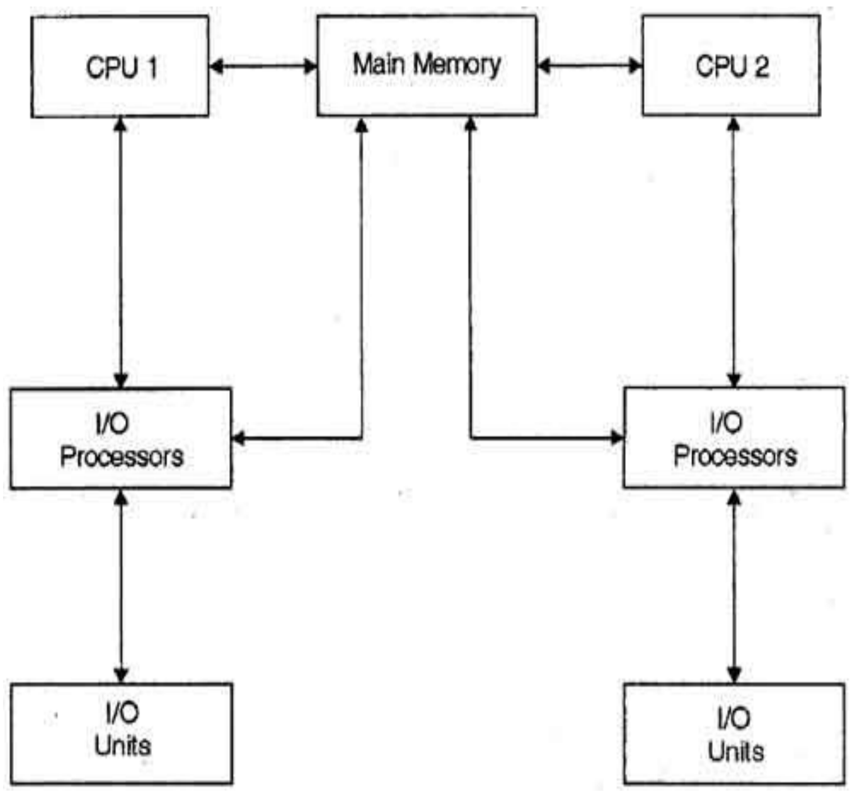
- A system where computing resources are distributed across multiple machines but appear as a single cohesive system to the user.
- **Example**:
  - **Google's Android System**: Combines cloud and local resources.
  - **Windows Server with Distributed Computing**: Data and processes are shared among multiple nodes.



A Typical View of Distributed System

Multiprocessing

- Utilizes multiple CPUs for faster execution of tasks.
- **Example**:
  - **Linux SMP (Symmetric Multiprocessing)**: Allows efficient distribution of tasks across multiple CPUs.
  - **Unix-based Systems**: Designed for multiprocessing.



Parallel

- Specifically designed for systems that execute tasks in parallel using multiple processors.
- **Example**:
  - **IBM's Blue Gene**: Used in supercomputers for high-performance computing.
  - **Cray OS**: Designed for parallel computation in advanced research.

Real Time Operating System (RTOS)

- Processes data and provides responses within a defined time frame. Commonly used in time-critical systems
  - Hard real-time OS
  - Soft real-time OS
- **Example**:
  - **FreeRTOS**: Used in embedded systems.
    - Amaxon Echo, Fitbit, Tesla
  - **VxWorks**: Used in aerospace systems.
    - Mars Rover, Boeing 787

Quiz

Which of the following is an example of a Single-User, Multi-Tasking Operating System?

a) MS-DOS

b) Windows

c) VxWorks

d) Unix

Quiz

What is a key characteristic of a Multi-User Operating System?

a) It allows a single user to run multiple tasks simultaneously.

b) It processes data in real-time.

c) It allows multiple users to access the same system resources concurrently.

d) It is used for supercomputers.

Quiz

Which type of operating system uses multiple CPUs for faster execution of tasks?

a) Single-User, Single-Task

b) Multiprocessing

c) Real-Time Operating System

d) Parallel

Quiz

What is the main advantage of a Distributed Operating System?

a) Faster execution of tasks using multiple processors.

b) Real-time data processing.

c) Efficient resource sharing across multiple machines.

d) High compatibility with single-user systems.

Types of Architectures - Operating System

- Monolithic Architecture
- Layered Architecture
- Micro-Kernel Architecture
- Hybrid Architecture

Types of Architectures - Operating System

- Monolithic Architecture
- Layered Architecture
- Micro-Kernel Architecture
- Hybrid Architecture

Monolithic Architecture

- Each component of the **operating system is contained in the kernel** and the components of the operating system **communicate with each other using function calls**.
- E.g. OS/360, VMX, and LINUX.

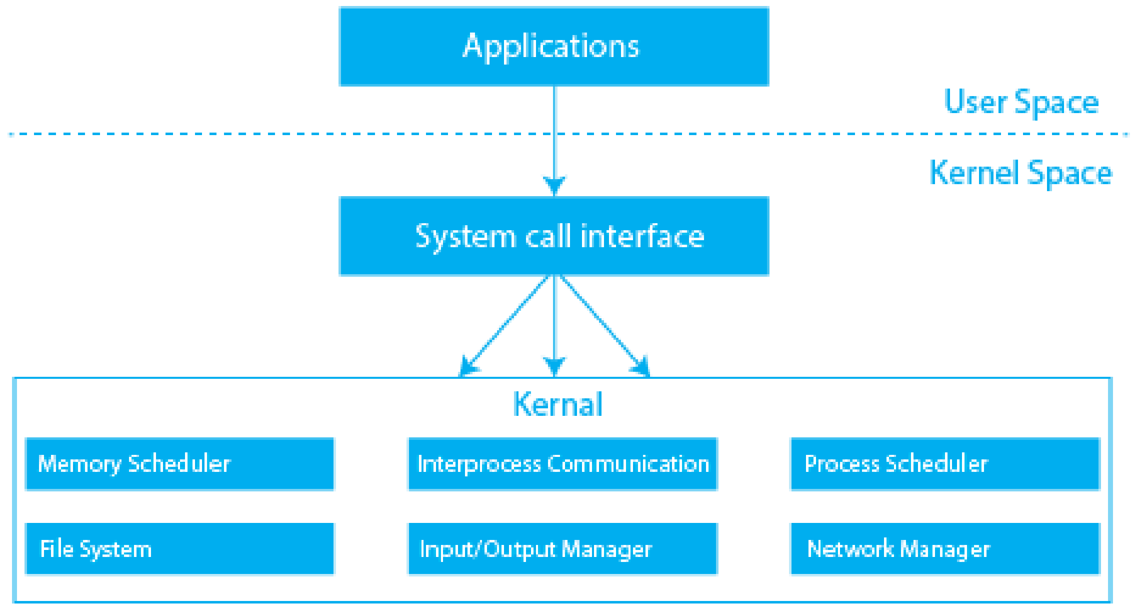


Image source : Operating System Architecture (prepbytes.com)

Monolithic Architecture (cont…)

- **Performance**:
  - High due to direct function calls within the kernel.
- **Modularity**:
  - Low, as all components are intertwined in a single large codebase.
- **Maintenance**:
  - Difficult, because changes in one part can affect many others.
- **Security**:
  - Lower, because a bug in any part of the kernel can compromise the entire system.

Layered Architecture

- The OS is separated into layers or levels in this kind of arrangement.
- Layer 0 (the lowest layer) contains the hardware, and layer 1 (the highest layer) contains the user interface (layer N).
- These layers are organized hierarchically, with the top-level layers making use of the capabilities of the lower-level ones.
- E.g. Windows XP, and LINUX

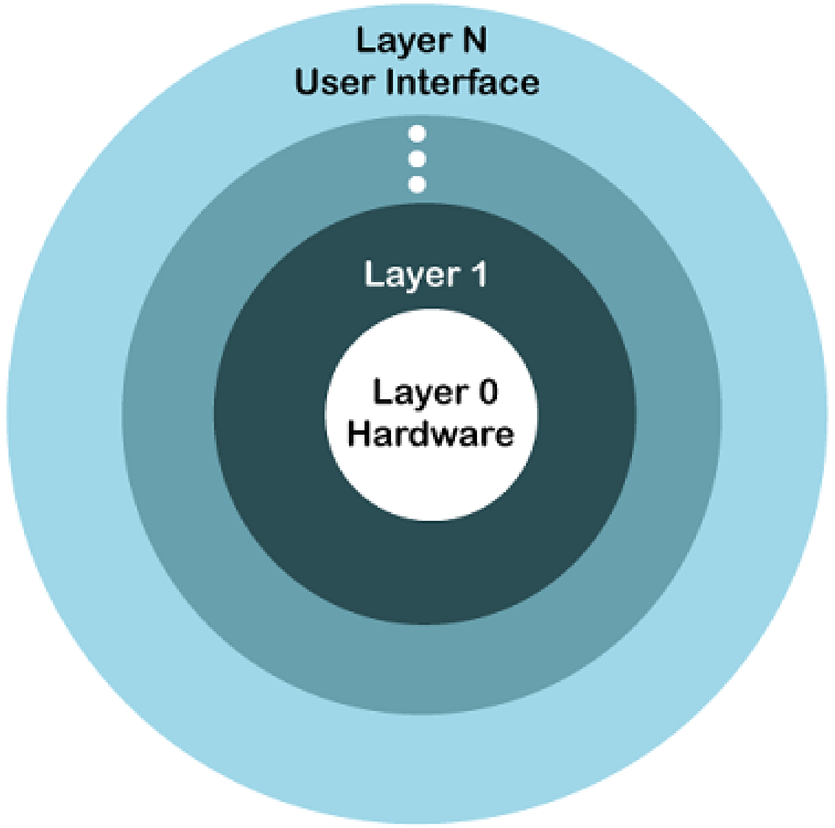
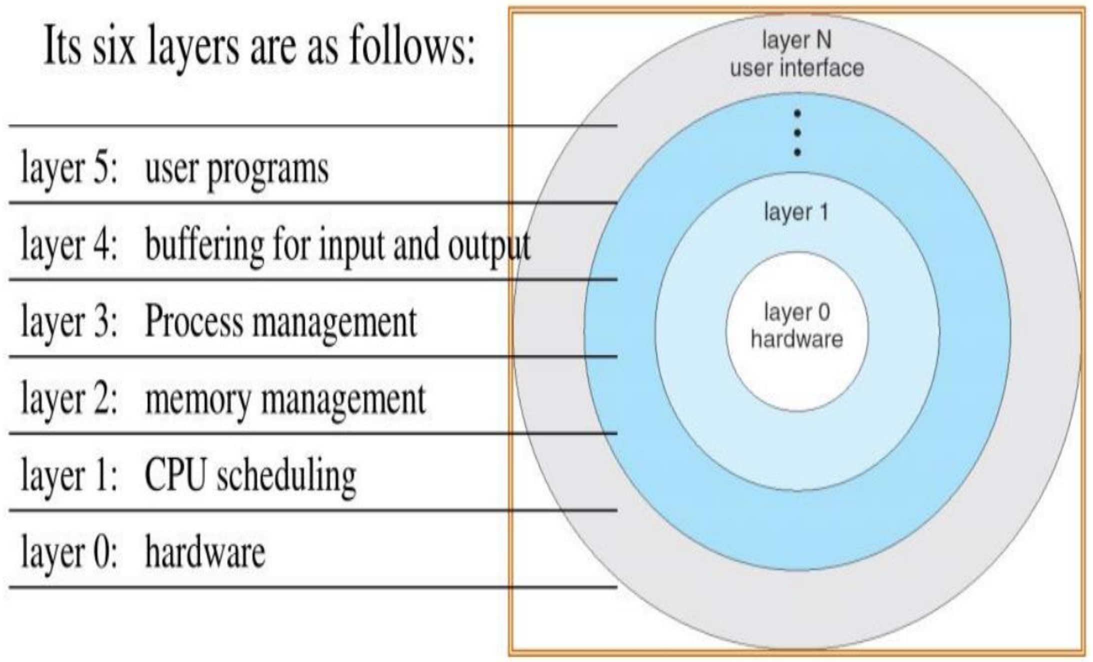


Image source : Operating System Structure - javatpoint

Layered Architecture (cont…)

- **Performance**:
  - Moderate, as each layer adds overhead.
- **Modularity**:
  - Higher than monolithic, since each layer has specific functionality.
- **Maintenance**:
  - Easier than monolithic, as changes are localized within layers.
- **Security**:
  - Improved compared to monolithic, but a bug in lower layers can still affect upper layers.


Its six layers are as follows:

layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	Process management
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

Image source : [Information to know about Operating System – Programmer Prodigy (code.blog)]

Microkernel Architecture

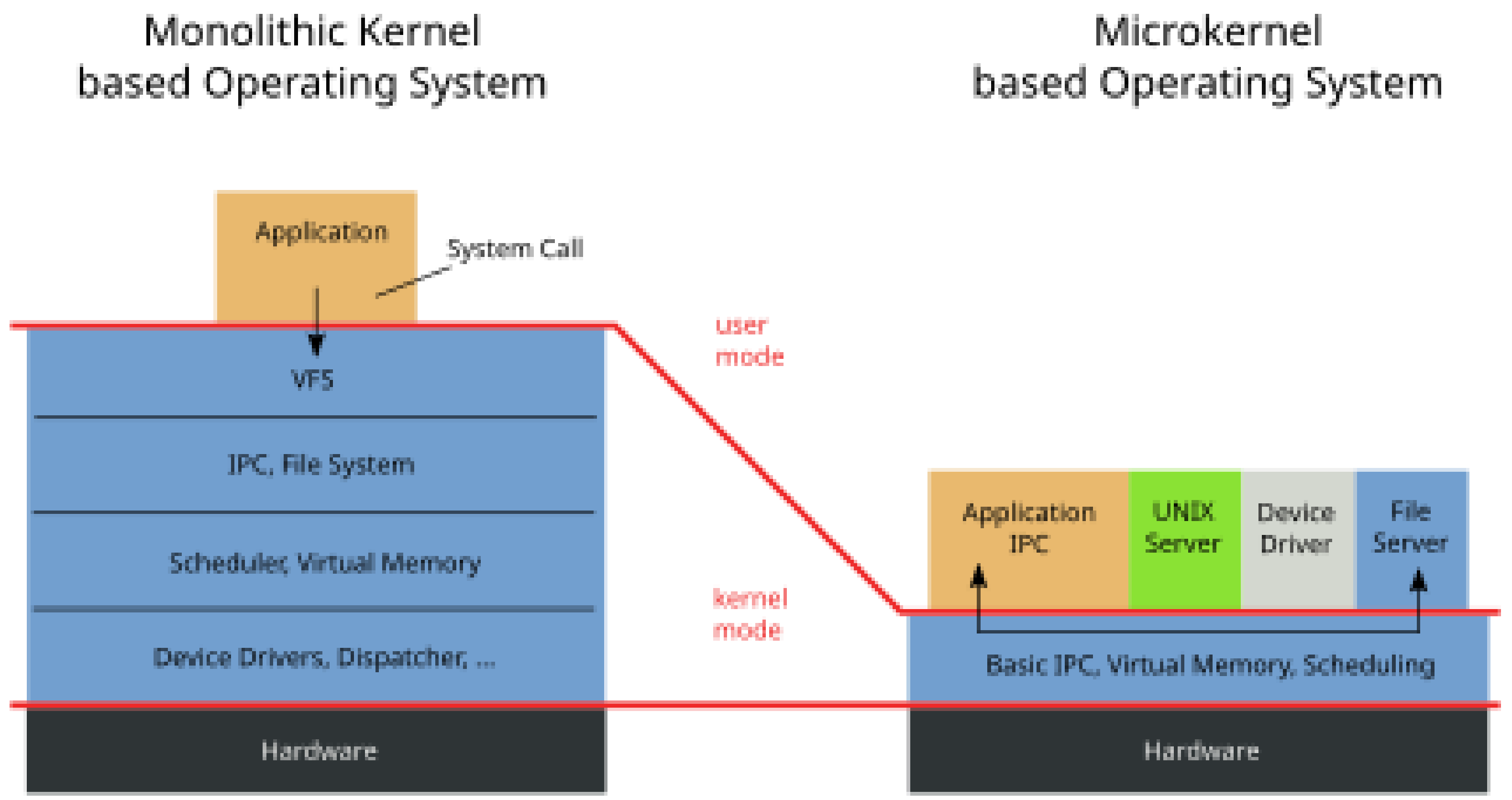

Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Image source: Wikipedia

- The microkernel architecture separates the basic functions of the kernel (like communication between hardware and software) from higher-level services
- **Minix** and **QNX;** run the most essential services in the kernel space and other services in user space.

Microkernel Architecture

- **Performance**:
  - Lower due to more context switches and inter-process communication.
- **Modularity**:
  - Very high, as only essential services are in the kernel and others run in user space.
- **Maintenance**:
  - Easiest, as most services are user-space programs that can be updated independently.
- **Security**:
  - Highest, because faults in user-space services do not affect the kernel.

Microkernel vs Monolithic

S. No.	Parameters	Microkernel	Monolithic kernel
1.	Address Space	user services and kernel services are kept in separate address space.	both user services and kernel services are kept in the same address space.
2.	Design and Implementation	OS is complex to design.	OS is easy to design and implement.
3.	Size	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.
4.	Functionality	Easier to add new functionalities.	Difficult to add new functionalities.
5.	Coding	To design a microkernel, more code is required.	Less code when compared to microkernel

Microkernel vs Monolithic (cont…)

S. No.	Parameters	Microkernel	Monolithic kernel
6.	Failure	Failure of one component does not effect the working of micro kernel.	Failure of one component in a monolithic kernel leads to the failure of the entire system.
7.	Processing Speed	Execution speed is low.	Execution speed is high.
8.	Extend	It is easy to extend Microkernel.	It is not easy to extend monolithic kernel.
9.	Communication	To implement IPC messaging queues are used by the communication microkernels.	Signals and Sockets are utilized to implement IPC in monolithic kernels.
10.	Debugging	Debugging is simple.	Debugging is difficult.

Microkernel vs Monolithic (cont…)

S. No.	Parameters	Microkernel	Monolithic kernel
11.	**Maintain**	It is simple to maintain.	Extra time and resources are needed for maintenance.
12.	**Message passing and Context switching**	Message forwarding and context switching are required by the microkernel.	Message passing and context switching are not required while the kernel is working.
13.	**Services**	The kernel only offers IPC and low-level device management services.	The Kernel contains all of the operating system’s services.
14.	**Example**	**Example :** Mac OS X.	**Example :** DOS, classical early versions of BSD, Unix, Solaris, Mac OS, etc.

Hybrid Architecture

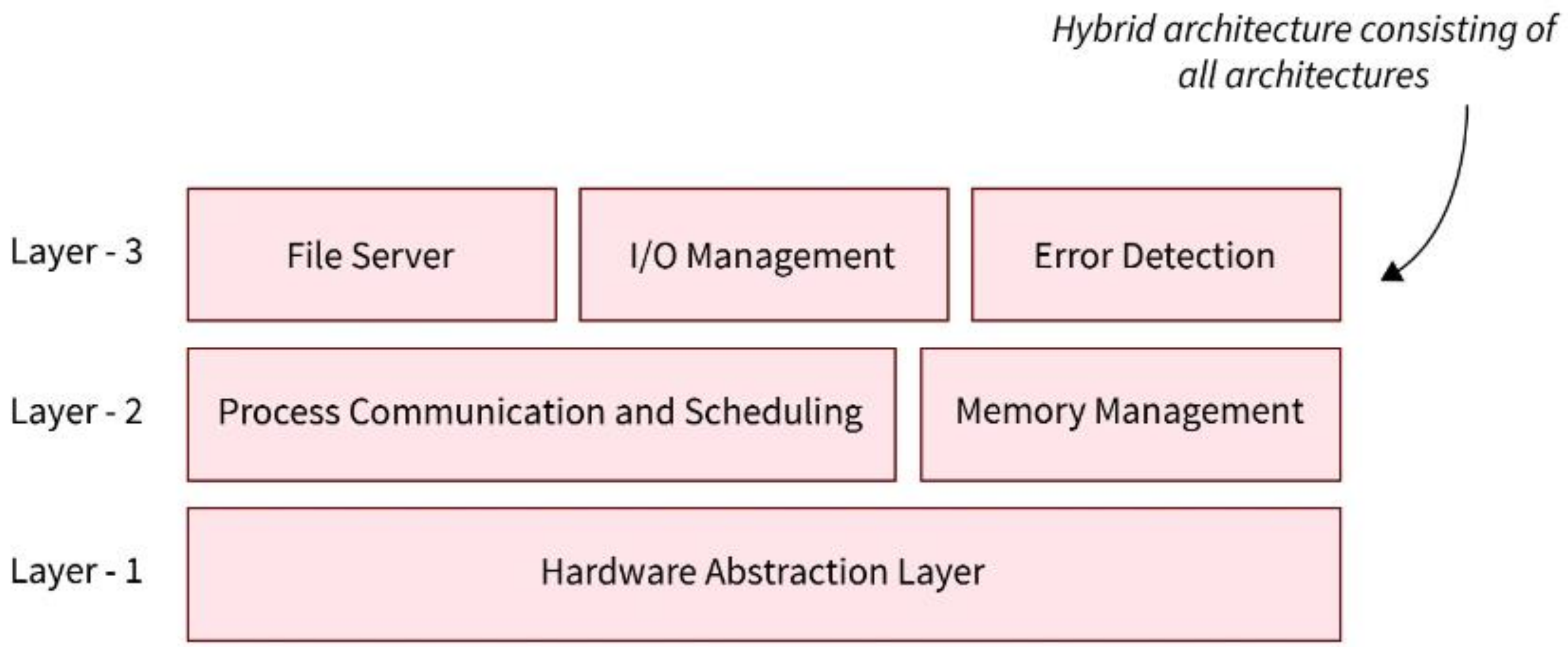

Hybrid architecture consisting of all architectures
Layer - 3
File Server
I/O Management
Error Detection
Layer - 2
Process Communication and Scheduling
Memory Management
Layer - 1
Hardware Abstraction Layer

Image source: Architecture of Operating System - Scaler Topics

- Combines elements of monolithic and microkernel architectures to leverage the benefits of both
- E.g. **Windows NT**; combines the performance of monolithic kernels with the modularity of microkernels.

Hybrid Architecture (cont…)

- **Performance:**
  - They allow various architectural components to provide specialized services.
  - This flexibility can lead to better overall system performance.
- **Modularity:**
  - Each layer focuses on specific functionality (e.g., file system, memory management).
  - Developers can work on individual layers independently.
- **Maintenance:**
  - Easier maintenance due to clear module boundaries.
  - Updates or bug fixes can be applied to specific layers without affecting the entire system.
- **Security:**
  - Separating critical services (kernel space) from less critical ones (user space) enhances security.
  - Kernel-level services are protected from user-level code.

Current OS designs

- **Windows NT and successors (2000, XP, Vista, 7, 8, 10, 11)**: Hybrid kernel with microkernel elements.
- **macOS (and iOS, iPadOS, watchOS, tvOS)**: Based on the XNU kernel, combining Mach microkernel and FreeBSD components.
- **Linux distributions**: Monolithic kernel with dynamically loadable modules, offering hybrid-like flexibility.
- **Android**: Built on the Linux kernel, following a similar modular approach.
- **Solaris**: Primarily monolithic but incorporates some microkernel principles with loadable kernel modules.

Quiz

In which among the given architecture executes different components of an operating system separately?

A. Monolithic Architecture
B. Layered Architecture
C. Microkernel Architecture
D. None of the given

OS Design Considerations for Symmetric Multiprocessor

- Simultaneous concurrent processes or threads
- Scheduling
- Synchronization
- Memory management
- Reliability and fault tolerance

OS Design Considerations for Symmetric Multiprocessor

Simultaneous concurrent processes or threads

- **Kernel routines need to be re-entrant**
  - **to allow several processors to execute the same kernel code simultaneously**

- With multiple processors executing **the same or different parts of the kernel,**
  - **kernel tables and management structures must be managed properly**
  - **to avoid data corruption or invalid operations**

OS Design Considerations for Symmetric Multiprocessor

Scheduling

- Any processor may perform scheduling,
  - complicates the task of enforcing a scheduling policy and assuring that **corruption of the scheduler data structures is avoided**

- If kernel-level multithreading is used, then
  - opportunity exists to schedule **multiple threads from the same process simultaneously on multiple processors**

OS Design Considerations for Symmetric Multiprocessor

Synchronization

- Multiple active processes have potential access to shared address spaces or shared I/O resources
- →care must be taken to provide effective synchronization
- Synchronization is a facility that enforces
  - **mutual exclusion and**
  - **event ordering**

- A common synchronization mechanism used in multiprocessor operating systems is **’Locks’**

OS Design Considerations for Symmetric Multiprocessor

Memory management

- Memory management on a multiprocessor must deal with **all of the issues found on uniprocessor computers**
- In addition, the OS needs to **exploit** the available **hardware parallelism** to achieve best performance
  - The **paging mechanisms** on different processors must be coordinated to **enforce consistency** when several processors share a page or segment and to decide on page replacement
  - The reuse of physical pages is the biggest problem of concern;
    - it must be guaranteed that a **physical page can no longer be accessed with its old contents before the page is put to a new use**

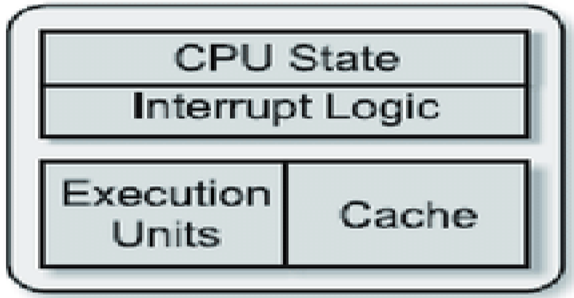
OS Design Considerations for Symmetric Multiprocessor

Reliability and fault tolerance

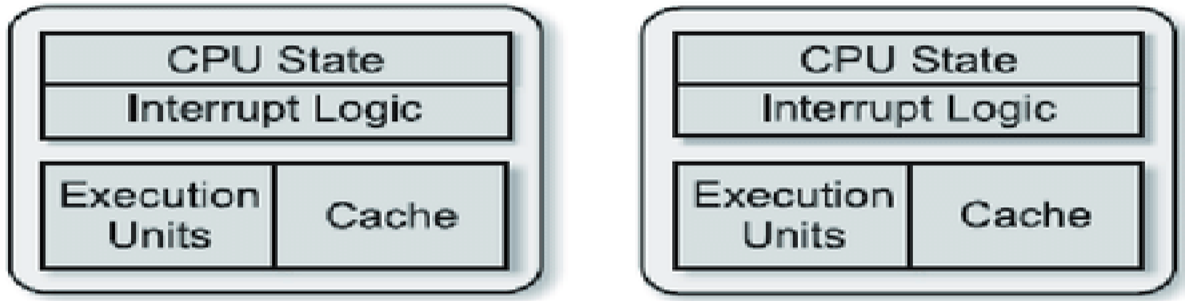
- The OS should provide **graceful degradation** in the case of processor failure
- The scheduler and other portions of the OS
  - **must recognize the loss of a processor** and
  - **restructure management tables accordingly**

- Multiprocessor OS design issues generally involve extensions to solutions of **multiprogramming uniprocessor design problems,**
  - →**we do not treat multiprocessor operating systems separately**

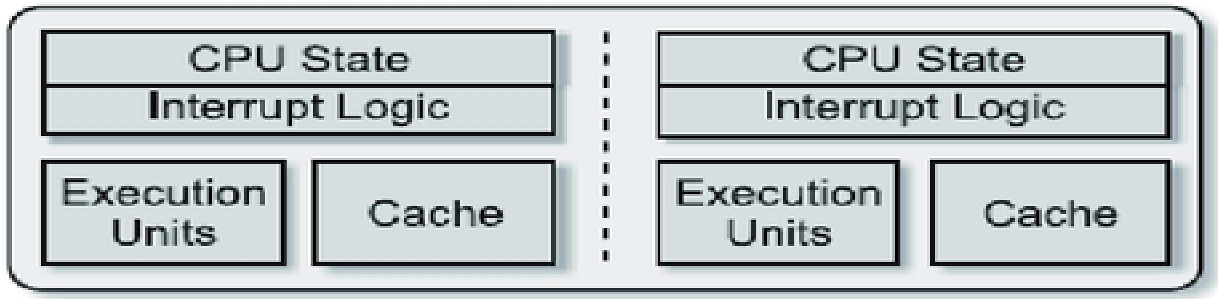
Multicore



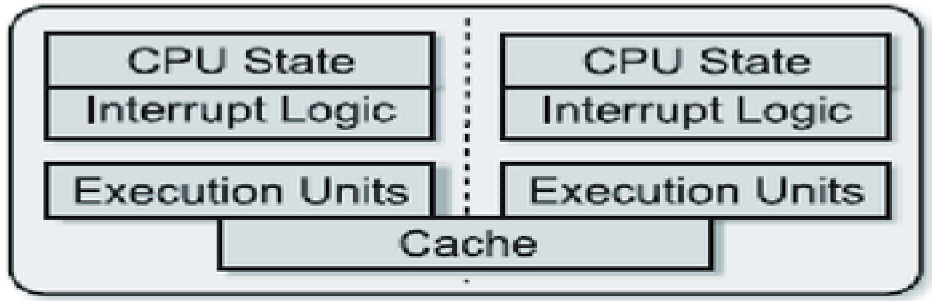
(a) Single core



(b) Multiprocessor



(c) Multi-core

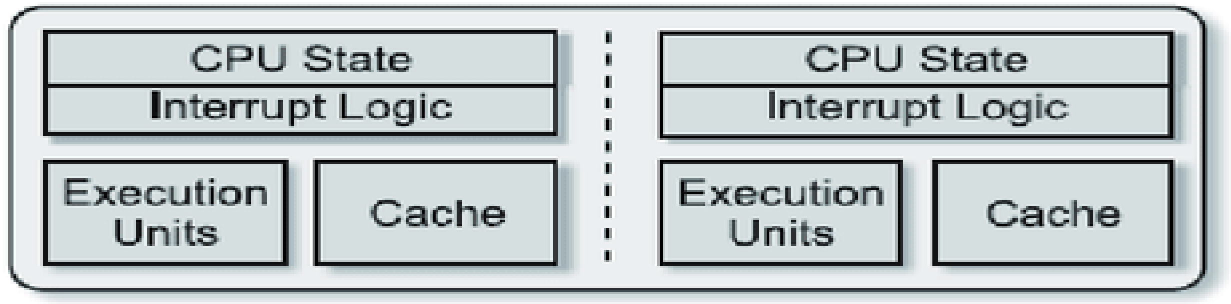


(d) Multi-core with shared cache

https://www.researchgate.net/figure/Simple-comparison-of-single-core-multi-processor-and-multi-core-architectures-after_fig3_266489075

Multicore

- The cores of a multicore processor can individually read and execute program instructions at the same time
- increases the speed of execution of the programs and supports parallel computing
- Eg: Quadcore processors, Octacore processors, etc.



(c) Multi-core

OS Design Considerations for Multicore architectures

- Current multicore vendors offer **systems with up to eight cores on a single chip**
  - With each succeeding processor technology generation,
  - **the number of cores and the amount of shared and dedicated cache memory increases → many core systems**

- The considerations for multicore systems include
  - all the design issues discussed so far in this section for SMP systems
- But additional concerns arise
  - The issue is one of the **scale of the potential parallelism**

OS Design Considerations for Multicore architectures

PARALLELISM WITHIN APPLICATIONS

- Applications can be subdivided into multiple tasks that can execute in parallel
    - tasks subdivided into multiple processes; perhaps each with multiple threads
- The difficulty is that the developer must decide
    - how to **split up the application into independently executable tasks**
    - **what pieces can/should be executed asynchronously or in parallel**
- It is primarily the compiler and the programming language features that **support the parallel programming** design process
    - the OS can support this design process, at minimum, by **efficiently allocating resources among parallel tasks as defined by the developer**

OS Design Considerations for Multicore architectures

VIRTUAL MACHINE APPROACH

- With the **ever-increasing number of cores on a chip**
  - the attempt to multiprogram individual cores to support multiple applications may be a misplaced use of resources
- If instead, **we allow one or more cores to be dedicated to a particular process** and
  - then **<u>leave the processor alone to devote its efforts to that process</u>**

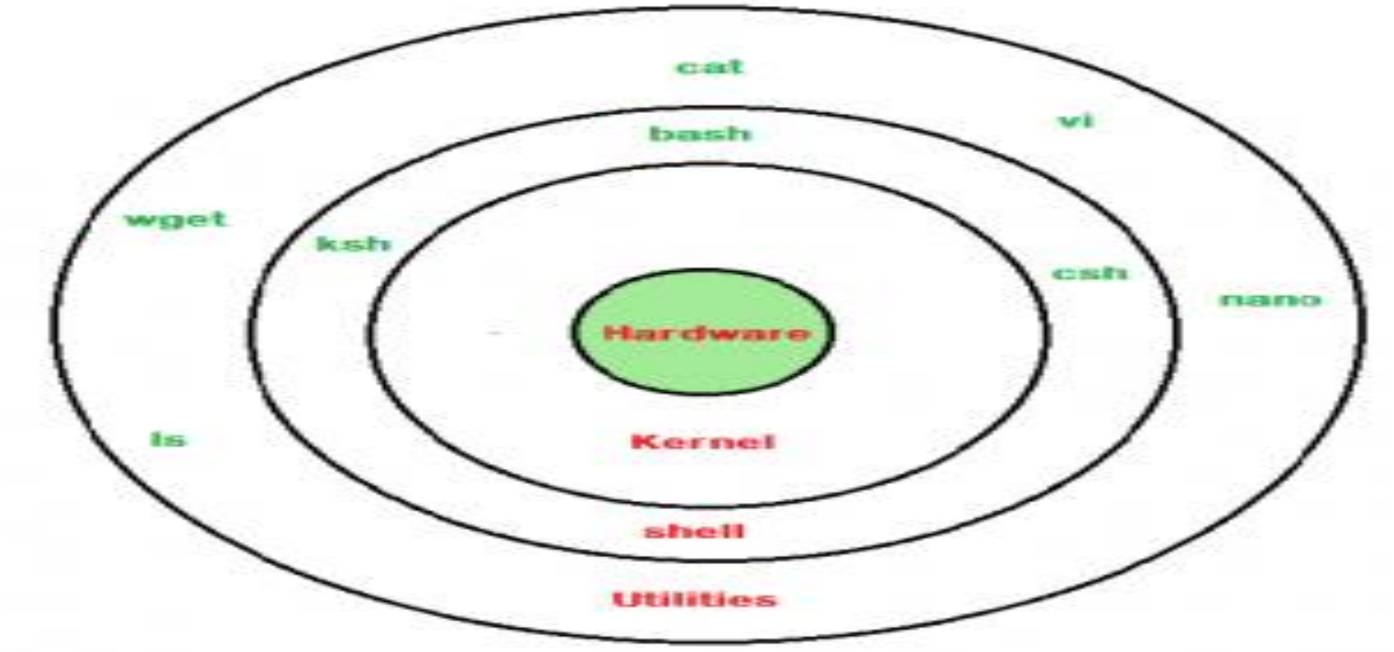
  - we avoid much of the **<u>overhead of task switching and scheduling decisions</u>**

- The **<u>multicore OS could then act as a hypervisor</u>**
  - that makes a high-level decision **to allocate cores to applications**
  - but does little in the way of resource allocation

What is Shell?

Shell is broadly classified into two categories-

- Command Line Shell
- Graphical shell



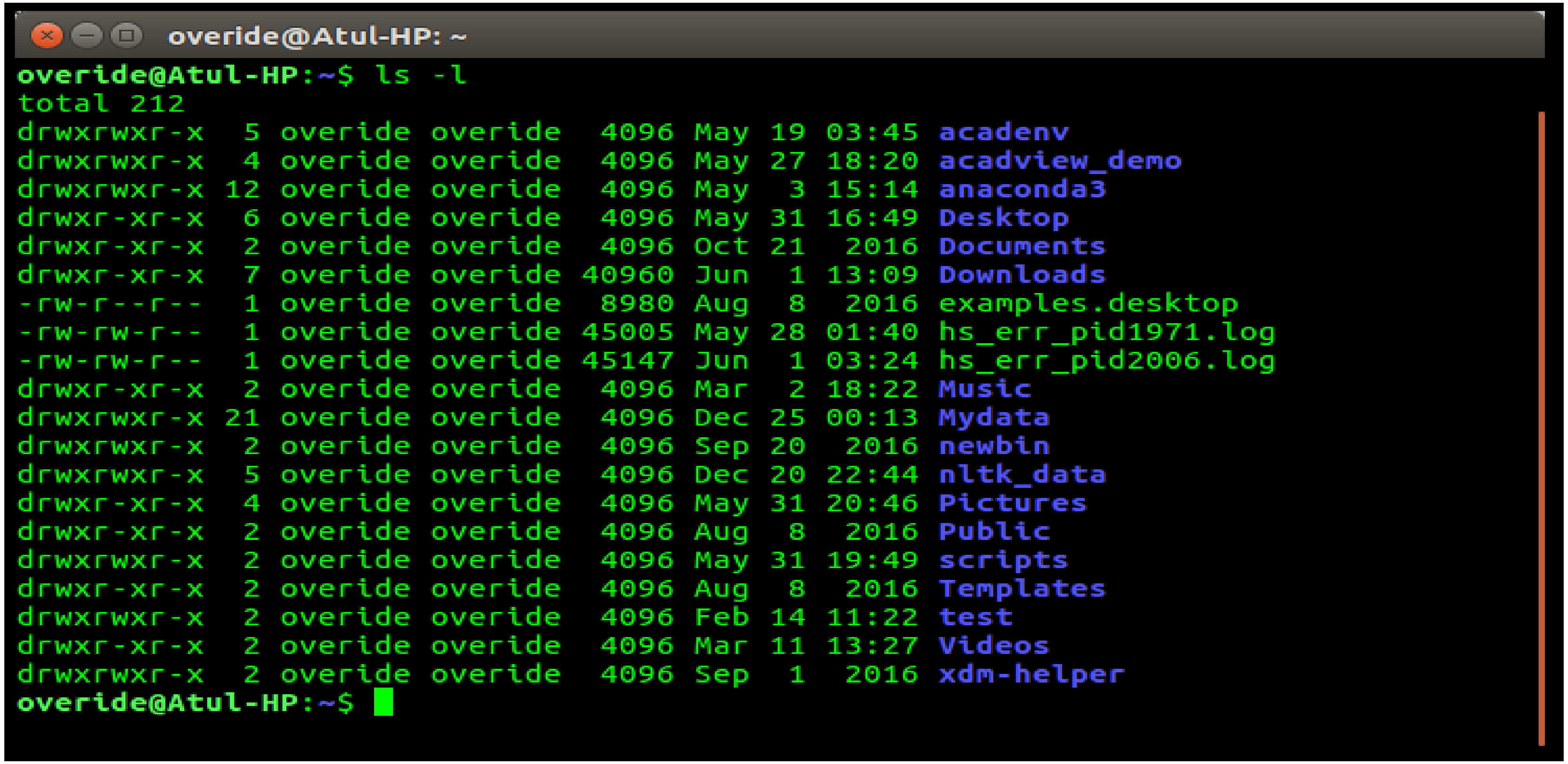
Command Line Shell

- Shell can be accessed by user using a command line interface.

- A special program called
  - **Terminal in linux/macOS or**
  - **Command Prompt in Windows OS**
  - is provided to type in the human readable commands such as “cat”, “ls” etc. for execution. The result is then displayed on the terminal to the user.

Command Line Shell

- A terminal in Ubuntu 16.4 system looks like this –



overide@Atul-HP: ~

```
overide@Atul-HP:~$ ls -l
total 212
drwxrwxr-x  5 overide overide  4096 May 19 03:45 acadenv
drwxrwxr-x  4 overide overide  4096 May 27 18:20 acadview_demo
drwxrwxr-x 12 overide overide  4096 May  3 15:14 anaconda3
drwxr-xr-x  6 overide overide  4096 May 31 16:49 Desktop
drwxr-xr-x  2 overide overide  4096 Oct 21  2016 Documents
drwxr-xr-x  7 overide overide 40960 Jun  1 13:09 Downloads
-rw-r--r--  1 overide overide  8980 Aug  8  2016 examples.desktop
-rw-rw-r--  1 overide overide 45005 May 28 01:40 hs_err_pid1971.log
-rw-rw-r--  1 overide overide 45147 Jun  1 03:24 hs_err_pid2006.log
drwxr-xr-x  2 overide overide  4096 Mar  2 18:22 Music
drwxrwxr-x 21 overide overide  4096 Dec 25 00:13 Mydata
drwxrwxr-x  2 overide overide  4096 Sep 20  2016 newbin
drwxrwxr-x  5 overide overide  4096 Dec 20 22:44 nltk_data
drwxr-xr-x  4 overide overide  4096 May 31 20:46 Pictures
drwxr-xr-x  2 overide overide  4096 Aug  8  2016 Public
drwxrwxr-x  2 overide overide  4096 May 31 19:49 scripts
drwxr-xr-x  2 overide overide  4096 Aug  8  2016 Templates
drwxrwxr-x  2 overide overide  4096 Feb 14 11:22 test
drwxr-xr-x  2 overide overide  4096 Mar 11 13:27 Videos
drwxrwxr-x  2 overide overide  4096 Sep  1  2016 xdm-helper
overide@Atul-HP:~$
```

Command Line Shell

- Working with command line shell is bit difficult for the beginners because it’s hard to memorize so many commands.

- **It is very powerful, it allows user to store commands in a file and execute them together. This way any repetitive task can be easily automated.**

- **These files are usually called**
  - **Batch files in Windows and**
  - **Shell Scripts in Linux/macOS systems.**

Graphical Shells

- Provide means for manipulating programs based on graphical user interface (GUI),
  - by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows.

- **Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program.**
- User do not need to type in command for every actions.

Graphical Shells

- Command-line shells require the user
  - to be familiar with commands and
  - their calling syntax, and
  - to understand concepts about the shell-specific scripting language

- Graphical shells place
  - a low burden on beginning computer users, and
  - are easy to use.
- Since they also come with certain disadvantages, most GUI-enabled operating systems also provide CLI shells.

Shell Prompt

- The prompt, **$**, which is called the **command prompt**, is issued by the shell.
- While the prompt is displayed, you can type a command.
- Shell reads your input after you press **Enter**.
- It determines the command you want to be executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

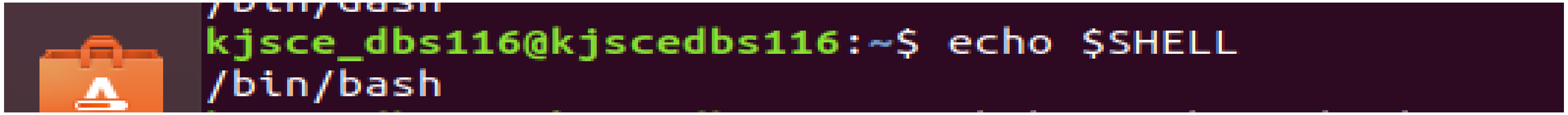
FINDING OUT YOUR SHELL

- When you are provided with a UNIX account, the system administrator chooses a shell for you.
- To find out which shell was chosen for you, look at your prompt.

1) If you have a **$** prompt, you're probably in a **Bash, Bourne** or a **Korn** shell.
2) If you have a **%** prompt, you're probably in a **C shell**.

FINDING OUT YOUR SHELL

- Using the Echo command



```
kjsce_dbs116@kjscedbs116:~$ echo $SHELL
/bin/bash
```

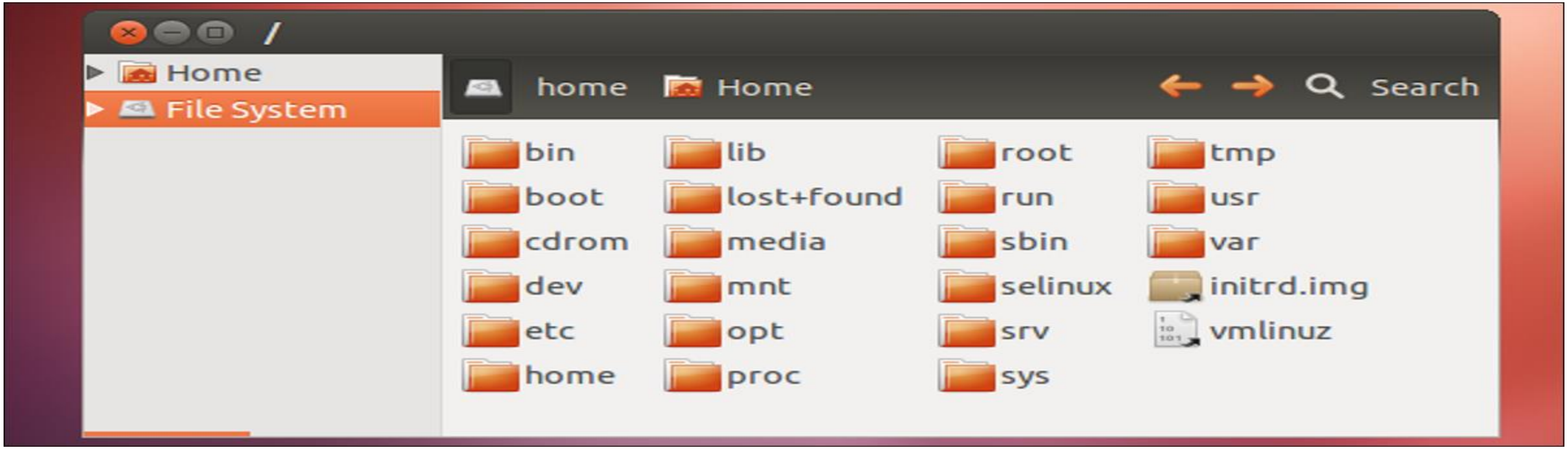
List all the shells on your PC?

- Using the Cat command

List all the shells on your PC?

- The list of all the shells which are currently installed in our Linux system is stored in the **<u>‘shells’ file which is present in /etc folder of the system.</u>**

- It has read-only access by default and is modified automatically whenever we install a new shell in our system.



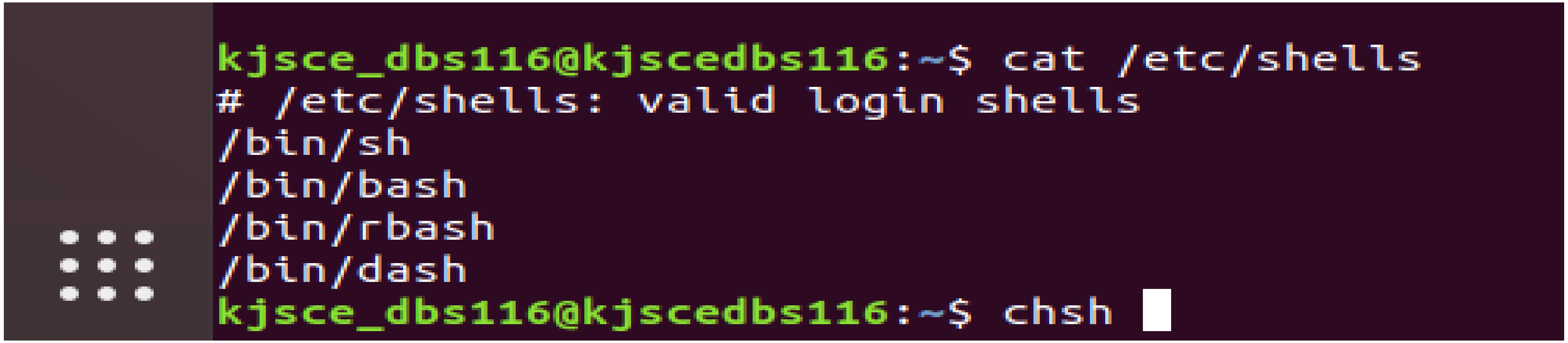
/etc — Configuration Files

- The /etc directory contains configuration files, which can generally be edited by hand in a text editor.

- Note that the /etc/ directory contains system-wide configuration files
- User-specific configuration files are located in each user’s home directory.

List all the shells on your PC?

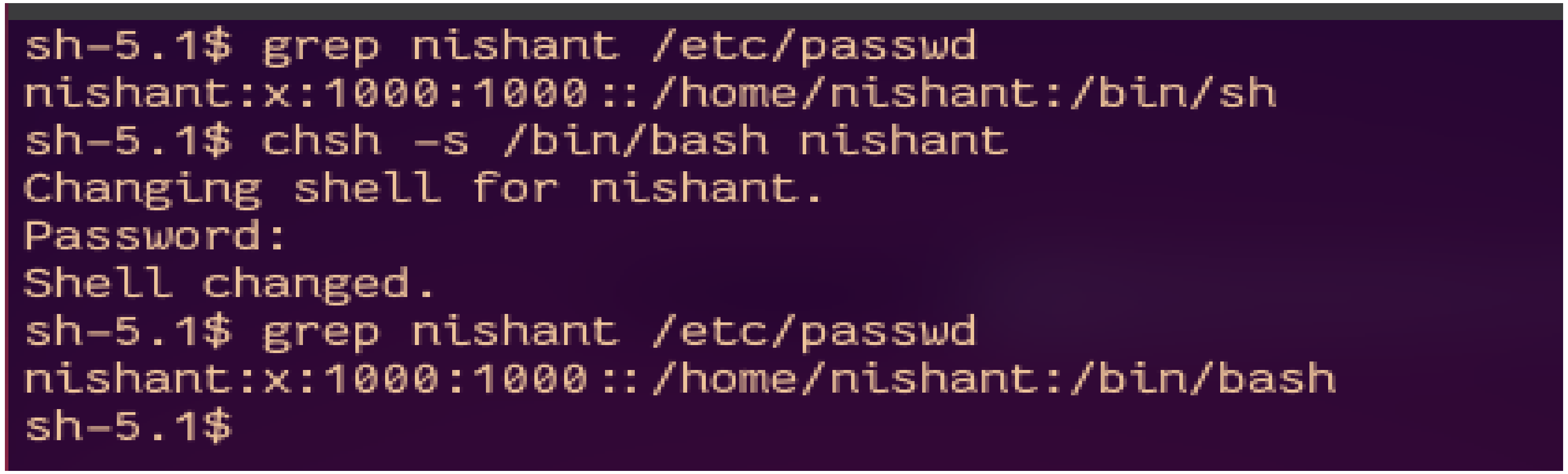
- The cat command displays the various installed shells along with their installation paths.



```
kjsce_dbs116@kjscedbs116:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
kjsce_dbs116@kjscedbs116:~$ chsh
```

Change the shell

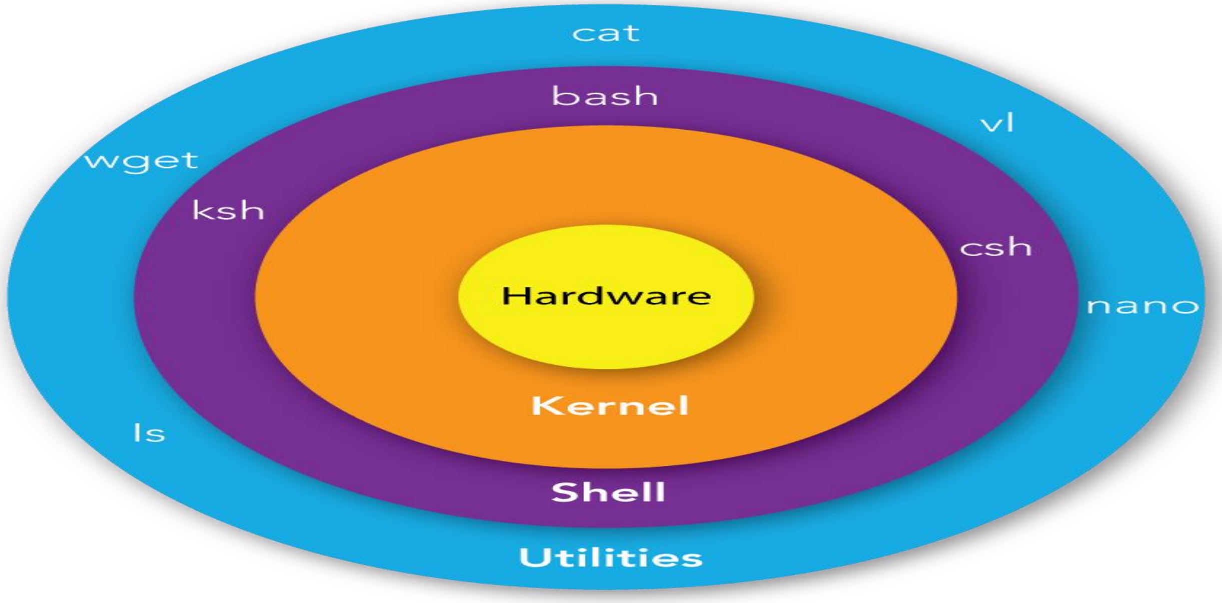
- **Using chsh Utility**
- chsh is the utility to change a user’s login shell. chsh provides the -s option to change the user’s shell.



```
sh-5.1$ grep nishant /etc/passwd
nishant:x:1000:1000::/home/nishant:/bin/sh
sh-5.1$ chsh -s /bin/bash nishant
Changing shell for nishant.
Password:
Shell changed.
sh-5.1$ grep nishant /etc/passwd
nishant:x:1000:1000::/home/nishant:/bin/bash
sh-5.1$
```

Shells

- **There are several shells** are available for Linux systems.
- Each shell does the same job but **understand different commands and provide different built in functions.**



Shell Types

- In UNIX there are two major types of shells:
  - The Bourne shell
    - **If you are using a Bourne-type shell, the default prompt is the $ character**
  - The C shell
    - **If you are using a C-type shell, the default prompt is the % character**

https://www.tutorialspoint.com/unix/unix-what-is-shell.htm

The Bourne Shell

- The original UNIX shell written in the **mid-1970s by Stephen R. Bourne at AT&T Bell Labs in New Jersey**

- **Original UNIX shell-The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell"**

- Denoted as **sh**

- It is faster and more preferred.

https://www.tutorialspoint.com/unix/unix-what-is-shell.htm

The Bourne Shell

- **It lacks features for interactive use like the ability to recall previous commands**
- **It also lacks built-in arithmetic and logical expression handling**
- It is default shell for Solaris OS

CSH (C SHell)

- The **C** Shell; Denoted as **csh**
  - created by **Bill Joy** at the University of California at Berkeley

- **It incorporated features such as aliases and command history**
- **includes helpful programming features like built-in arithmetic**

- **C-like expression syntax; shell syntax and usage are very similar to the C programming language**

Shell Types

- There are various subcategories for Bourne Shell–
  - **Bourne shell (sh)**
  - **Korn shell (ksh)**
  - **Bourne Again shell (bash)**
  - **POSIX shell**

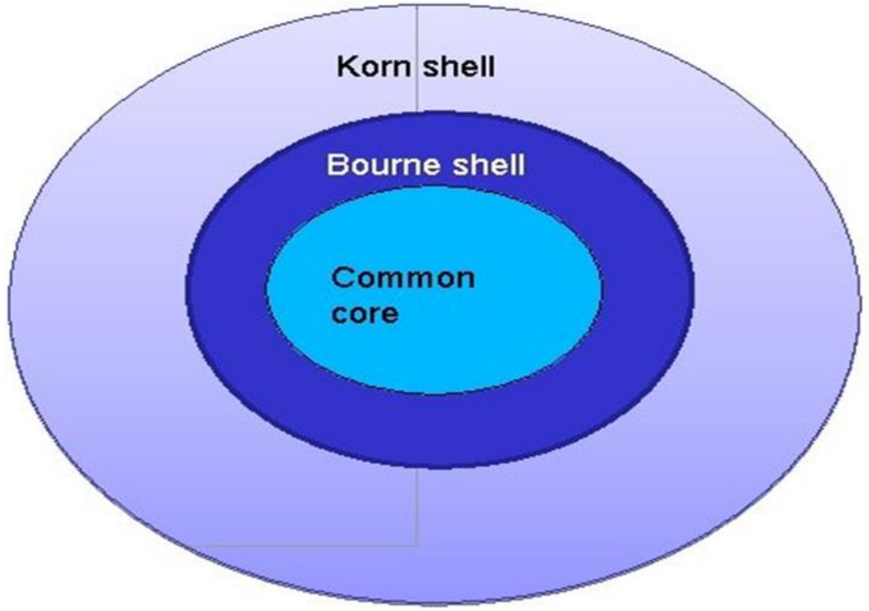
- The different C-type shells –
  - **C shell (csh)**
  - **TENEX/TOPS C shell (tcsh)**

https://www.tutorialspoint.com/unix/unix-what-is-shell.htm

The Korn Shell

- It is denoted as **ksh**
  - written by **David Korn** at AT&T Bell Labs
  - It is a superset of the Bourne shell; so it supports everything in the Bourne shell.

- Has interactive features
  - includes features like built-in arithmetic
  - and C-like arrays, functions, and string-manipulation facilities



- Faster than C shell
- compatible with script written for C shell

- The Korn Shell also was the base for the POSIX Shell standard specifications.

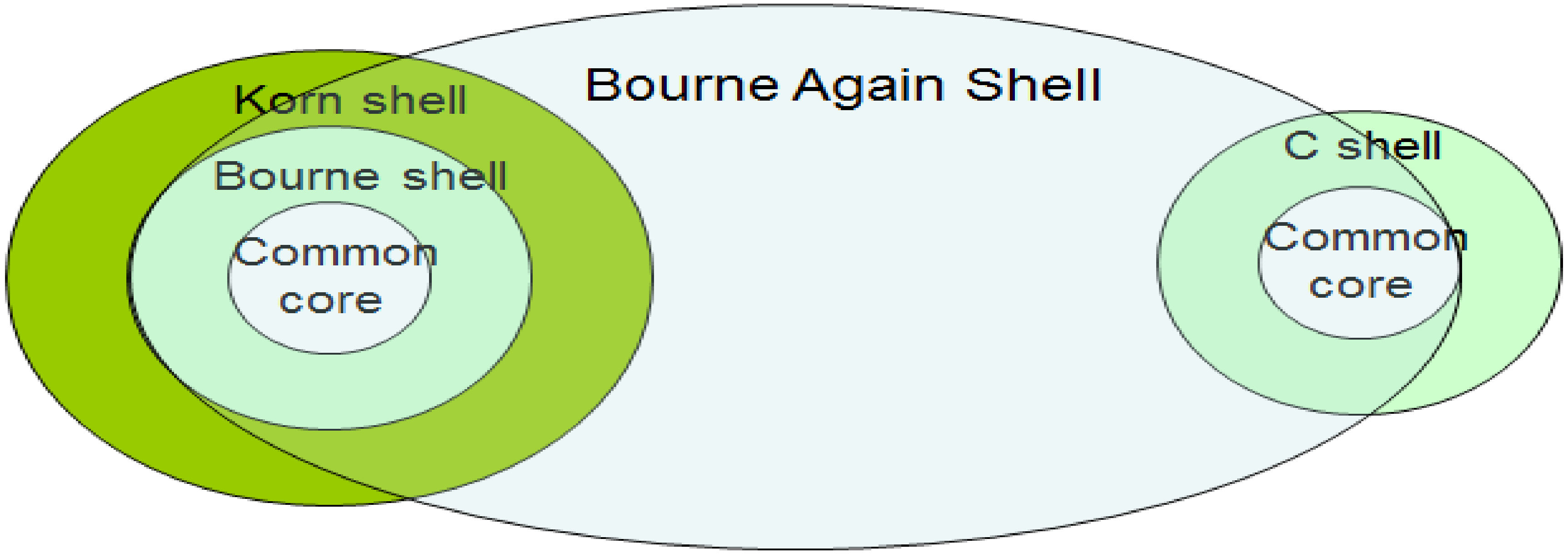
BASH

- BASH= Bourne Again Shell
- The shell's name is an acronym for Bourne Again Shell
- **A pun on the name of the Bourne shell that it replaces and**
- **The notion of being "born again".**

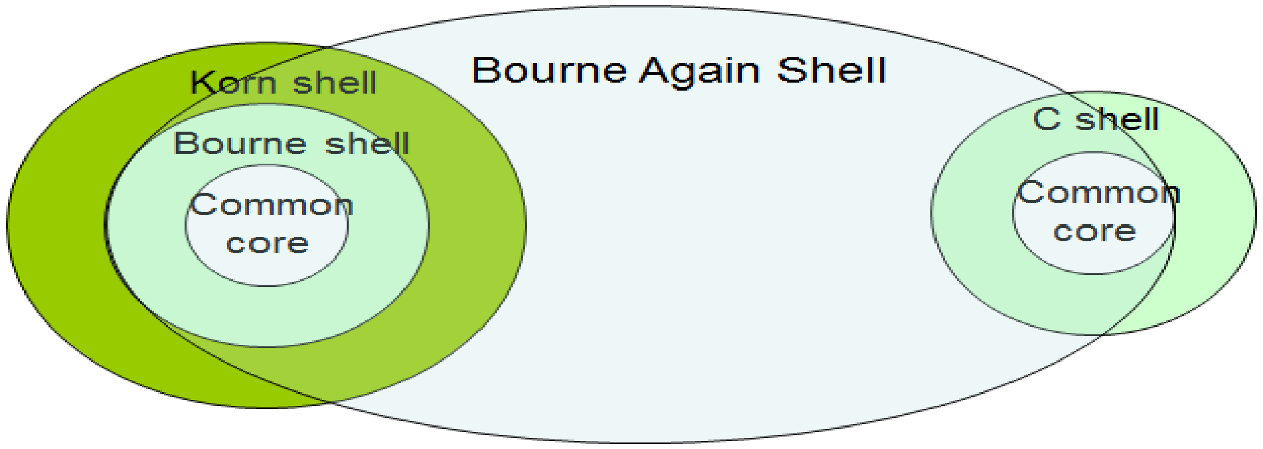
https://en.wikipedia.org/wiki/Bash_(Unix_shell)#:~:text=Bash%20is%20a%20Unix%20shell,ported%20to%20Linux%2C%20alongside%20GCC.

GNU Bourne-Again Shell

- **Bash command syntax is a superset of the Bourne shell command syntax**
- includes ideas drawn from the KornShell (ksh) and the C shell (csh) such as -
- **command line editing, command history (history command),**
- **the directory stack, and POSIX command substitution syntax**



GNU Bourne-Again Shell



- Denoted as **bash**
- It is compatible to the Bourne shell
- includes features from Korn and Bourne shell
- It is most widely used shell in Linux systems
- It is used as **default login shell in Linux systems and in macOS**
  - can also be installed on Windows OS.

GNU ?

- **GNU is an operating system** and also an extensive collection of utility programmes wholly free software and also the project within which the free software concept originated

- **GNU is a recursive acronym for "GNU's Not Unix!“**

- **Chosen because GNU's design is Unix-like, but differs from Unix by being free software and containing no Unix code.**

Shell Scripting

- Usually shells are interactive;
  - they accept command as input from users and execute them.
  - to execute a bunch of commands routinely,
  - **we have to type in all commands each time in terminal.**

Shell Scripting

- As shell can also take commands
  - **as input from file we can write these commands in a file**
  - can execute them in shell to avoid this repetitive work.

- These files are called Shell Scripts or Shell Programs.
  - Shell scripts are **similar to the batch file in MS-DOS.**
  - Each shell script is saved with **.sh file extension** eg. myscript.sh

Shell Scripting

- A shell script have syntax just like any other programming language.

- A shell script comprises following elements –

  - **Shell Keywords – if, else, break etc.**
  - **Shell commands – cd, ls, echo, pwd, touch etc.**
  - **Functions**
  - **Control flow – if..then..else, case and shell loops etc.**

The Shell Definition Line

- The shell definition line tells the system
  - **what shell should be used** to interpret the script's commands, and
  - where the program (shell) is located.

- The shell definition line tells the system to use the **Korn Shell when executing this script.**

  - (#!/bin/ksh)

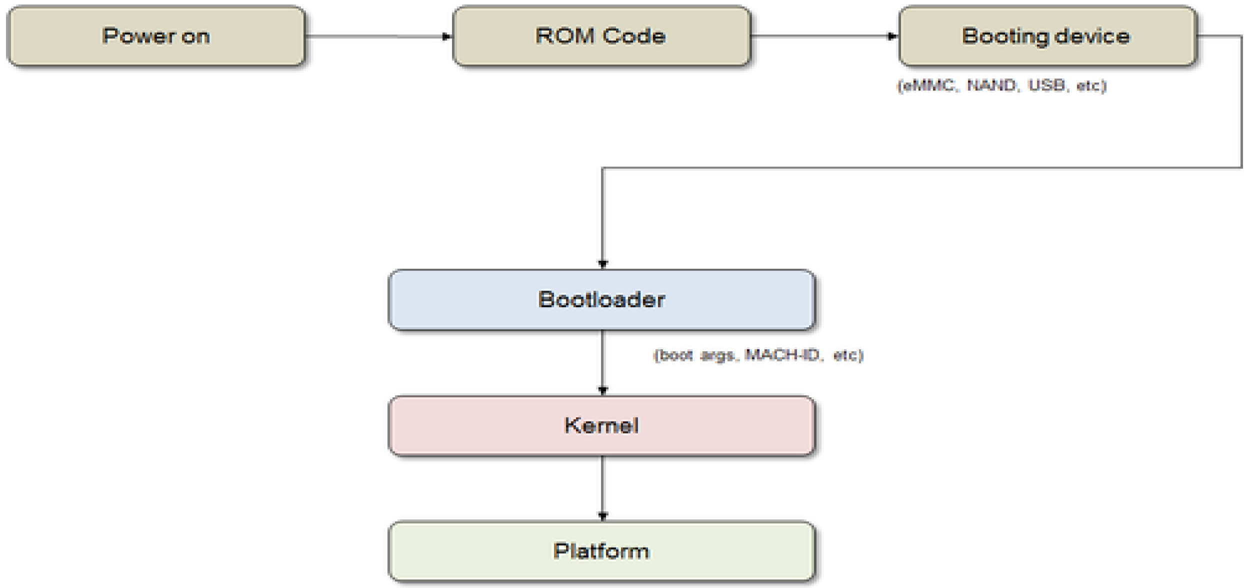
System Boot

- The boot process is something that
  - happens every time we turn our computer **ON**
  - we don’t really see it, because it happens so fast;
    - You press the power button, come back a few minutes later and
    - Windows XP, or Windows 10, or whatever Operating System you use is all loaded

- When power is initialized on system,
  - execution starts at a fixed memory location,
  - **Firmware ROM used to hold initial boot code**
- Operating system must be
  - made available to hardware , so hardware can start it

System Boot

- Sometimes two-step process where
  - **Boot block** at fixed location loaded by ROM code,
  - which **loads bootstrap program from disk**
  - Then **loader** continues with its job



System Boot

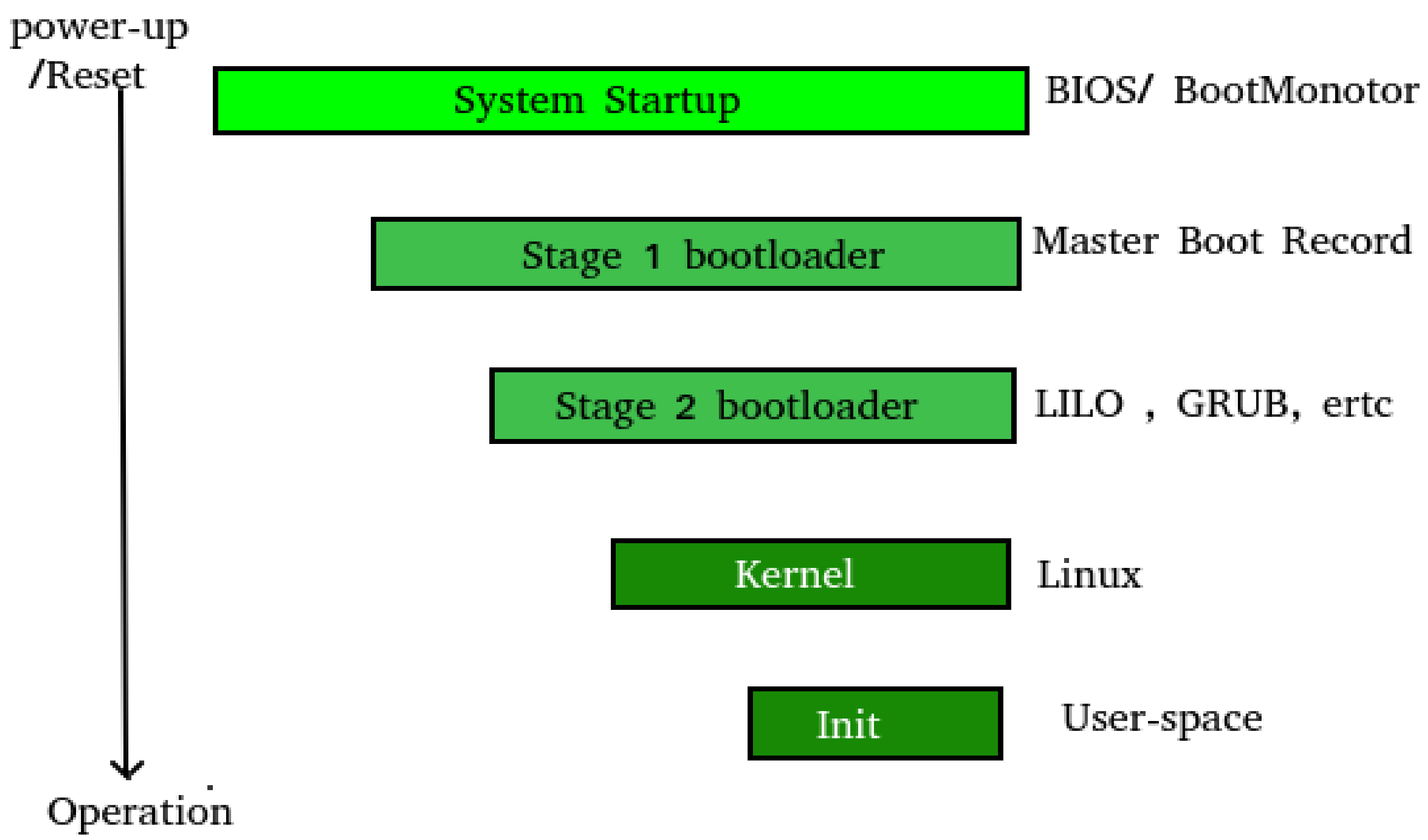
- **Bootstrap loader**
  - Small piece of code,
  - stored in **ROM** or **EEPROM** (electrically erasable programmable read-only memory)
  - locates the kernel,
  - loads it into memory, and
  - starts it

System Boot

- Common bootstrap loader,
  - **GRUB**, (the **GRand Unified Bootloader)**
  - allows selection of kernel from
    - multiple disks,
    - versions,
    - kernel options

- Kernel loads and
  - system is then **running**

System Boot


power-up /Reset
System Startup
BIOS/ BootMonotor
Stage 1 bootloader
Master Boot Record
Stage 2 bootloader
LILO , GRUB, ertc
Kernel
Linux
Init
User-space
Operation

System Boot- The working

1. The **CPU initializes itself** after the power in the computer is first turned on
   - This is done by triggering a series of clock ticks; generated by the system clock

2. **CPU looks for the system's ROM BIOS**
   - to obtain the first instruction in the start-up program.
   - This first instruction is stored in the ROM BIOS -it instructs the system to
     - run POST (Power On Self Test)
     - in a memory address that is predetermined

- Since ROM is read only, it cannot be infected by a computer virus

System Boot- The working

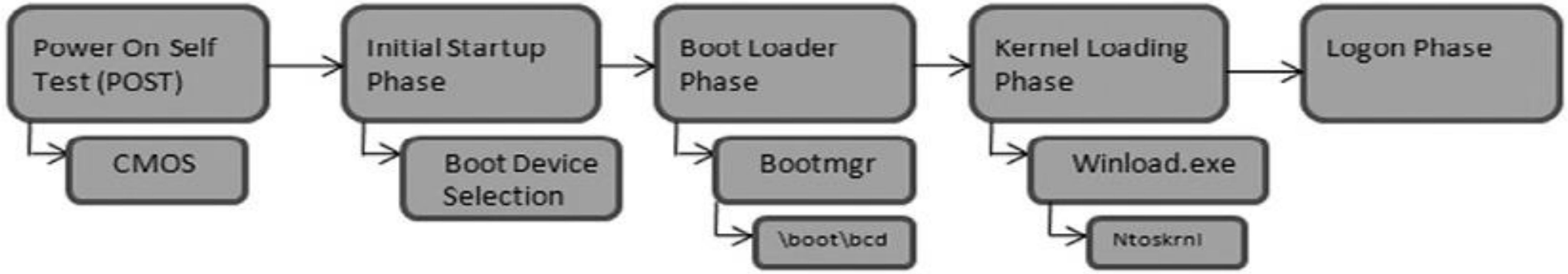
- POST first checks
  - the BIOS chip and
  - then the CMOS RAM
  - If no battery failure is detected by POST,
  - it continues to initialize the CPU

- POST also checks
  - the hardware devices,
  - secondary storage devices such as hard drives, ports etc.
  - And other hardware devices such as the mouse and keyboard
  - This is done to make sure they are working properly.

After POST makes sure that all the components are working properly, then the BIOS starts Boot Strap program.

POST method

- To perform this check, the POST
  - Sends out a standard command that says to all devices, “Check yourselves out!”
  - All the standard devices in the computer then run their own internal diagnostic
  - The POST doesn’t specify what they must check
  - The quality of the diagnostic is up to the people who made that particular device



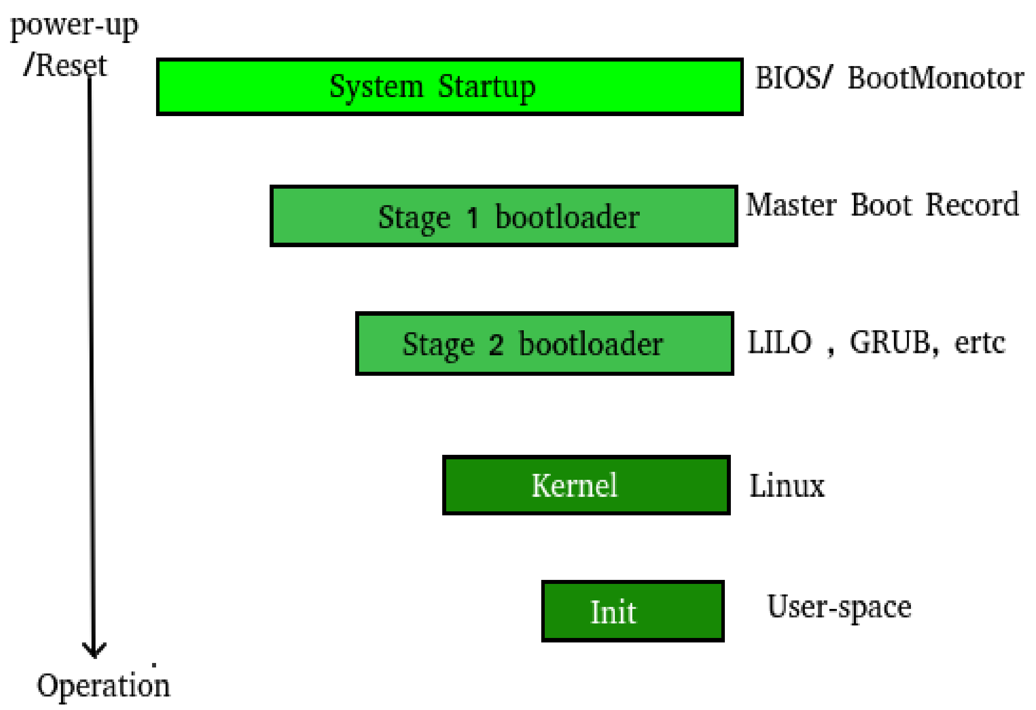
Courtesy : http://www.c-jump.com/CIS24/Slides/Booting/Booting.html

System Boot- The working

- For most computers, **Boot Strap is stored in BIOS ROM**
  - Changing the Bootstrap code requires changing the ROM hardware chips

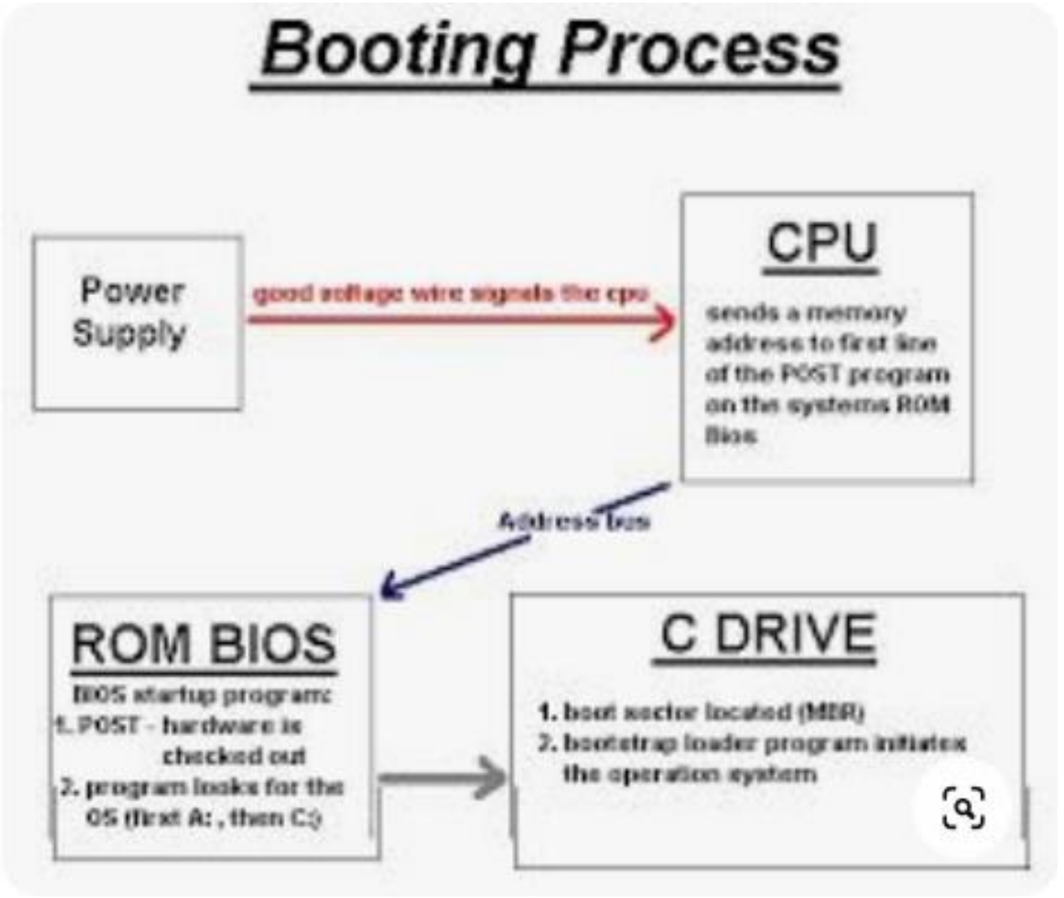
Alternative:

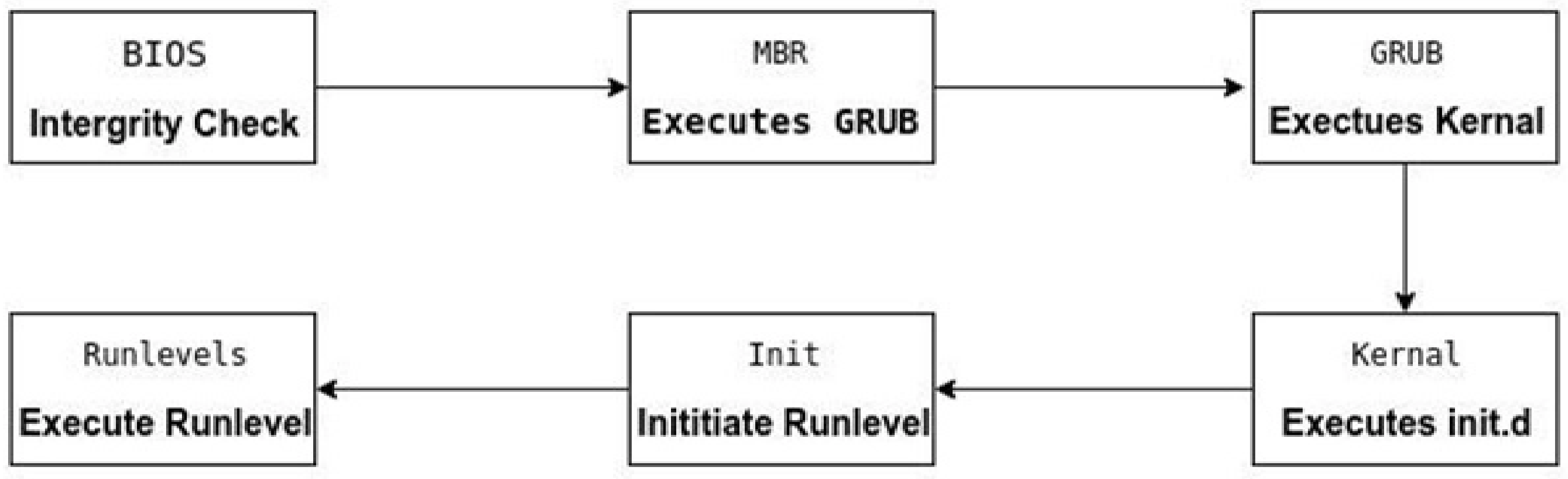
- A **tiny Bootstrap loader** program is stored in the ROM,
  - whose only **job is to bring in a full bootstrap program from the disk**
  - The full bootstrap program can be changed easily



System Boot- The working

- Bootstrap program
  - is **stored in a partition** called the **boot blocks**, at a fixed location on the disk
- A disk that has a boot partition
  - is called a **Boot Disk** or **System Disk**
- The Bootstrap program is loaded into memory and starts its execution
- The bootstrap program
  - **finds the OS Kernel on disk,**
  - **loads the Kernel into Memory and**
  - **jumps to an initial address to begin the OS execution**


Booting Process
Power Supply
good voltage wire signals the cpu
CPU
sends a memory address to first line of the POST program on the systems ROM Bios
Address bus
ROM BIOS
BIOS startup program:
1. POST - hardware is checked out
2. program looks for the OS (first A: , then C:)
C DRIVE
1. boot sector located (MBR)
2. bootstrap loader program initiates the operation system



Master Boot Record (MBR)
It is the information which is in the **first sector of any hard disk**

It holds information about GRUB (or LILO in old systems).

init

- This is the **last step** of the booting process

- ‘init’ is the first process that starts when the system boots
  - Starts system processes; Runs startup scripts
  - Creates processes from a script in the /etc/inittab file
  - Controls independent processes required by the system
  - Establishes and operates the entire user space

- It decides the run level by looking at the /etc/inittab file

- initial state of the operating system is decided by the **run level**

init

Following are the run levels of Operating System:

Level
0 - System Halt
1 - Single user mode
2 - Multiuser, without NFS (Network File System)
3 - Full multiuser mode
4 - Unused
5 - Full multiuser mode with network and X display manager (X11)
6 - Reboot

Default run level to be set, would be either 3 or 5

- The step after init is to:
- Start up various **daemons** that support networking and other services

- Daemons are **processes that run unattended**
  - They run constantly in the background and are available at all times
  - Daemons are usually started when the system starts, and they run until the system stops.

- **X server** daemon manages **display, keyboard, and mouse**
  - You can see a Graphical Interface and
  - a login screen is displayed during X server daemon startup

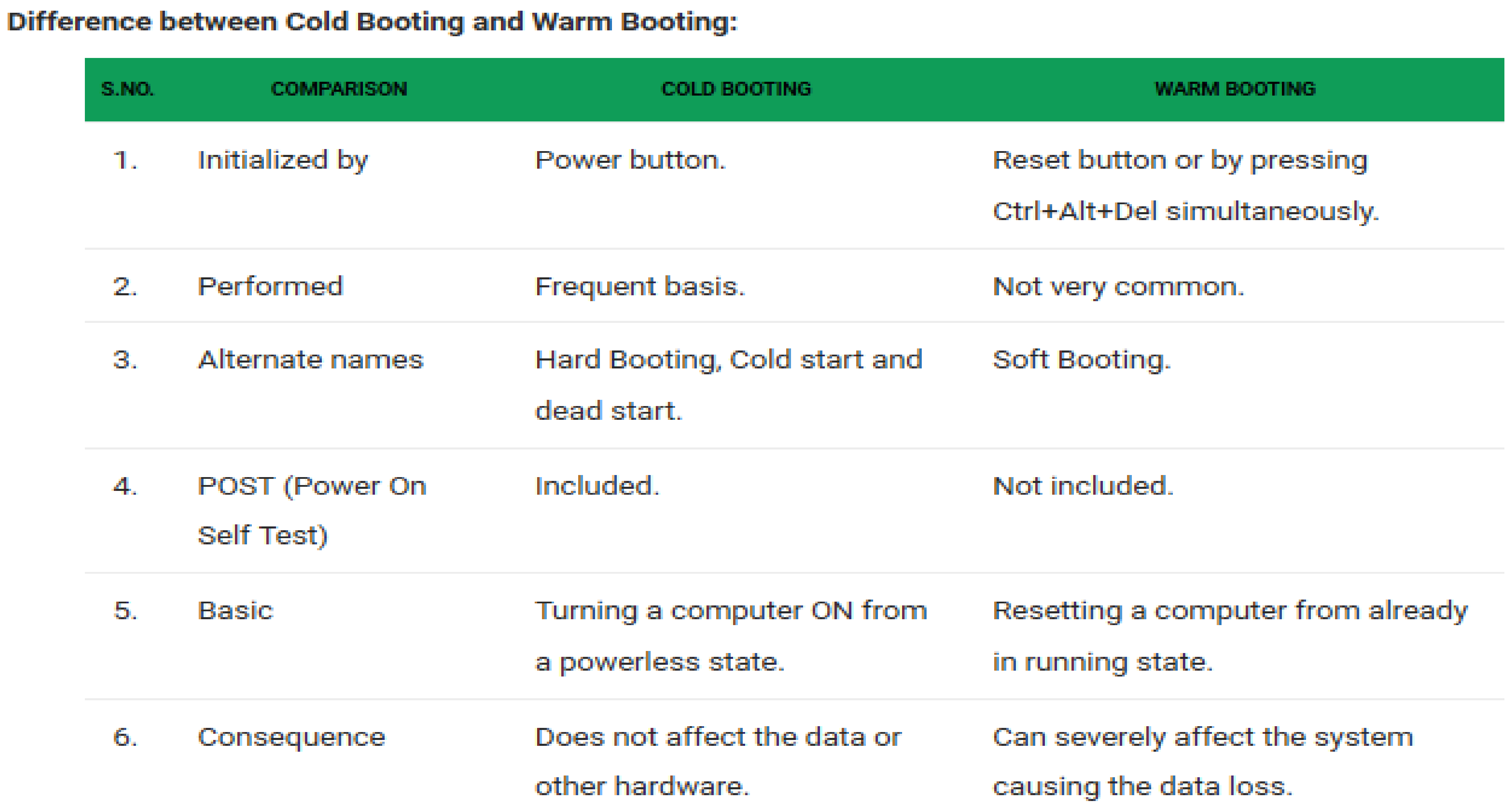
Failure during boot

- If the computer cannot boot, we get a **Boot Failure Error**
  - indicates that the computer is not passing POST or
  - a device in the computer, such as the hard drive or memory, has failed
  - we may also hear a beep code to identify which hardware is failing during the POST
  - An error message or blue screen may show on the screen as operating system files cannot be loaded, due to not being found or being corrupt

Boot

- Cold Boot (also called “hard boot”)
  - To perform a cold boot means to **start up a computer that is turned off**
- Warm Boot-
  - It is often used in contrast to a cold boot, which **refers to restarting a computer once it has been turned ON**
- While a warm boot and cold boot are similar,
  - a cold boot performs a more complete reset of the system than a warm boot

- Warm Boot-
  - If the computer hangs because of some reason while working, and demands to be restarted to make it working
  - The process of reset/restart of the computer system is called as warm booting
  - It is done with the help of reset button or keys (Ctrl+Alt+Del).



Difference between Cold Booting and Warm Booting:

S.NO.	COMPARISON	COLD BOOTING	WARM BOOTING
1.	Initialized by	Power button.	Reset button or by pressing Ctrl+Alt+Del simultaneously.
2.	Performed	Frequent basis.	Not very common.
3.	Alternate names	Hard Booting, Cold start and dead start.	Soft Booting.
4.	POST (Power On Self Test)	Included.	Not included.
5.	Basic	Turning a computer ON from a powerless state.	Resetting a computer from already in running state.
6.	Consequence	Does not affect the data or other hardware.	Can severely affect the system causing the data loss.

What is the name given to the process of initializing a microcomputer with its operating system?

(a) Cold booting

(b) Booting

(c) Warm booting

(d) Boot recording

(e) None of the above

What is the name given to the process of initializing a microcomputer with its operating system?

(a) Cold booting

(b) Booting **(Ans)**

(c) Warm booting

(d) Boot recording

(e) None of the above

Consider the following statements :

UGC-NET | UGC-NET CS 2017 Nov – III | Question 73

(a) UNIX provides three types of permissions
* Read
* Write
* Execute
(b) UNIX provides three sets of permissions
* permission for owner
* permission for group
* permission for others

Which of the above statement/s is/are true ?

(A) only (a)
(B) only (b)
(C) Both (a) and (b)
(D) Neither (a) nor (b)

Consider the following statements :

UGC-NET | UGC-NET CS 2017 Nov – III | Question 73

(a) UNIX provides three types of permissions
* Read
* Write
* Execute
(b) UNIX provides three sets of permissions
* permission for owner
* permission for group
* permission for others
Which of the above statement/s is/are true ?

(A) only (a)
(B) only (b)
(C) Both (a) and (b)
(D) Neither (a) nor (b)

Answer: (C)
Explanation: UNIX provides Read, Write and Execute permisision on files
UNIX provides three sets of permissions

permission for owner

permission for group

permission for others

UGC-NET | UGC NET CS 2015 Dec – III | Question 56

In Unix, the command to enable execution permission for file “mylife” by all is ______________.

(A) Chmod ugo + X myfile

(B) Chmod a + X myfile

(C) Chmod + X myfile

(D) All of the above

UGC-NET | UGC NET CS 2015 Dec – III | Question 56

In Unix, the command to enable execution permission for file “mylife” by all is ____________.

(A) Chmod ugo + X myfile
(B) Chmod a + X myfile
(C) Chmod + X myfile
(D) All of the above

Answer: (D)

Explanation: In Unix, the command to enable execution permission for file “mylife” by all are:
Chmod + X myfile
Chmod a + X myfile
Chmod ugo + X myfile
So, option (D) is correct.

System Calls

System Calls

Provide the Interface between a user process and the OS

WHEN?

- System calls are usually made when a process in user mode requires access to a resource
- Then it requests the kernel to provide the resource via a system call.

System Calls

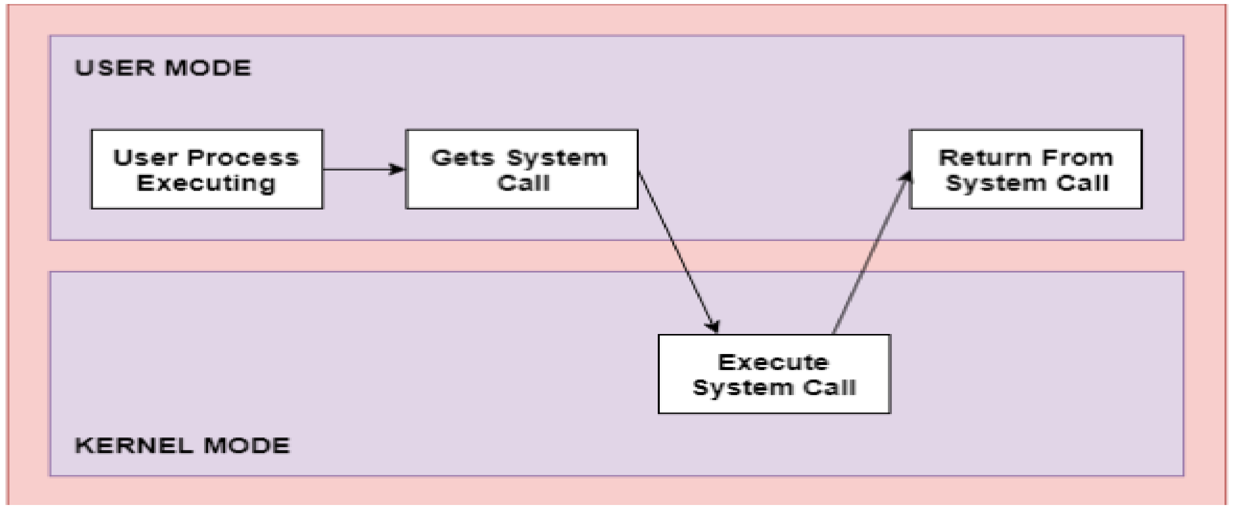
WHEN?

System calls are required in the following situations −

1) **Creation or deletion of files** by file system
2) **Reading and writing from files**
3) **Creation and management of new processes**
4) **Access to hardware devices** such as a printer, scanner etc. requires a system call
5) **Network connections** also require system calls; this includes **sending and receiving packets**

System Calls

1) The processes execute normally in the user mode until a system call interrupts
2) The system call is executed on a **priority basis in the kernel mode**
3) After the execution of the system call, the control returns to the user mode
4) Execution of user processes can be resumed



System Calls

- Generally available as **Assembly language instructions**
- **Certain systems allow system calls to be made directly from a higher level language program,**
  - In this case, they resemble predefined functions or subroutine calls
- Languages like **C,C++, Perl have been defined to replace Assembly language** for system programming
  - They **allow system calls to be made directly from programs.**
- **Unix system call may be invoked directly from C or C++ program**

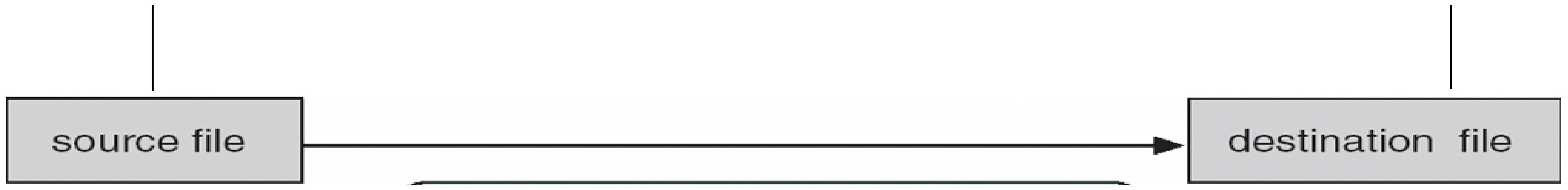
System Calls

- **Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use**

- 3 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)

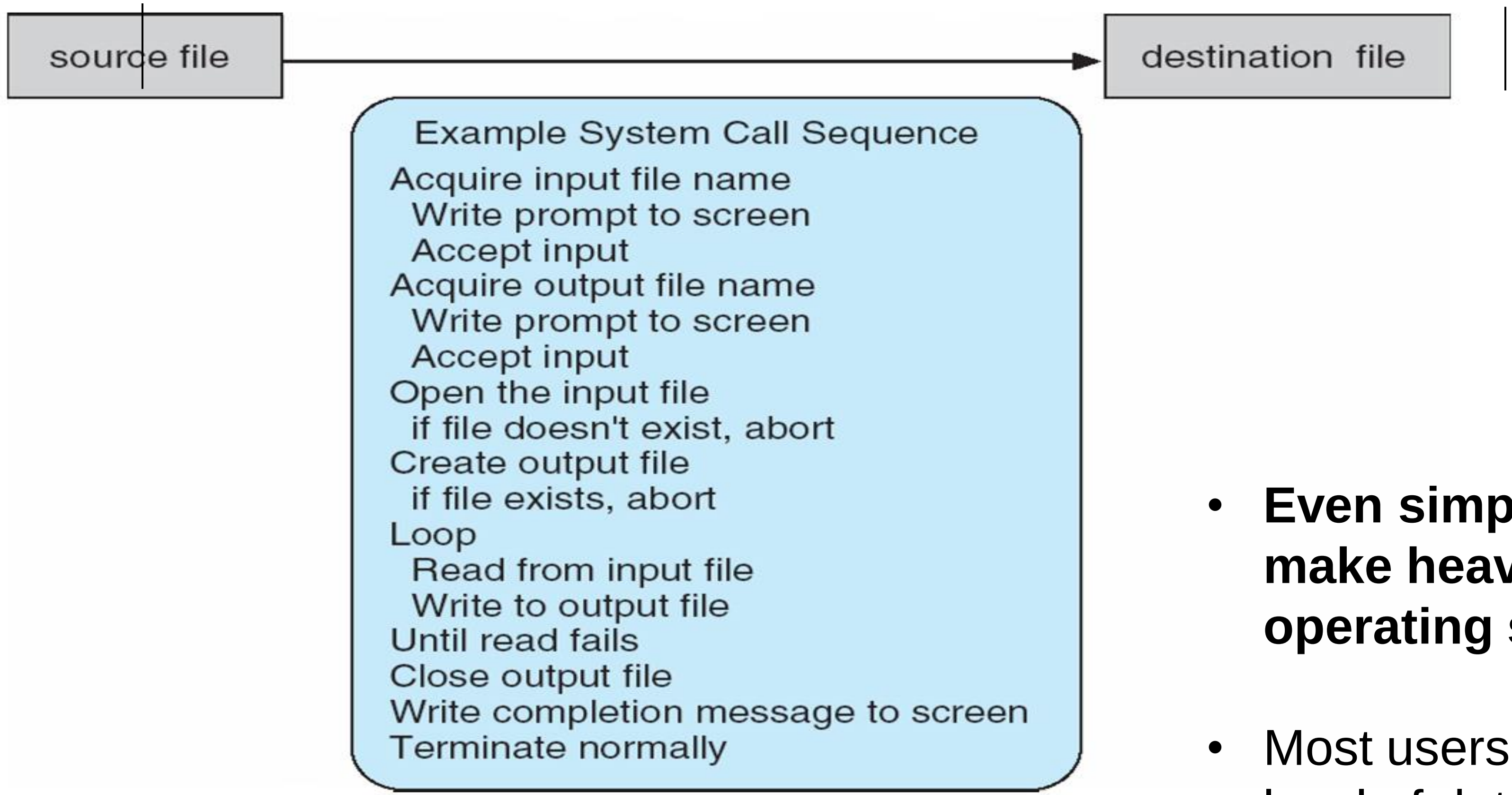
Example of System Calls

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



Example of System Calls

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



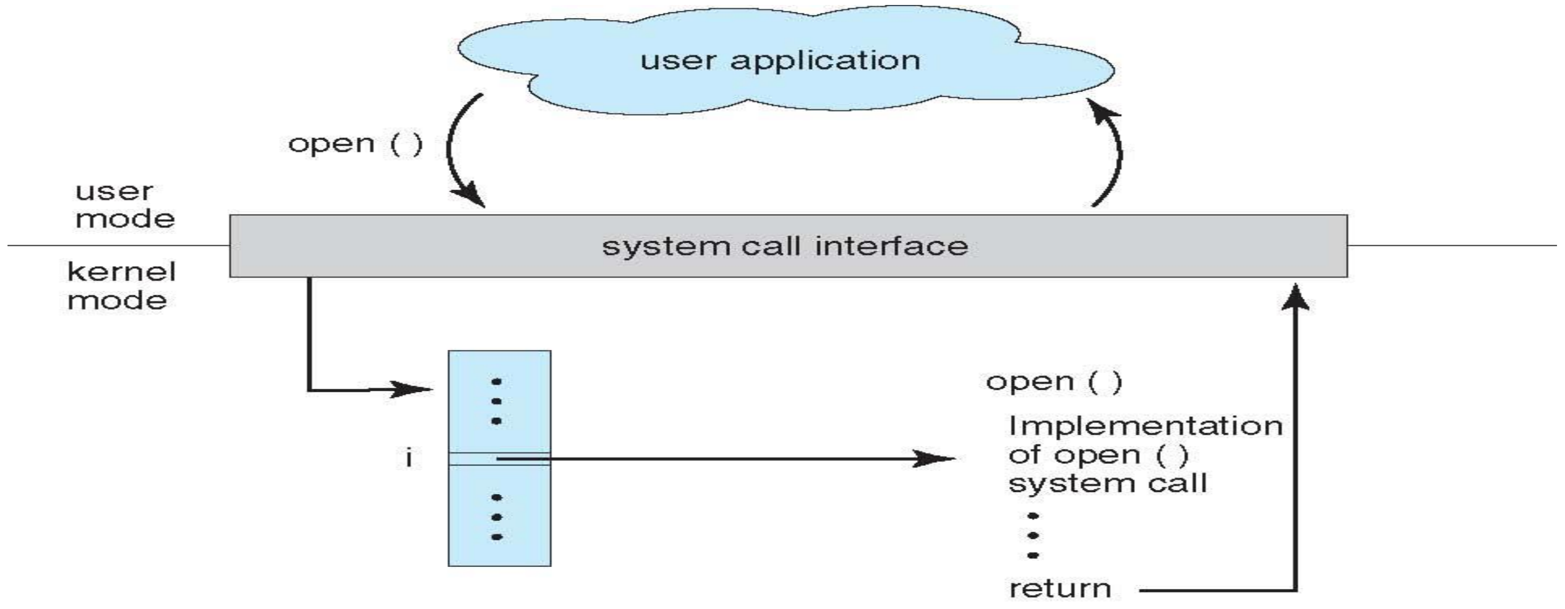
Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

- **Even simple programs make heavy use of the operating system**
- Most users never see this level of details

System Call Implementation

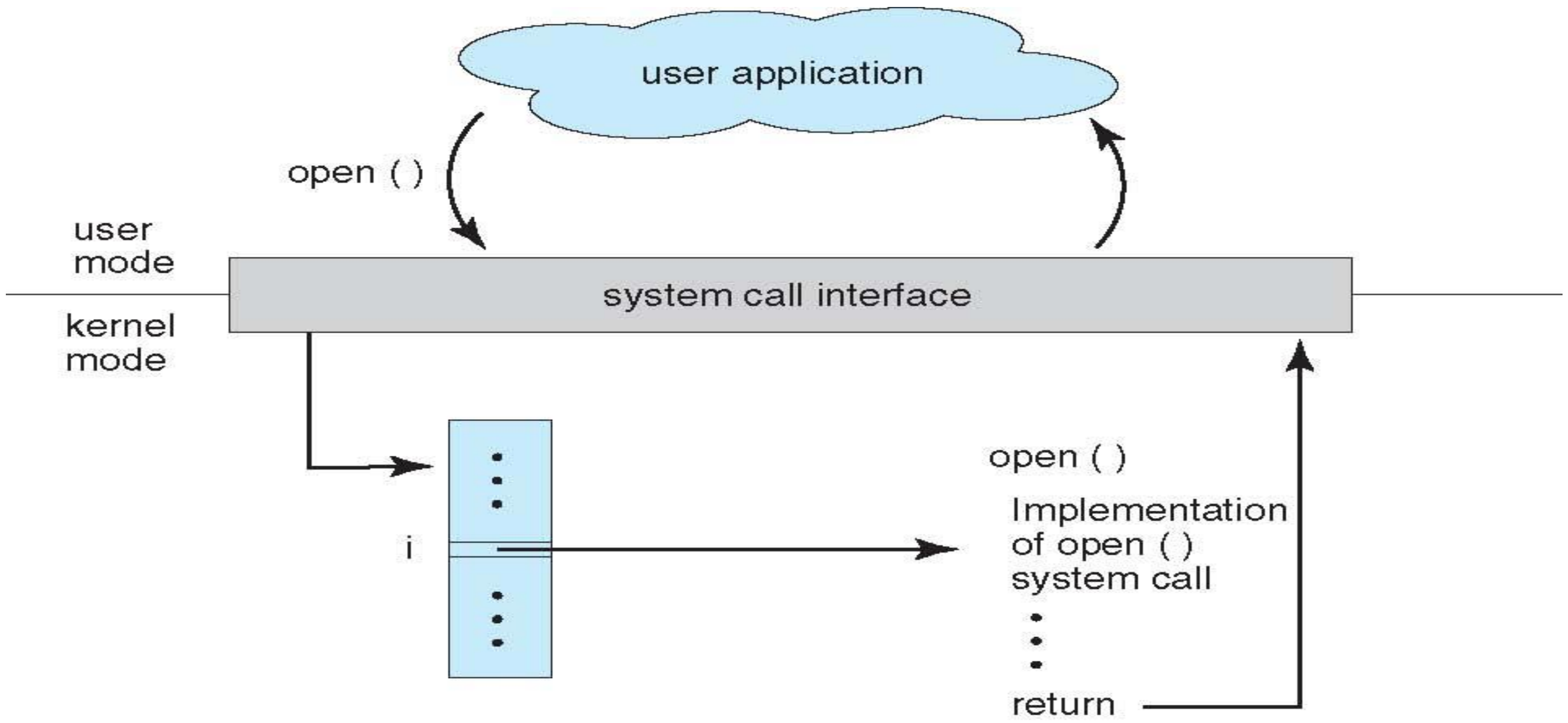
- **A <u>number</u> is associated with Each system call**
- **System-call interface maintains a Table indexed according to these numbers**
- The system call interface
  - **invokes the intended system call in OS kernel and**
  - **returns status of the system call and any return values**



System Call Implementation

- **The caller need not know anything about how the system call is implemented**
  - Just needs to obey API and **understand what OS will do as a result of 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

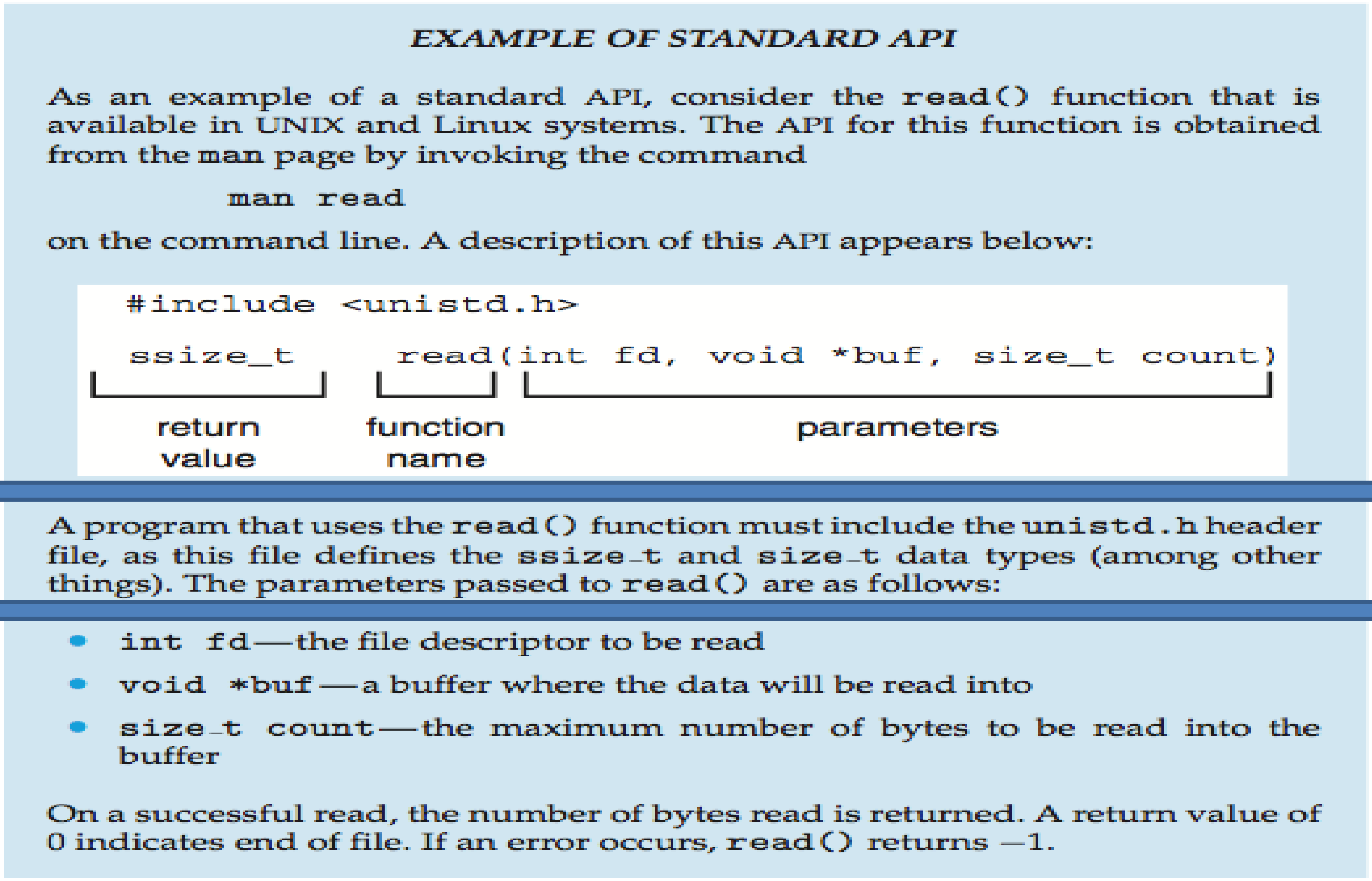


System Call Parameter Passing

- System Calls occur in different ways
  - Often, more information is required than simply identity of desired system call
  - **Exact type and amount of information vary according to OS and call**

  - Eg- To get input,
    - **we need to specify file or device to use as the source** and
    - the address and **length of the memory buffer** into which the input should be read

Example of Standard API



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>
```

$$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\quad \underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$$

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

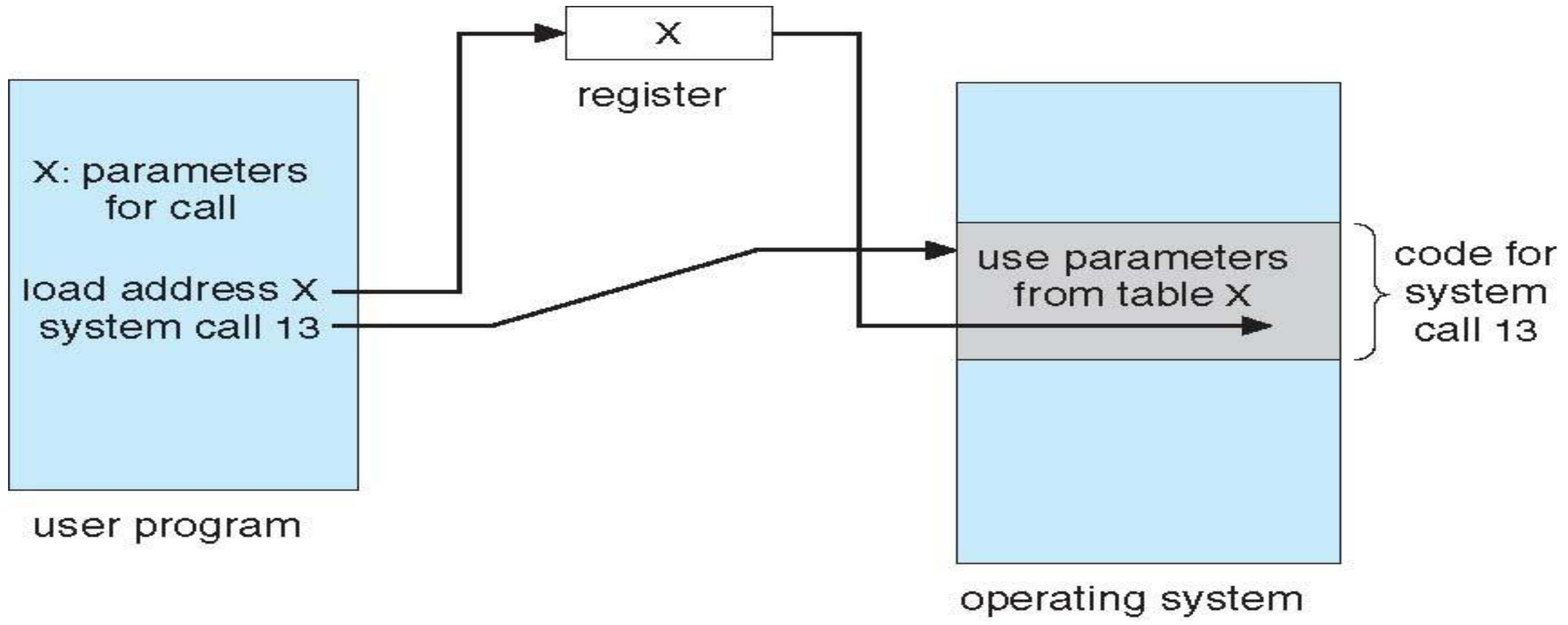
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

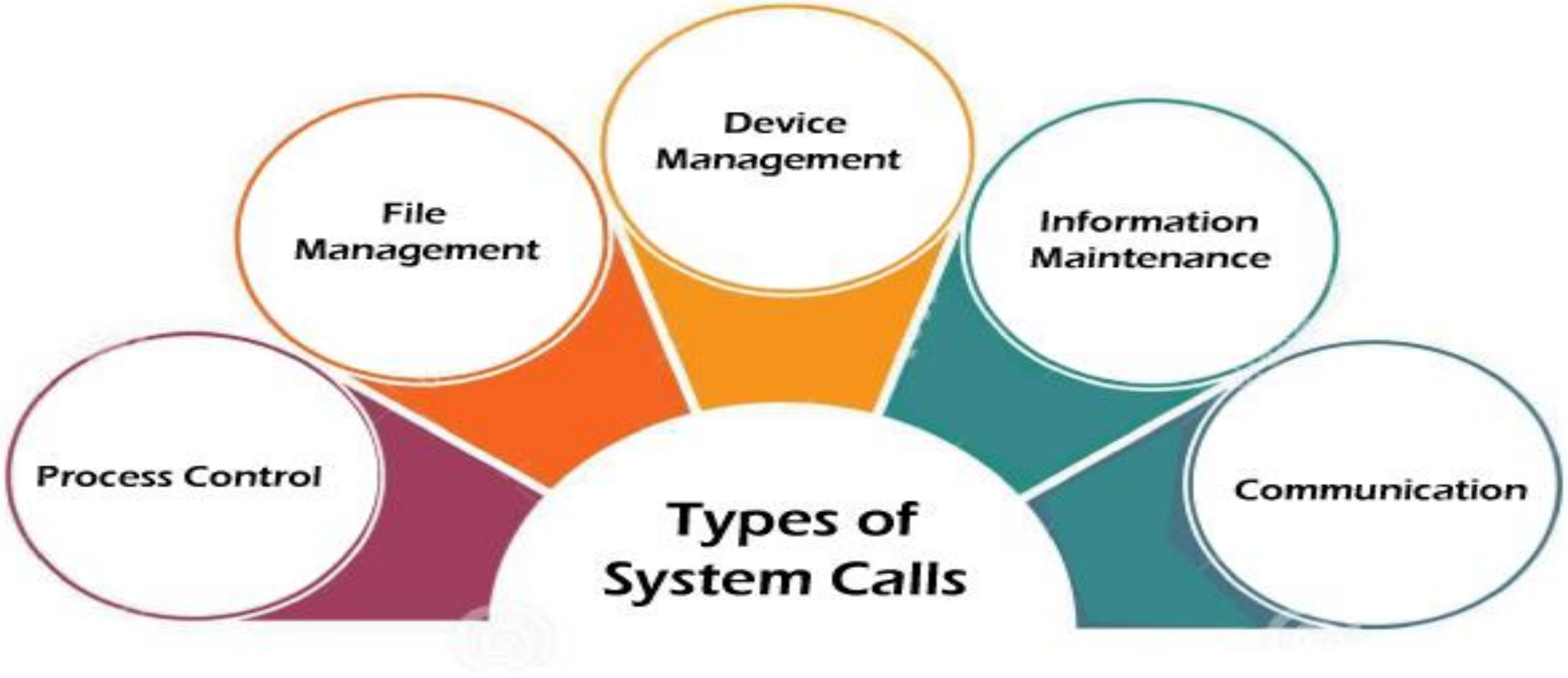
System Call Parameter Passing

- **3 general methods** used to pass parameters to the OS
  - **Simplest**: **pass the parameters in registers**
    - In some cases, there 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 are placed, or pushed, onto the stack by the program and popped off the stack, by the OS**
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls



Types of System Calls

- **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

Types of System Calls

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

Types of System Calls

- **Device management**
  - **request device, release device**
  - **read, write**, reposition
  - **get device attributes, set device** attributes
  - **logically attach or detach devices**

Types of System Calls (Cont.)

- **Information maintenance**
  - **get time or date, set time or date**
  - get system data, set system data
  - **get and set process (process id), file, or device** attributes

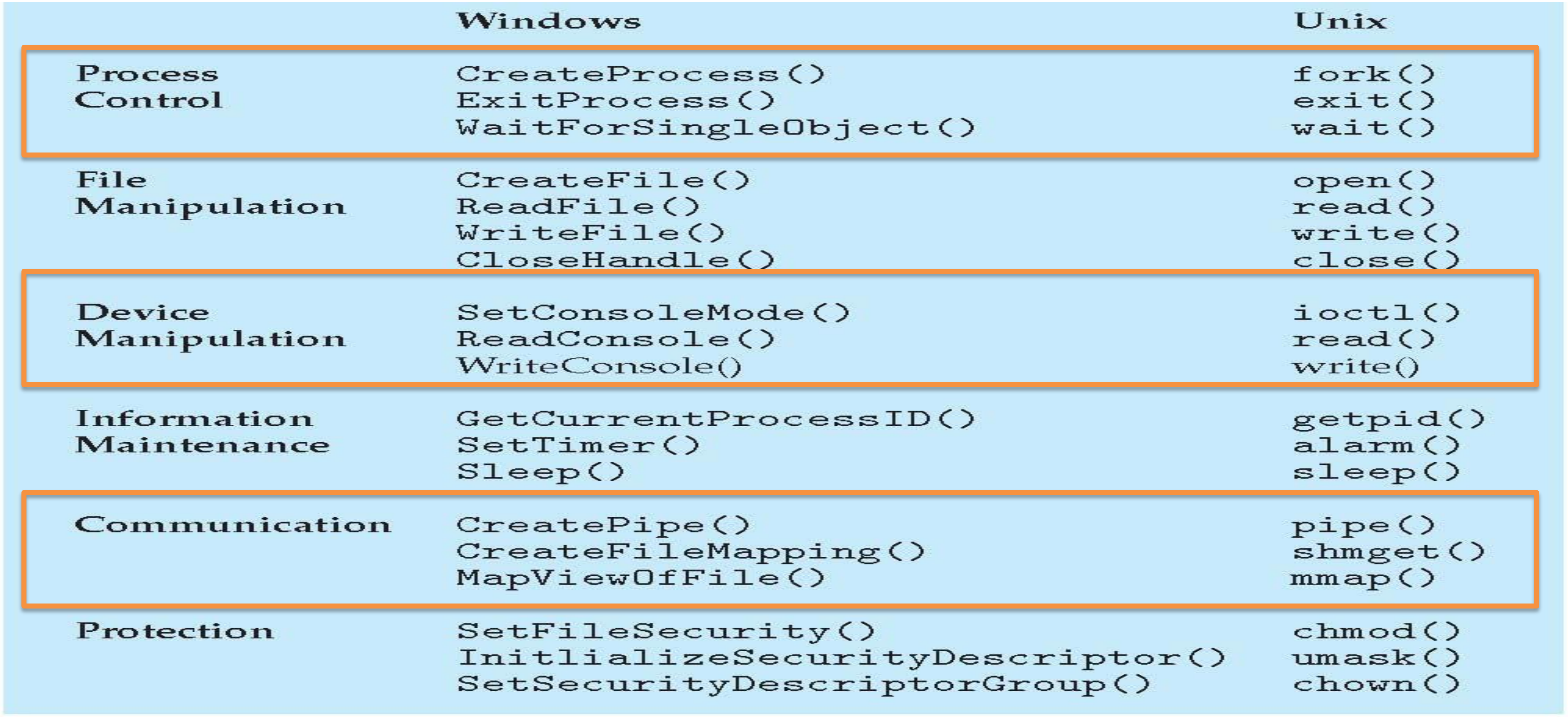
Types of System Calls (Cont.)

- **Communication**
  - **create, delete communication connection**
  - **send, receive messages**
  - **If message passing model to host name or process name**
    - **From client to server**
  - **Shared-memory model create and gain access to memory regions**
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

- **Protection**
  - **Controlled access to resources**
  - **Get and set permissions**
  - Allow and deny user access

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()

Contact info.

- Email:
  prasannashete@somaiya.edu

- Mobile/WhatsApp : 9960452937