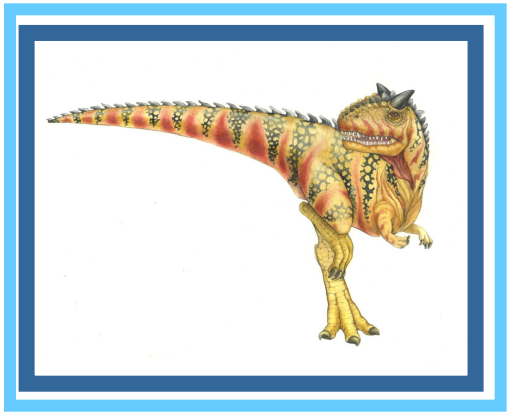
Chapter 1: Introduction



(By:Ashish Singh Rathore)

Operating System?

Introduction to Operating System : Operating System Meaning, Supervisor & User Mode, review of computer organization, introduction to popular operating systems like UNIX, Windows, etc., OS structure, system calls, functions of OS, evolution of OSs

Process Management : PCB, Operations on Processes, Co-operating and Independent Processes,Inter-Process Communication, Process states, Operations on processes, Process management in UNIX, Process concept, Life cycle, Process and threads

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**

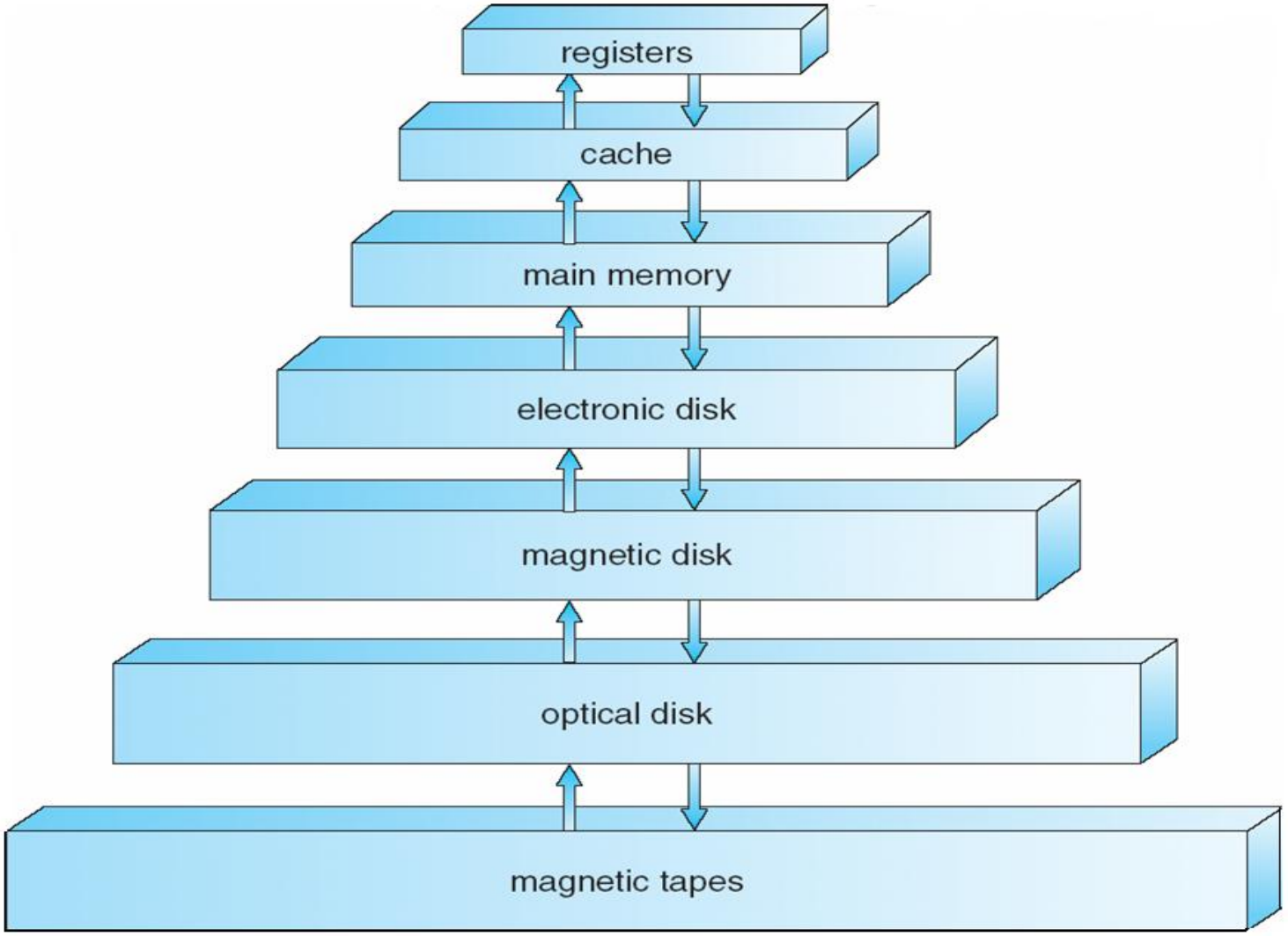
- **Operating system goals:**
  - **Execute user programs and make problem solving easier.**
  - **Make the computer system convenient to use**
  - **Efficiently use available resources**

- 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.

History of Operating system

Generation	Year	Electronic device used	Types of OS Devices
First	1945-55	Vacuum Tubes	Plug Boards
Second	1955-65	Transistors	Batch Systems
Third	1965-80	Integrated Circuits(IC)	Multiprogramming
Fourth	Since 1980	Large Scale Integration	PC

Storage Structure and Hierarchy

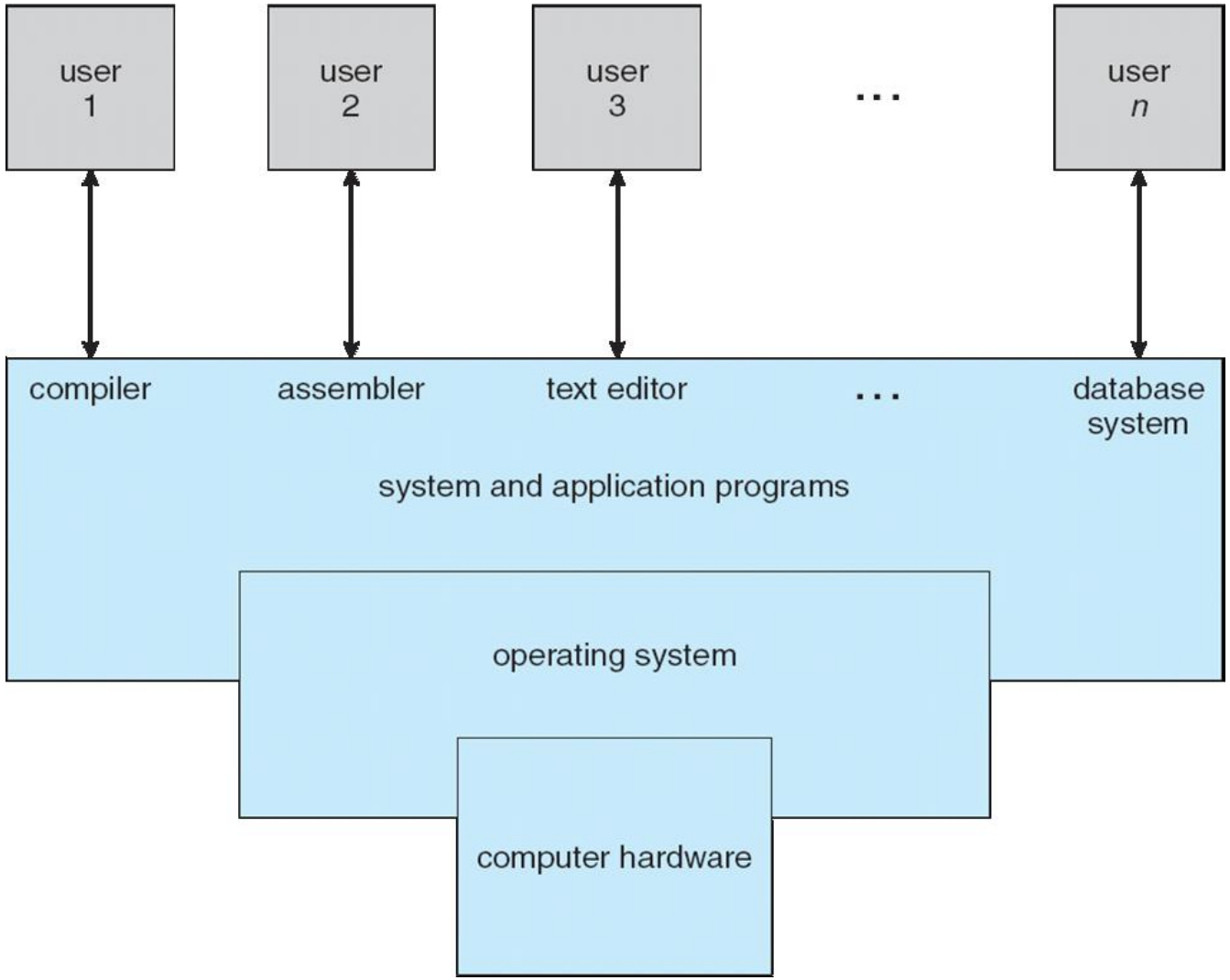


Computer System Structure

- **Computer system can be divided into four components:**

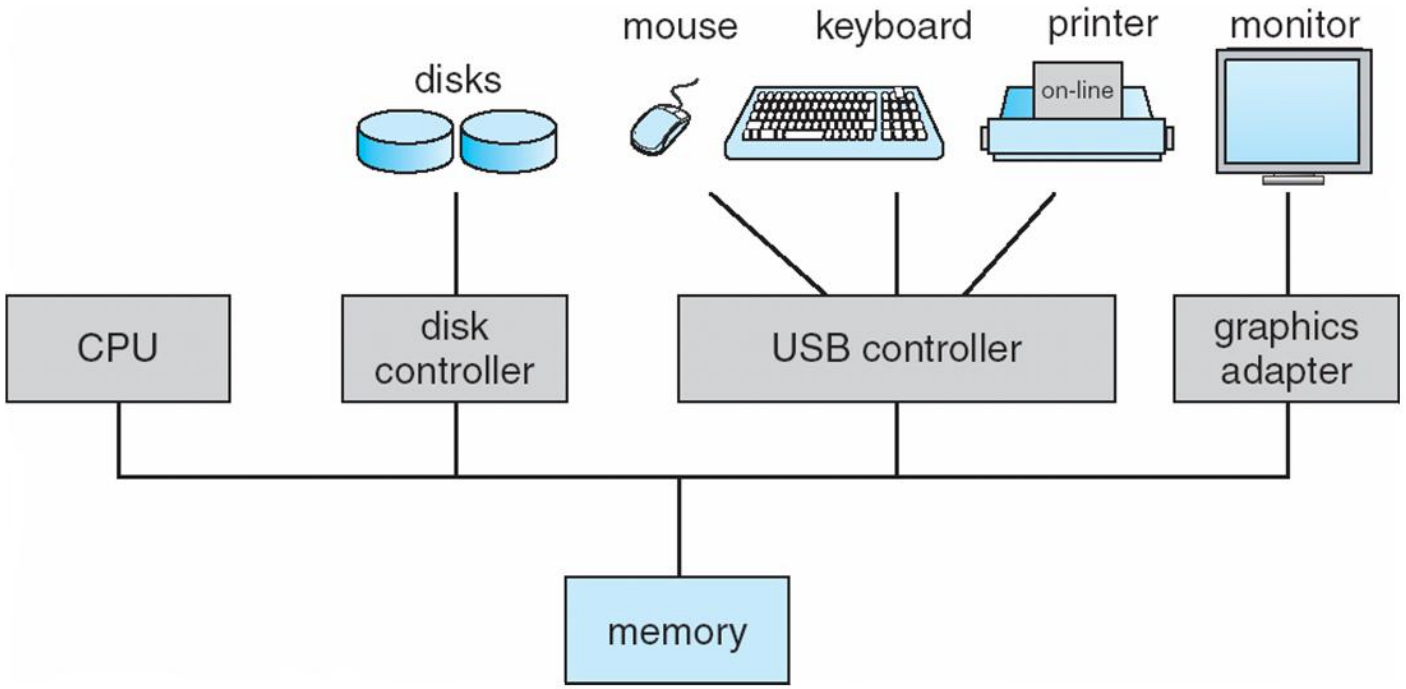
- **Hardware** – provides **basic computing resources**
  - CPU, memory, I/O devices
- **Operating system**
  - **Controls and coordinates use of resources** among various applications and users
- **System/Application programs** – define the ways in which the system resources are used to solving user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

Four Components of a Computer System

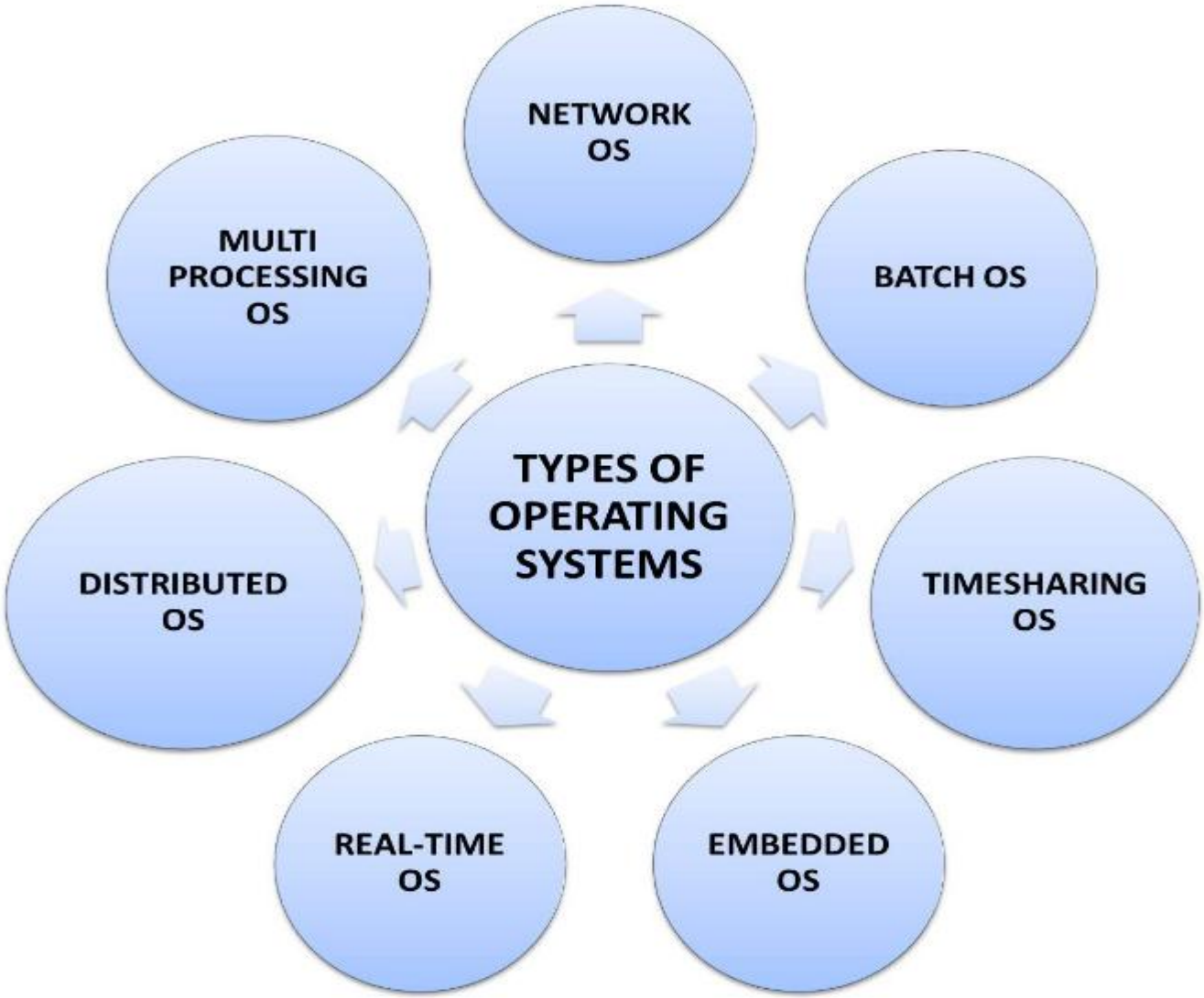


Computer System Organization

- 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


TYPES OF OPERATING SYSTEMS
NETWORK OS
BATCH OS
TIMESHARING OS
EMBEDDED OS
REAL-TIME OS
DISTRIBUTED OS
MULTI PROCESSING OS

Serial Processing

The Serial Processing Operating Systems are those which Performs all the instructions into a **Sequence Manner** or the Instructions those are given by the user will be **executed by using the FIFO Manner** means First in First Out. All the Instructions those are Entered First in the System will be Executed First and the Instructions those are Entered Later Will be Executed Later. **For Running the Instructions the Program Counter is used which is used for Executing all the Instructions**. In this the Program Counter will determines which instruction is going to Execute and the which instruction will be Execute after this. Mainly the **Punch Cards** are used for this. In this all the Jobs **are firstly Prepared and Stored on the Card** and after that card will be entered in the System and after that all the Instructions will be executed one by One. But the **Main Problem is that a user doesn't interact with the System** while he is working on the System, means the user can't be able to enter the data for Execution.


From John Ousterhout slides

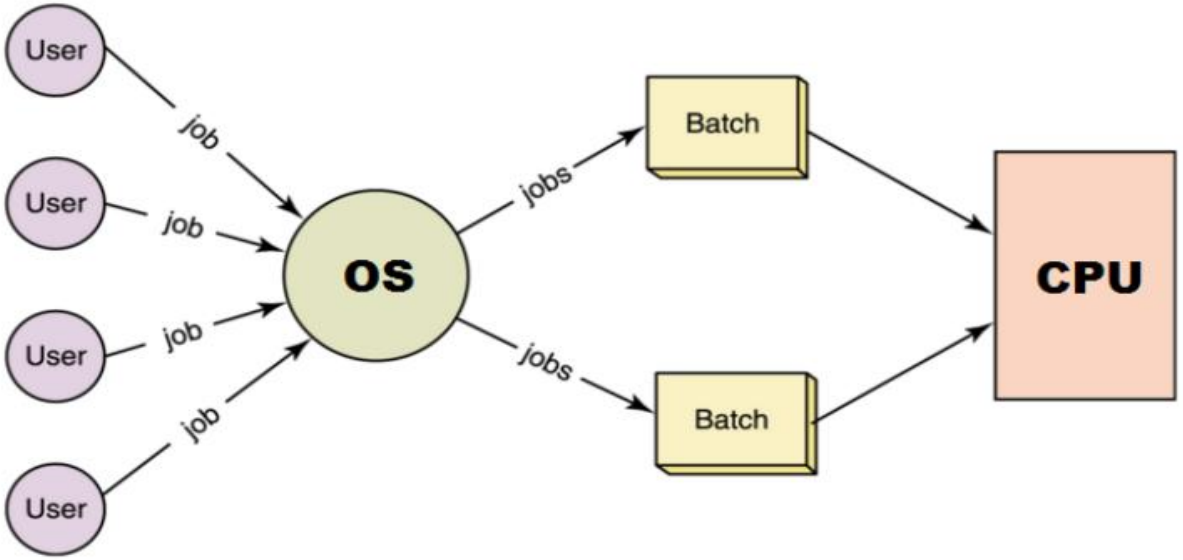
TYPES OF OS

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”.



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

Batch Processing: The Batch Processing is **same as the Serial Processing Technique.** But in the Batch Processing **Similar Types of jobs are Firstly Prepared** and they are Stored on the Card. and that card will be Submit to the System for the Processing. The System then Perform all the Operations on the Instructions one by one. And a **user can't be Able to specify any input**. And **OS** wills increments his Program Counter for Executing the Next Instruction.

The **Main Problem is that the Jobs those are prepared for Execution must be the Same Type** and if a job requires for any type of Input then this will not be Possible for the user. And Many Time will be wasted for Preparing the Batch. The Batch Contains the Jobs and all those jobs will be executed without the user Intervention. And **Operating System will use the LOAD and RUN Operation**. **This will first LOAD the Job from the Card and after that he will execute the instructions**. By using the RUN Command.

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**

There are two other approaches of improving the system performance by overlapping – input, output and processing. They are as follows -

(*a*) Buffering

(*b*) Spooling.

We discuss these one by one now.

(*a*) **Buffering**

It is a method of overlapping input, output and processing of a single job. The idea is quite simple. After data has been read and the CPU is about to start operating on it, the input device is instructed to begin the next input immediately. The CPU and the input device are then both busy. By the time that the CPU is ready for the next data item, the input device will have finished reading it. The CPU can then begin processing the newly read data, while the

There are two other approaches of improving the system performance by overlapping – input, output and processing. They are as follows -

(*a*) Buffering

(*b*) Spooling.

We discuss these one by one now.

(*a*) **Buffering**

It is a method of overlapping input, output and processing of a single job.

(*b*) **SPOOLING**

It stands for **S**imultaneous **P**eripheral **O**peration **Online**. 'Simultaneous' means, say if two or more users issue the print command and the printer can accept the requests even if it is printing some other job.

In disk technology, rather than the cards being read from the card reader directly into memory and then the job being processed, cards are read directly from the card reader onto the disk. The location of the card images is recorded in a table kept by the OS. When a job is executed, the OS satisfied its request for card reader input by reading from the disk. Similarly, when the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the jobs is completed, the output is actually printed. **This form of processing is called as spooling.** This is shown below in Fig. 1.2.

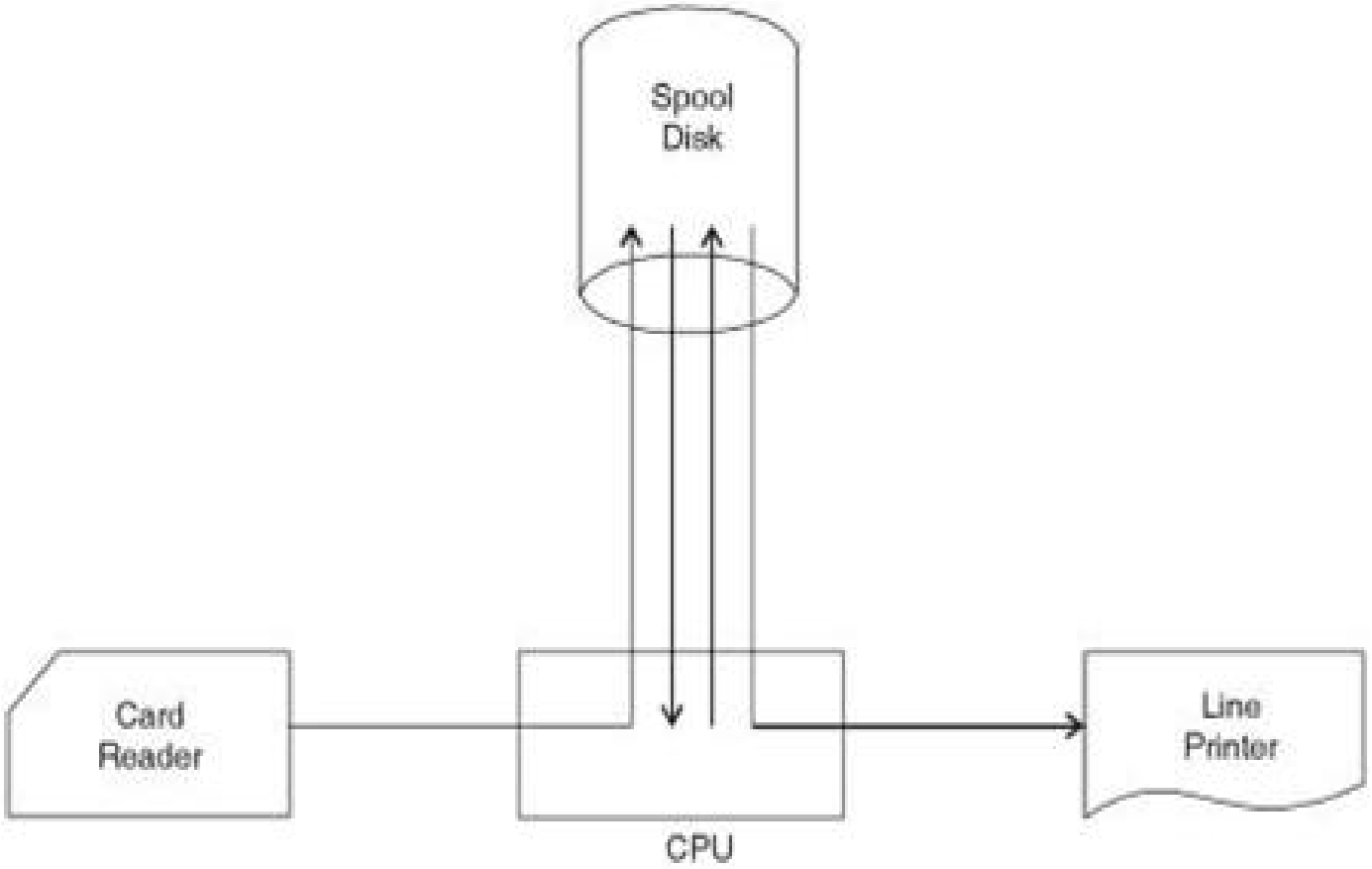


Fig. 1.2. Spooling

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

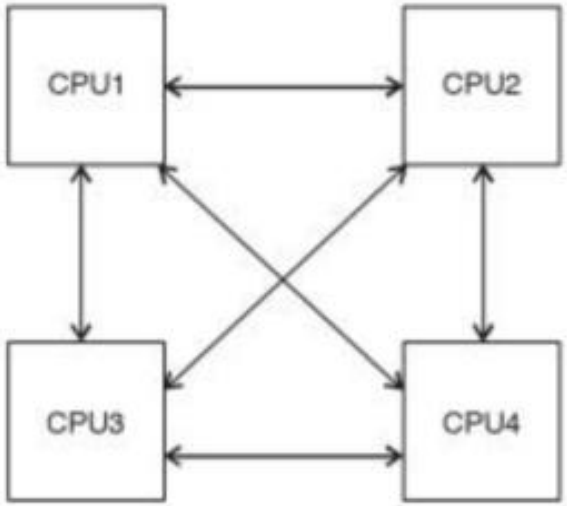
- Multi-processor systems; that is, they **have multiple CPU.**
- **Also known as parallel systems** or **tightly coupled systems**
- **Such systems have more than one processor** in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

We can thus, have two types of multiprocessing sysems—

(*a*) Symmetric Multiprocessor system.

(*b*) Asymmetric Multiprocessor system.

In symmetric multiprocessor configuration (Fig. 1.5), all CPUs are essentially identical and perform identical functions.



(Fig. 1.5),

Herein, any CPU can initiate an I/O operation or can handle any external interrupt and can run any process in the system. So, all the processors are potentially available.

In asymmetric multiprocessor configuration, different processors do different things. We may design in such a fashion that one processor is made a supervisor or Master and it controls other CPUs. This is shown in Fig. 1.6. below—

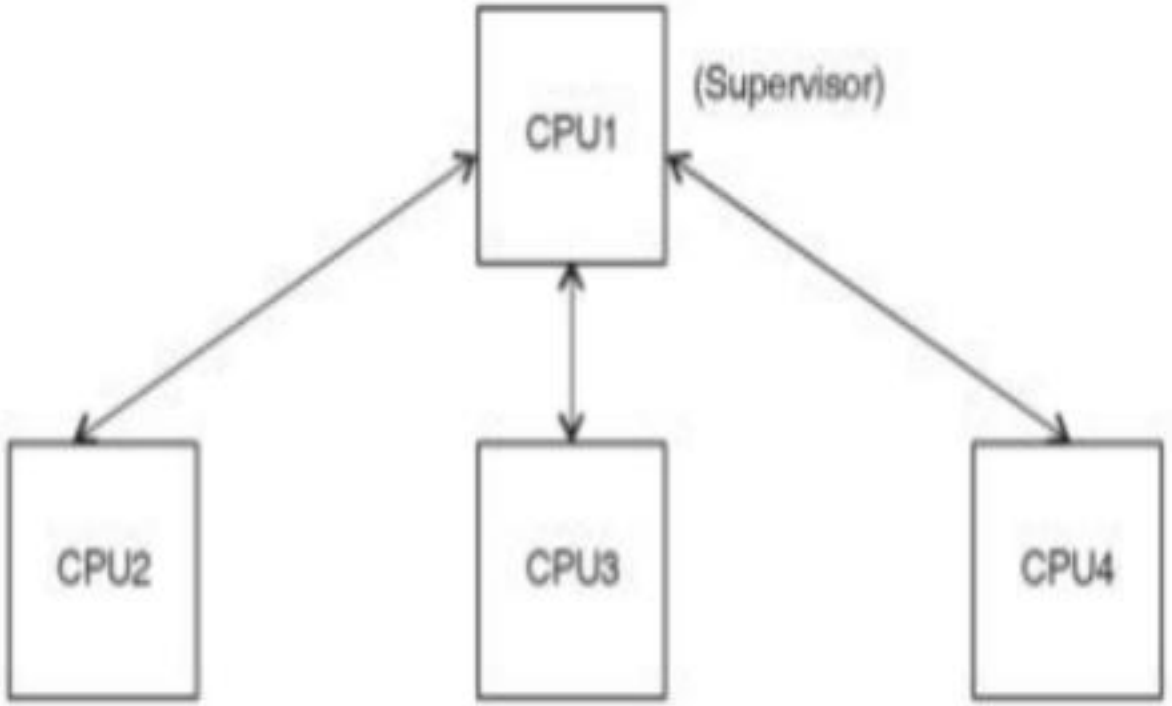


Fig. 1.6. Asymmetric multiprocessing

On one hand, processing may be very fast as then many processors may be available for a single job. On the other hand, processing may be slow because all of the processors (CPU) may not be available to handle peak loads. Also, if the supervisor CPU fails then the entire system will go down.

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

Distributed Means Data is Stored and Processed on Multiple Locations. When a Data is stored on to the Multiple Computers, those are placed in Different Locations. Distributed means In the Network, Network Collections of Computers are connected with Each other. Then if we want to Take Some Data From other computer, Then we uses the Distributed Processing System. And we can also Insert and Remove the Data from out Location to another Location. In this Data is shared between many users. And we can also Access all the Input and Output Devices are also accessed by Multiple Users.

Distributed Operating System

A **distributed environment** refers to a collection of autonomous systems, capable of communicating and cooperating with each other through the network, are connected to each other through LAN/WAN. A distributed OS governs such a distributed system and provides a **virtual machine abstraction** to its users. A distributed OS is one that looks to its users like an ordinary centralized OS but runs on multiple independent CPUs. **Please note that the key concept here is transparency.**

It means that the use of multiple CPUs should be invisible to the user, the resource distribution must be hidden from the users and the application programs unless required by the user. **Please note here that the user views the system as virtual uniprocessor and not as a collection of distinct machines. Also note that a true distributed system is one in which users are not aware of where their programs are being run or where their files are residing. It should all be handled automatically and efficiently by the OS.**

Distributed OS often allow porgrams to run on several processors at the same time, thus requiring more complex CPU scheduling algorithms in order to achieve maximum CPU utilization. Also, these distributed systems are more reliable than uniprocessor based system. They perform even if certain part of the hardware is malfunctioning.

Examples of Distributed OS

- Amoeba
- Angle
- Alpha kernel
- Chorus

Advantages of Distributed OS

1. It allows users to access remote resources in the same manner as they do with local resources.

Comparison of Network OS and Distributed OS – tabular form

Network OS	Distributed OS
1. In these systems, the control over file placement must be done manually by the users.	1. In these systems, it can be done automatically by the system itself.
2. In network OS, there are various machines each with its oven user-id (UID) mapping.	2. In distributed OS, there is a single system wide mapping that is valid everywhere.
3. We can realize a network OS if we put a layer of software on top of the native OS of individual machines.	3. A distributed OS governs the operation of a distributed system.
Examples: BSD, MS-LAN MANAGER, WIN-NT etc.	**Examples:** Amoeba, Angle, Chorus, Alpha kernel etc.

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…

We shall tabulate the differences between them

Real time System	Soft Real time system
1. Autonomous error detection for hard real time system.	1. User assisted error detection for soft real time system.
2. These systems have limited utility for rollback/recovery mechanism due to short-term integrity of data.	2. These systems have long-term integrity of data.
3. The data file sizes are small/medium.	3. The data file sizes are large.
4. Their safety is often critical.	4. Their safety is non-critical.
5. Predictable (well defined) peak load performance for these systems.	5. Degraded peak load performance.
Examples:	**Examples:**
(*a*) Air Traffic Control	(*a*) Telecom (Voice)
(*b*) An Embedded system controlling the operation of a refinery.	(*b*) Delayed information about flight path will lose its utility. It is a soft system as it will not be catastrophic but the output loses its utility.

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.

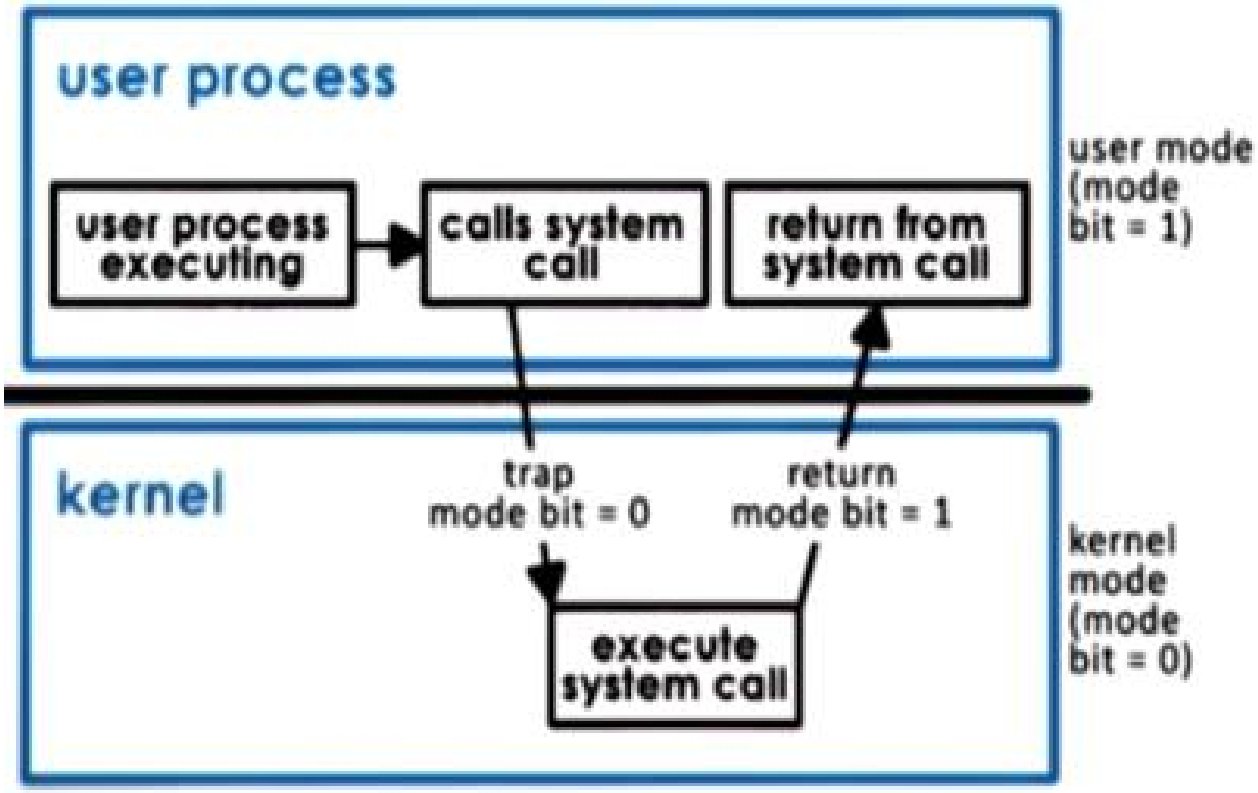
- Program or **software send, Generate Events by using system calls**. Error or request by a software creates **exception** or **trap**
  - E.g. Division by zero

- **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

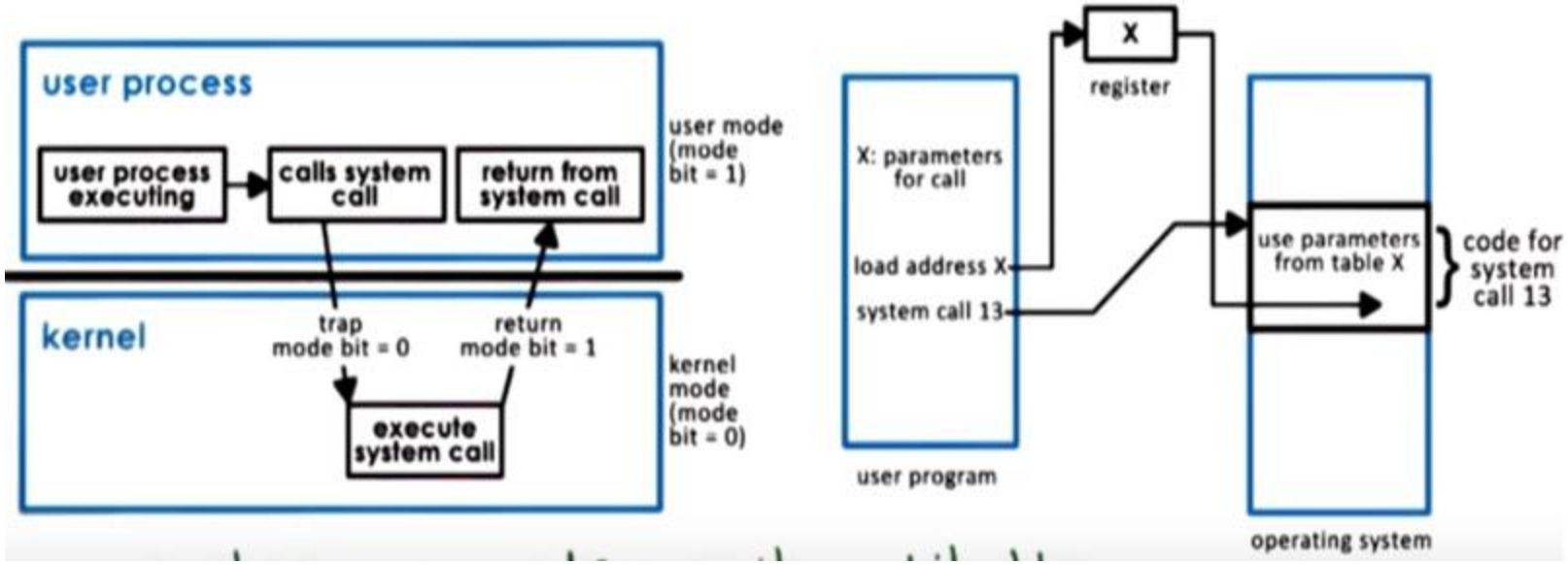
Transition from User to Kernel Mode



To make a system call an application must

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

Transition from User to Kernel Mode



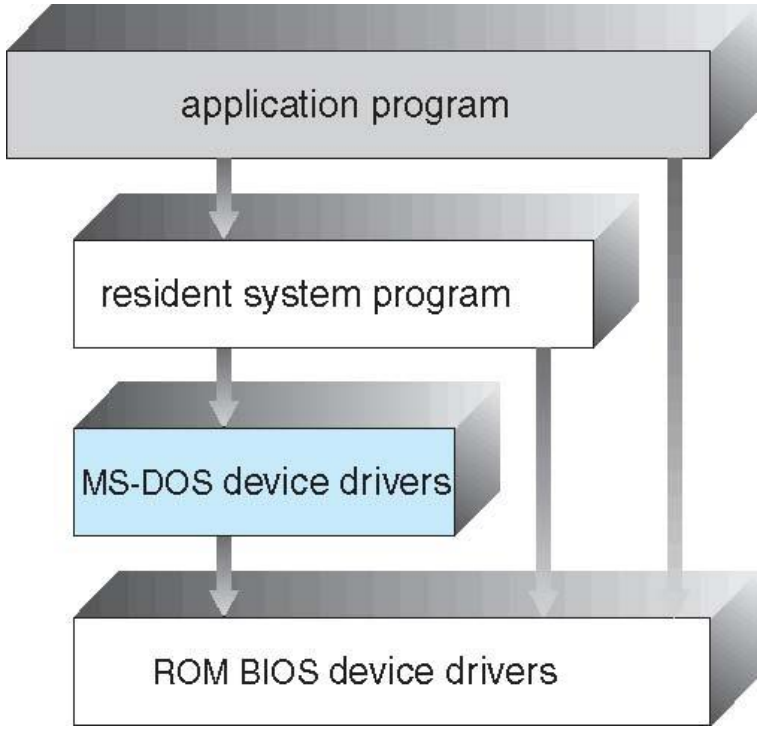
Operating System Structure

A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

- **Simple Structure**
- **Layered approach**
- **Microkernel**

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



Key point:

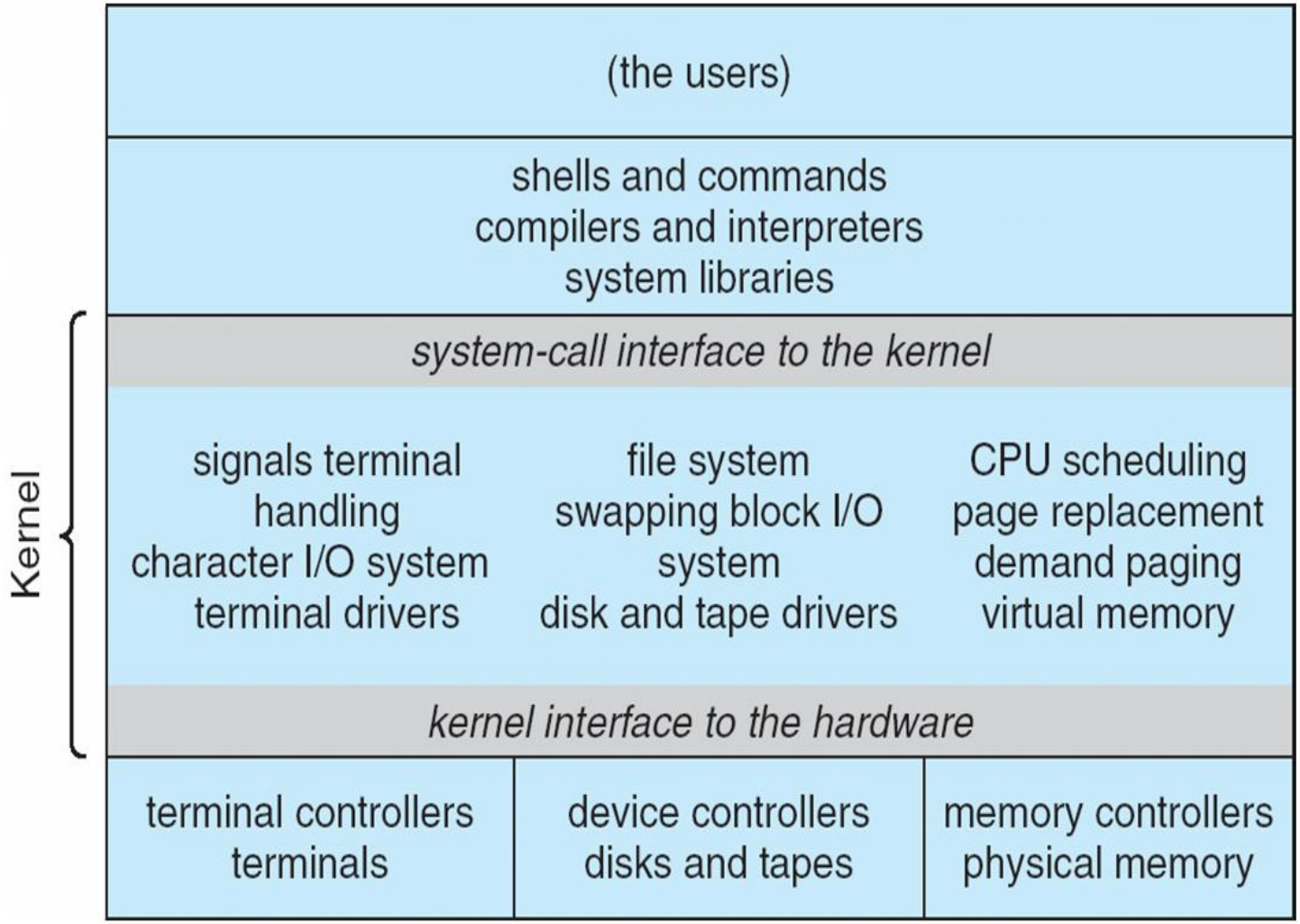
- MS-DOS – written to provide the most functionality in the least space
- Not divided into modules Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.
- No dual mode and no hardware protection (Intel 8088) in MS- DOS.

Non Simple Structure -- UNIX

The UNIX OS consists of two parts:

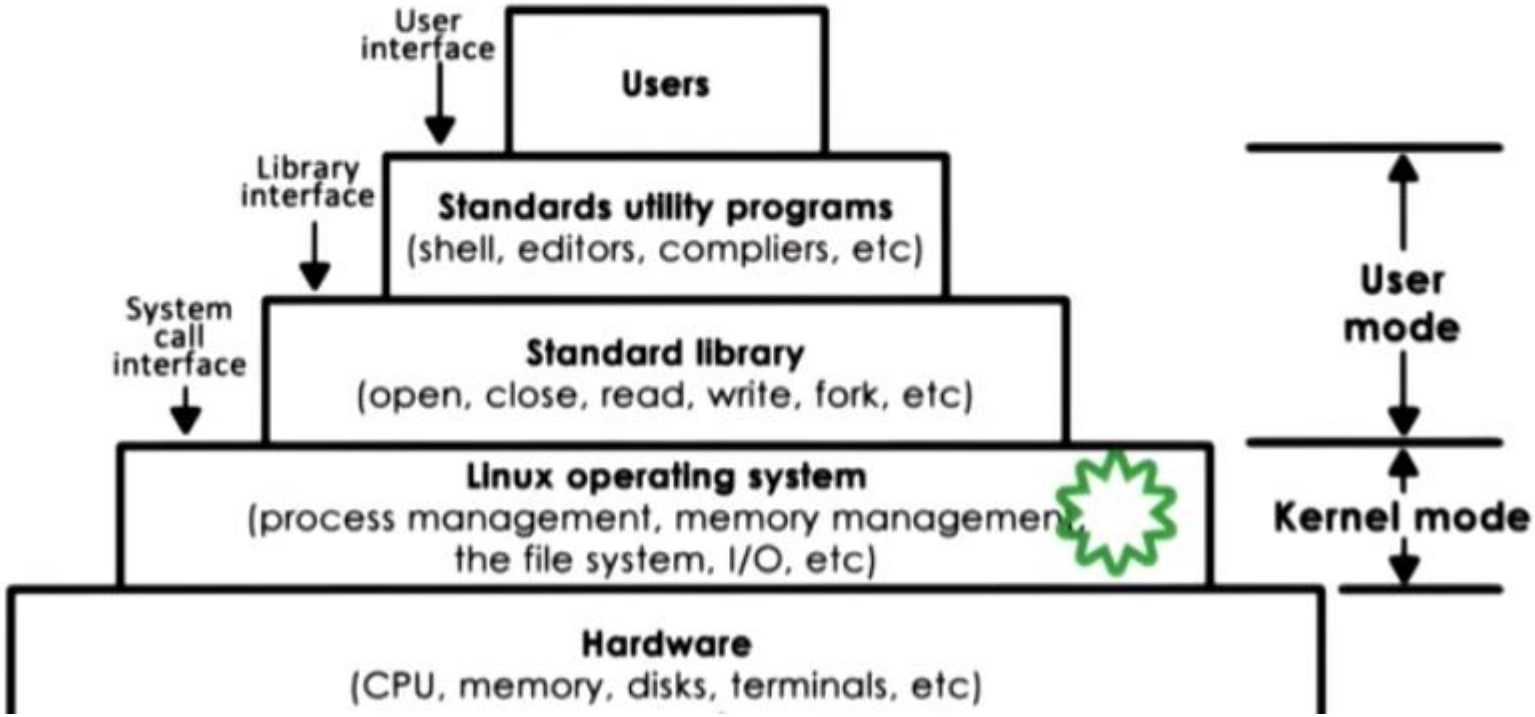
- System programs
- The kernel
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Traditional UNIX System Structure

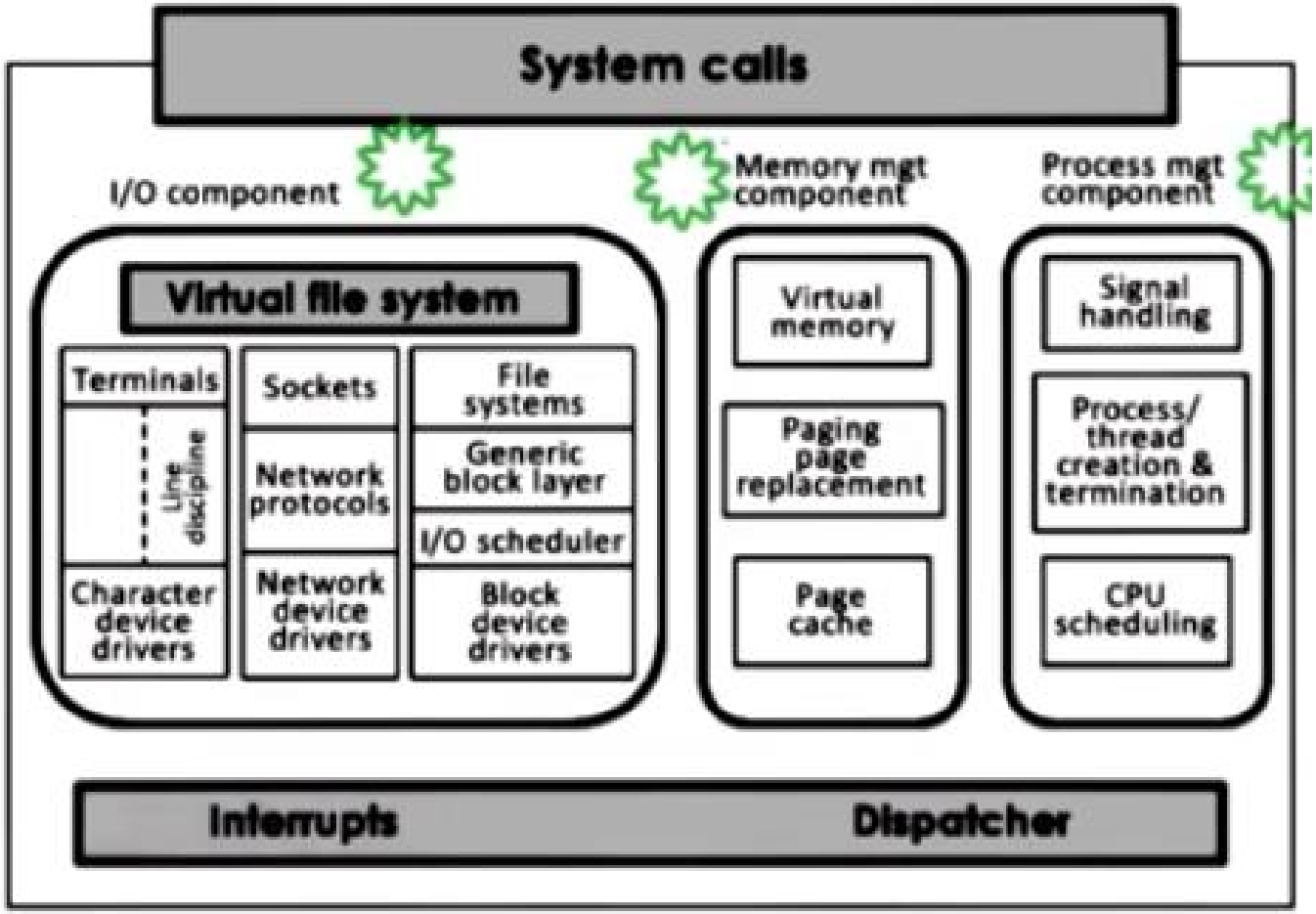


Traditional LINUX System Structure

Linux Architecture

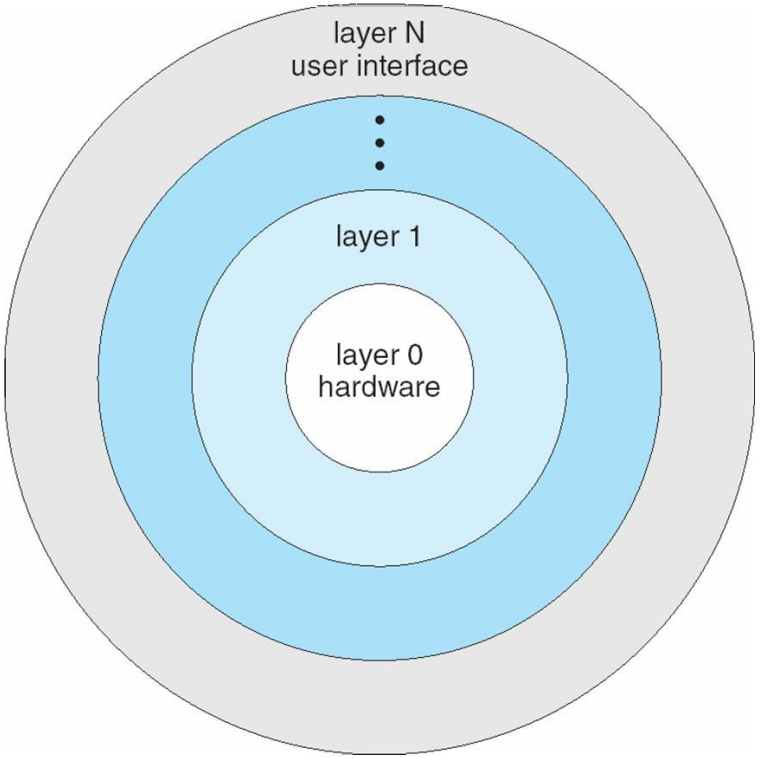


Kernel has many inbuilt modules



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



Key points:

• 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.

• Main advantage of layered approach:

– First layer can be debugged without any concern, because it uses only basic hardware.

– Once the first layer is debugged, its correct functioning while second layer is worked on and so on.

– If an error occurs we know in which layer.

– Each layer is implemented using those operations provided by lower-level layers.

– A layer does not need to know how the low-level operations are implemented; it needs to know what these operations do.

– Each layer hides the existence of data structures, operations and hardware from higher-level layer.

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)

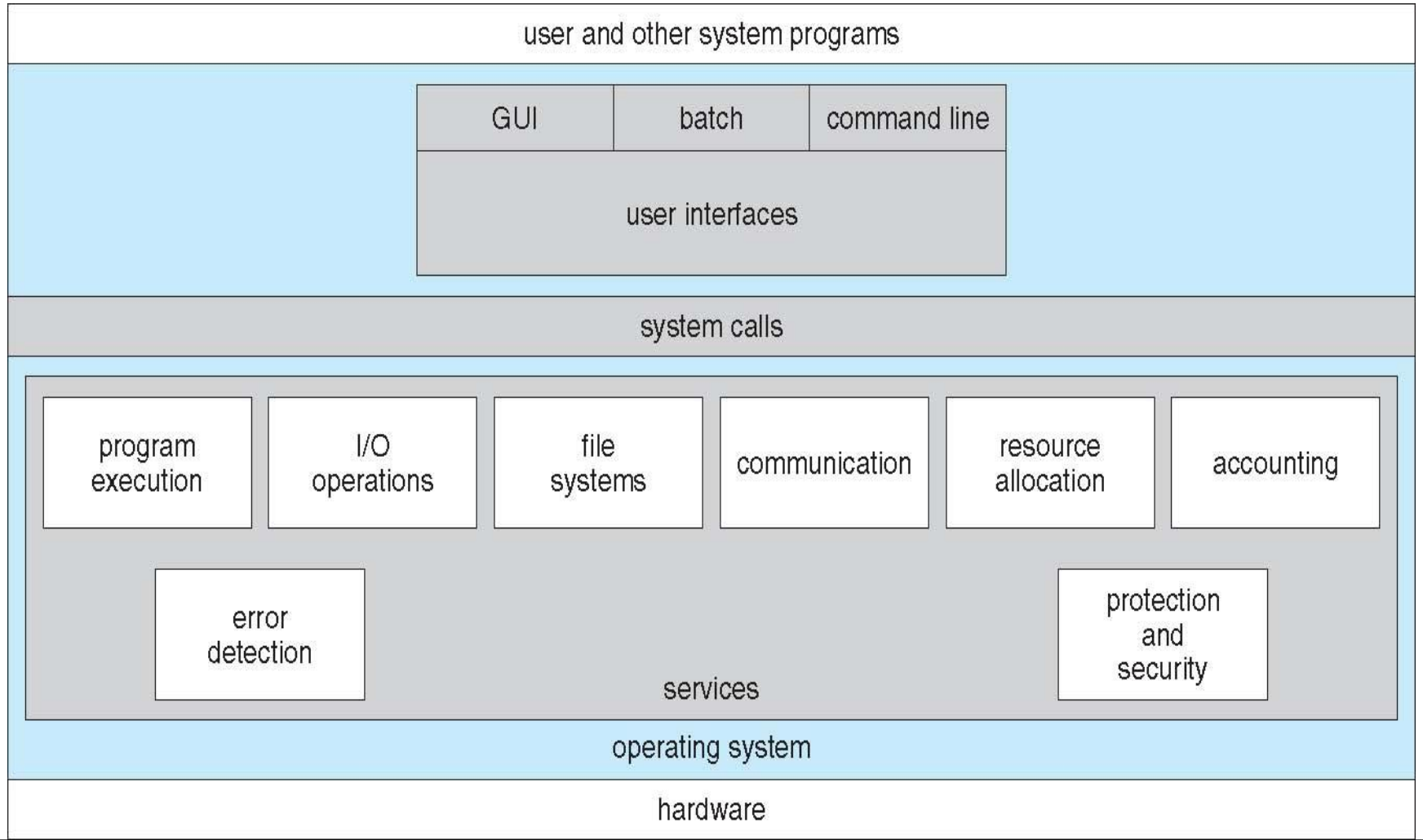
Layered Structure of the THE OS

- A layered design was first used in THE operating system.
  Its six layers are as follows:

layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	operator-console device driver
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

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

Kernel **:** An Operating system is a well organized collection of a basic set of program.

The most important program in the collection is called kernel of Operating system (the kernel is also called supervisor monitor or nucleus). The kernel is the minimal OS program that is loaded in the main memory when computer is initialized during bootstrapping. It resides permanently in the main memory until the system is shutdown. It typically resides in the low memory address part of the main memory. The term operating system is defined as the notion of the kernel (the case).

The supporting hardware management software, the various system libraries and the specialized system application is called utilities.

The kernel is the controlling authority and provides care Operating system functionalities. It typically manages hardware components and export services such as memory management and I/O device management. It provides the generic interface for the rest of Operating system programs. The non-kernel parts of Operating system are loaded into and unloaded from the main memory as the situation demands during run time(In some system these other part are supported the rough system applications).

The kernel contains all the critical functionalities of an Operating system which are the needed for the effective and smooth function of a computer. However in modern monolithic system the demarcation between the Operating system and the kernel is blushed unless stated otherwise we will use the terms kernel and Operating system to mean the same.

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
- Nano kernels
- Exokernels
- reentrant

Monolithic kernels

- The OS kernel is a single (monolithic) structure
  - executing entirely in supervisor mode
    - process management
    - memory management
    - drivers for hardware
  - provides a set of system calls to interface with operating system services
  - may use modules to optimize resource utilization

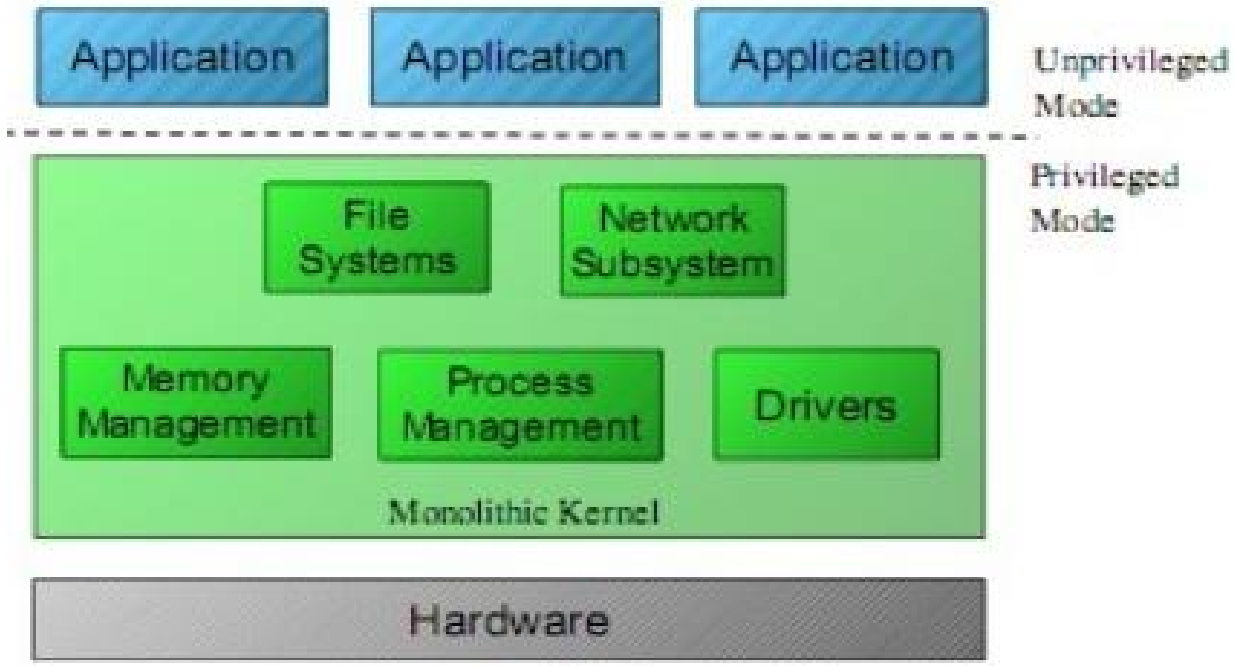
Monolithic Kernel

- 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, Unix kernels , Solaris, OS-9, DOS, Microsoft Windows (95,98,Me) , OpenVMS, XTS-400etc.

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.

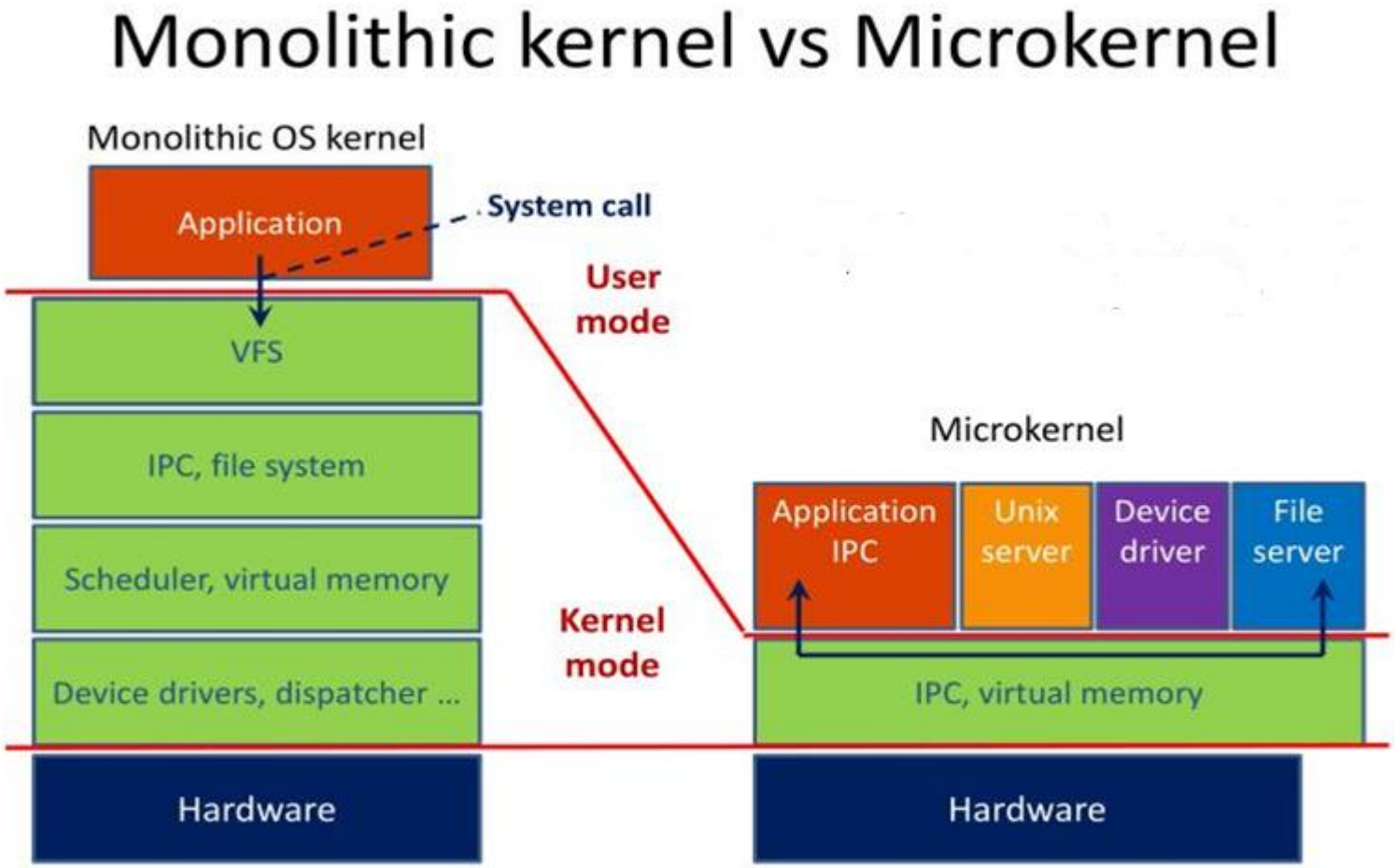
Cons:

- 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

Nano kernels:

A nanokernel delegates virtually all services — including even the most basic ones like interrupt controllers or the timer — to device drivers to make the kernel memory requirement even smaller than a traditional microkernel.

Nano kernel: **Nanokernel** is hardware access without IPC s sometimes. Or a microkernel that is really micro as opposed to MINO kernels (Micro In Name Only). Kind of a marketing term.

Nanokernels are relatively small kernels which provide hardware abstraction, but lack system services. They provide the very basic OS functionalities and rest is implemented as applications

A nanokernel is a small kernel that offers hardware abstraction, but without system services. Larger kernels are designed to offer more features and manage more hardware abstraction. Modern microkernels lack system services as well, hence, the terms microkernal and nanokernal have become analogous.

Exokernels :

Exokernels are a still experimental approach to operating system design. They differ from the other types of kernels in that their functionality is limited to the protection and multiplexing of the raw hardware, providing no hardware abstractions on top of which to develop applications. This separation of hardware protection from hardware management enables application developers to determine how to make the most efficient use of the available hardware for each specific program.

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

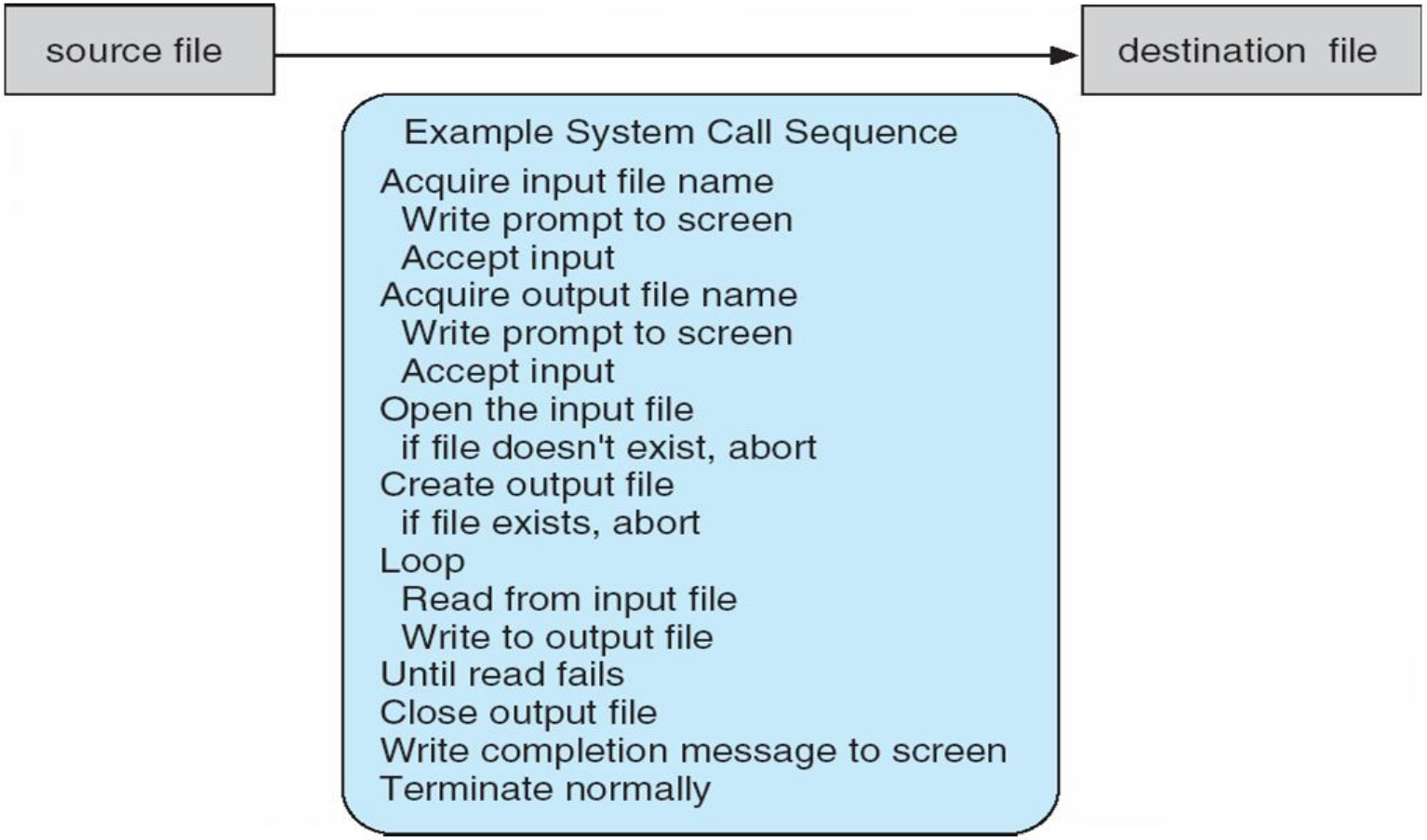
Accessing System Calls

- 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

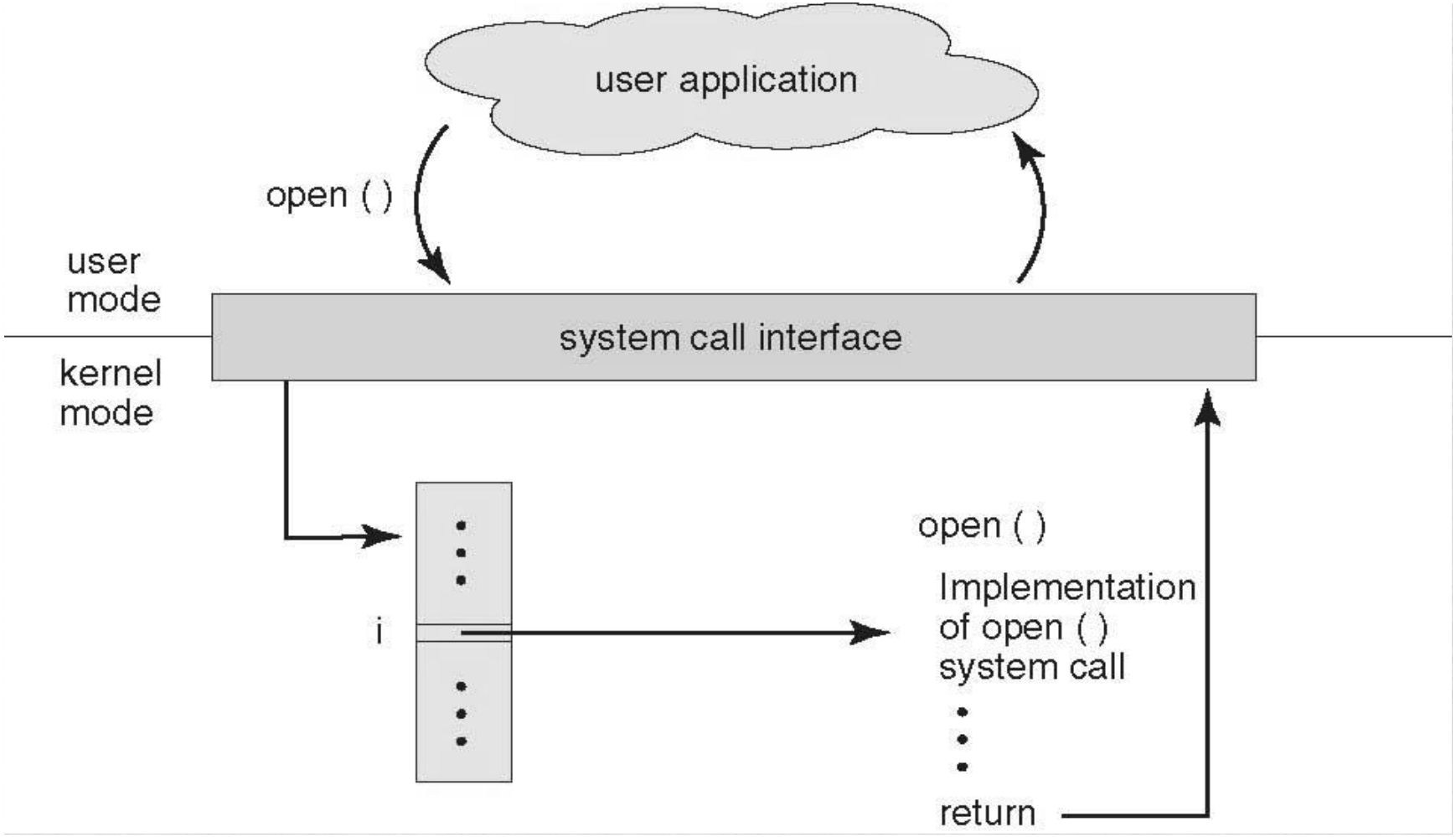
- System call sequence to copy the contents of one file to another file



System Call Implementation

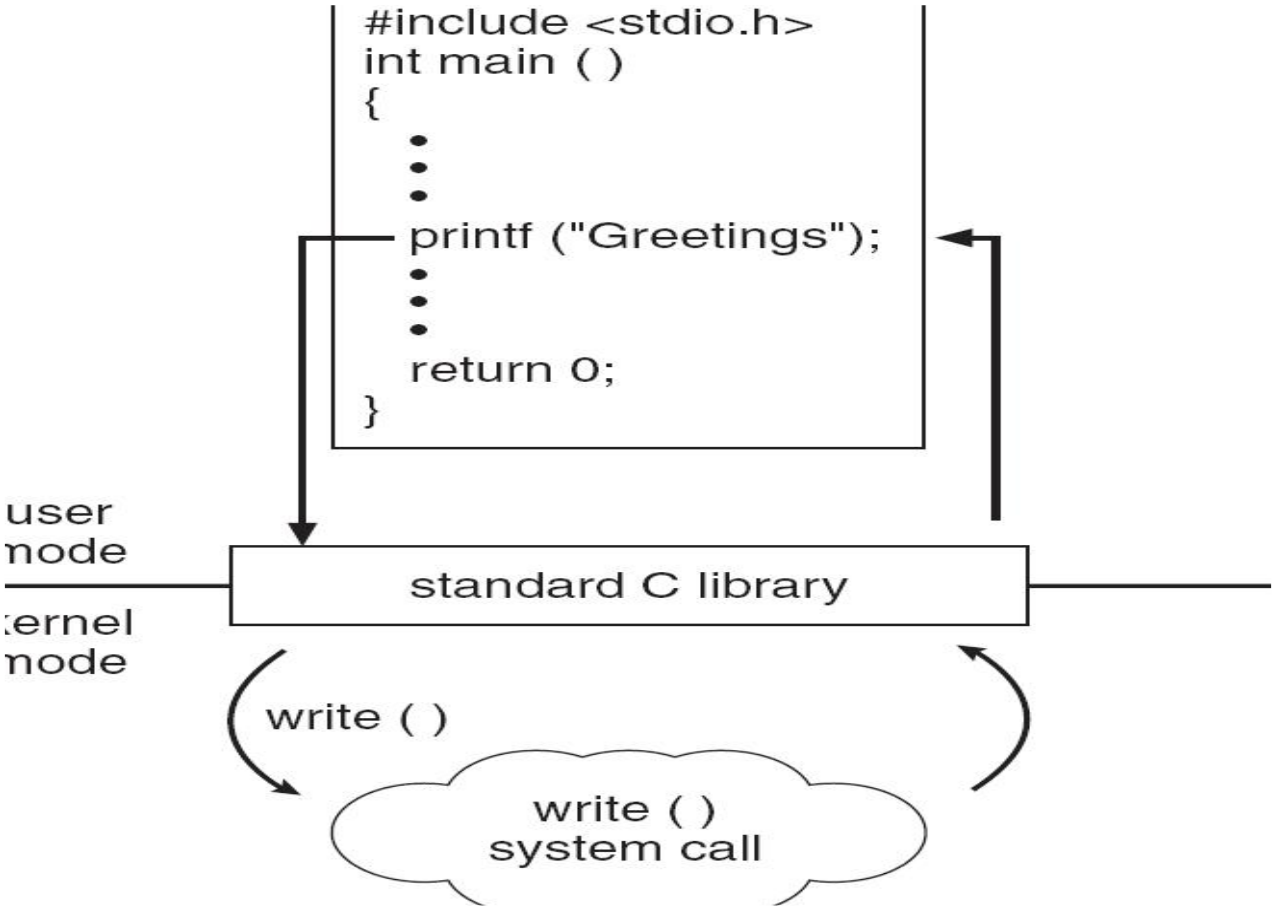
- A number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call with a return value.
- The caller needs to know nothing about how the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

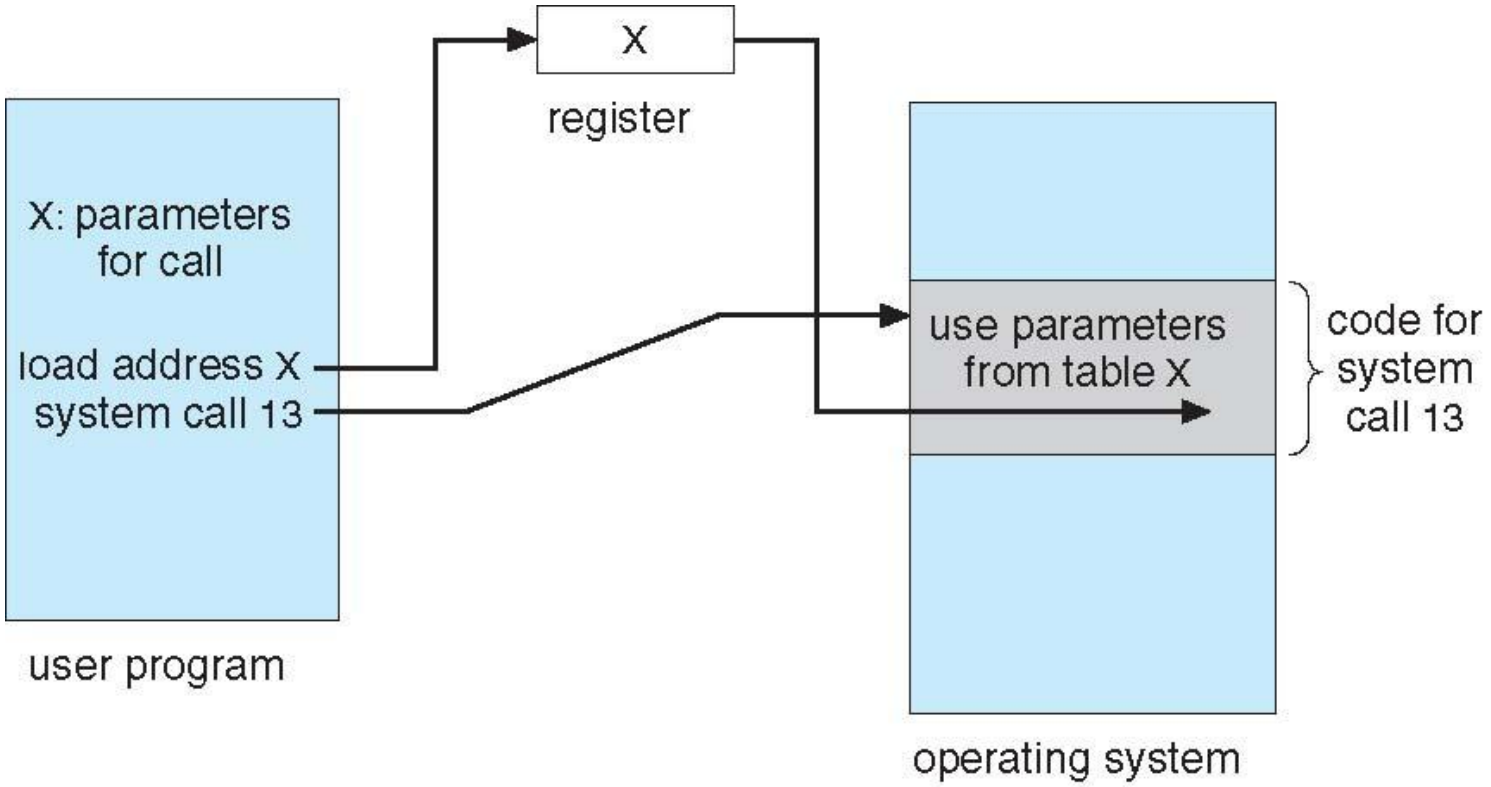
- **Three general methods used to pass parameters to the OS**

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

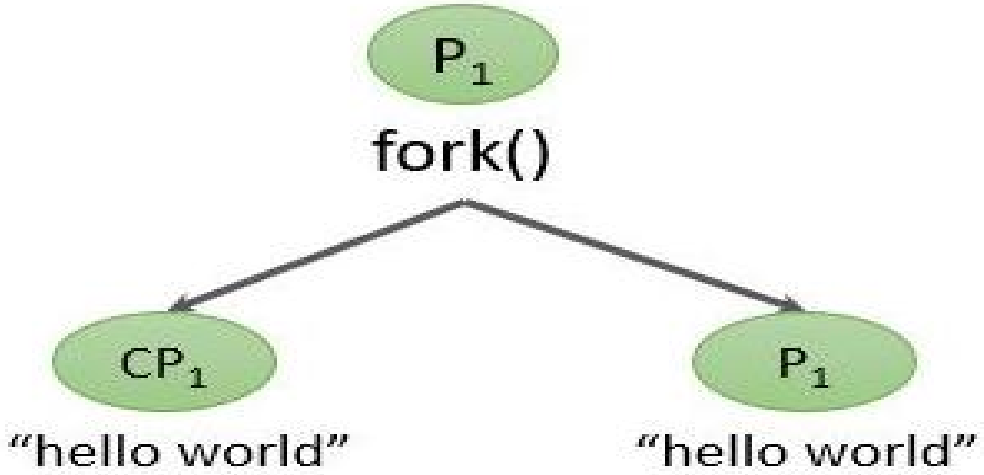
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

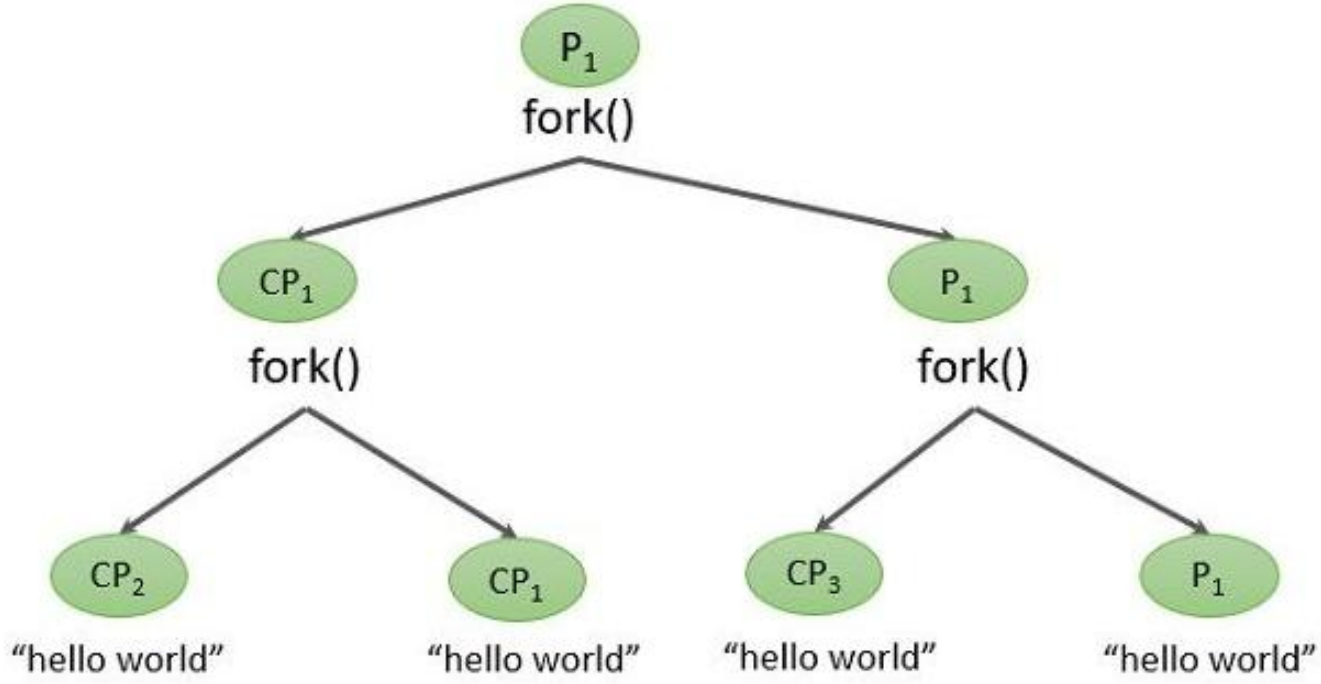
(Imp)Fork system call

- Fork system call is used for creating a new process
- which is called ***child process***
- which runs concurrently with the process that makes the fork() call (parent process)

Main Aim –

- To achieve multiprocessing




P1
fork()
CP1
P1
fork()
fork()
CP2
CP1
CP3
P1
"hello world"
"hello world"
"hello world"
"hello world"

Q1. Who controls the execution of programs to prevent errors and improper use of computer?

a) Resource allocator

b) Control Program

c) Hardware

d) None of the above

Q2. The operating system switches from user mode to kernel mode so the mode bit will change from?

a) 0 to 1

b) 1 to 0

c) Remain constant

d) None

Q3. In which type of operating system users do not interact directly with the computer system?

a) Multiprogramming operating systems

b) Multiprocessing operating systems

c) Batch operating systems

d) Distributed operating systems

Ans 3) c

Q4. What is the objective of multiprogramming operating systems?

a) Maximize CPU utilization

b) Switch the CPU among processes

c) Achieve multitasking

d) None of the above

Ans4. a)

Q5. Who signalled for the occurrence of an event either from the hardware or the software?

a) Bootstrap program

b) Interrupt

c) Disk Controller

d) CPU

Ans5: b

Q6. In which type of I/O interrupts the control return to the user program after the completion of I/O operation?

a) Synchronous I/O interrupts

b) Asynchronous I/O interrupts

c) System Call

d) Hardware

Ans 6) a

Q7: The device-status table contains

a) each I/O device type

b) each I/O device state

c) each I/O device address

d) all of the above

(d)

Ans 7 (d)

UNIX operating system

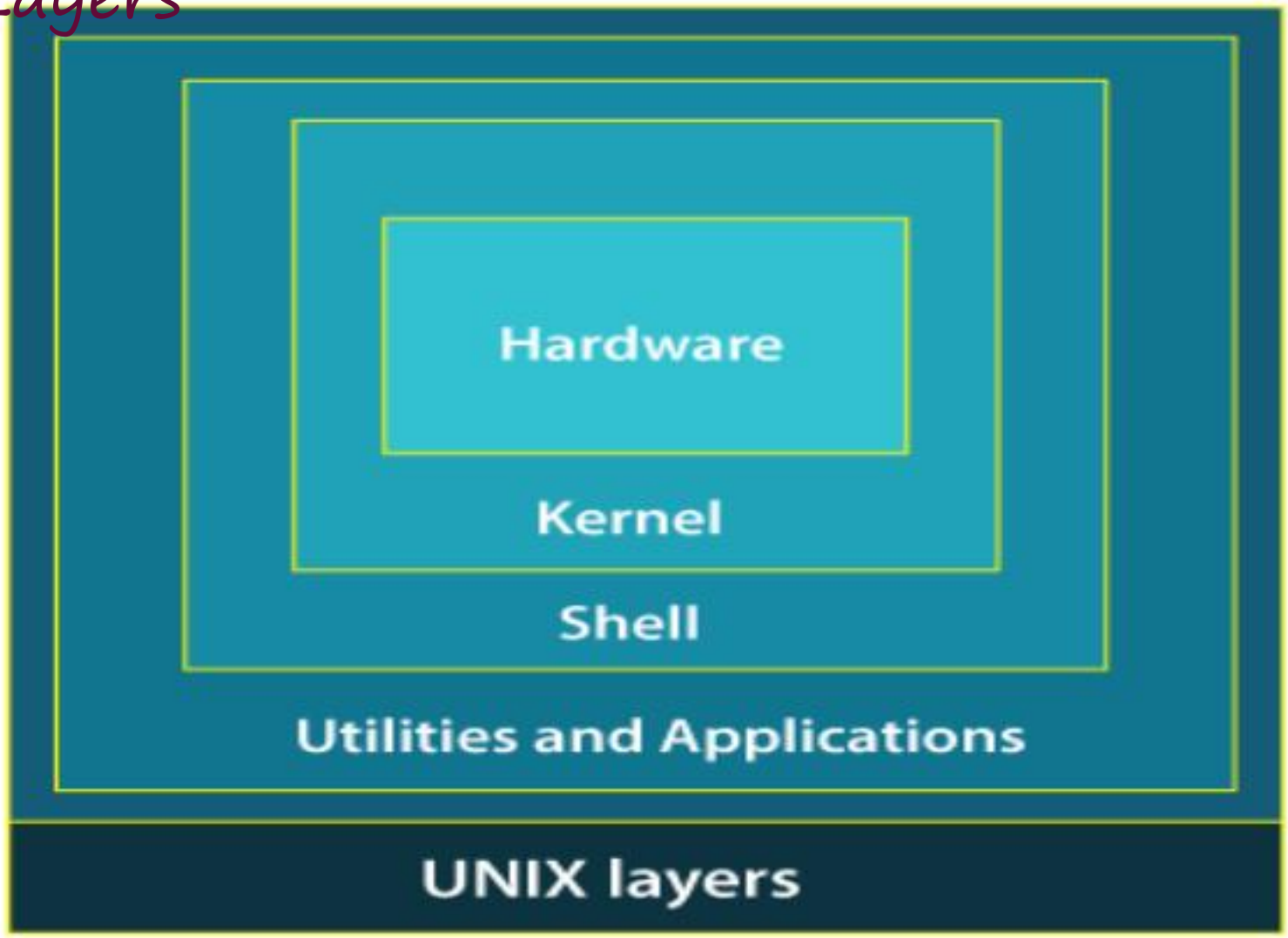
•UNIX is a powerful Operating System initially developed by Ken Thompson, Dennis Ritchie at AT&T Bell laboratories in 1970.

•It is prevalent among scientific, engineering, and academic institutions due to its most appreciative features like multitasking, flexibility, and many more.

• In UNIX, the file system is a hierarchical structure of files and directories where users can store and retrieve information using the files.

Features	Linux	Unix
Basic Definition	Linux is an open-source operating system. This OS is supported on several computer platforms and includes multiple software features that handle computer resources, and allow you to do tasks.	Unix is a powerful and multitasking operating system that behaves like a bridge between the user and the computer.
Launched by	This operating system was launched by Linus Torvalds at the University of Helsinki in 1991.	This operating system was launched in 1960 and released by AT&T Bell Labs.
OS family	It belongs to the Unix-like family.	It belongs to the Unix family.
Available in	It is available in multiple languages.	It is available in English.
Kernel Type	It is monolithic.	It can be microkernel, monolithic, and hybrid.
Written in	C and other programming languages.	C and assembly language.
File system support	It supports more file systems than Unix.	It also supports less than Linux.
Usage	It is used in several systems like desktop, smartphones, mainframes and servers.	Unix is majorly used on workstations and servers.
Examples	Some examples of Linux are: Fedora, Debian, Red Hat, Ubuntu, Android, etc.	Some examples of unix are IBM AIX, Darwin, Solaris, HP-UX, macOS X, etc.
Security	Linux provides higher security.	Unix is also highly secured.
Price	Linux is free and its corporate support is available at a price.	Unix is not totally free. There are some Unix versions that are free,

The structure of Unix OS Layers



Chapter : Processes

Chapter : Processes

Process Concept

Process Scheduling

Operations on Processes

Process

A program is a passive entity, such as a file containing a list of instructions stored on disk.

Process is an instance of a program.

A process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources.

Process

Process – a program in execution

Process execution must progress in sequential manner

A process includes:

- program counter
- stack
- data section

Process

Process memory is divided into four sections for efficient working :

The **Text section** is made up of the compiled program code

The **Data section** is made up of the global and static variables

The **Heap** is used for the dynamic memory allo

The **Stack** is used for local variables.

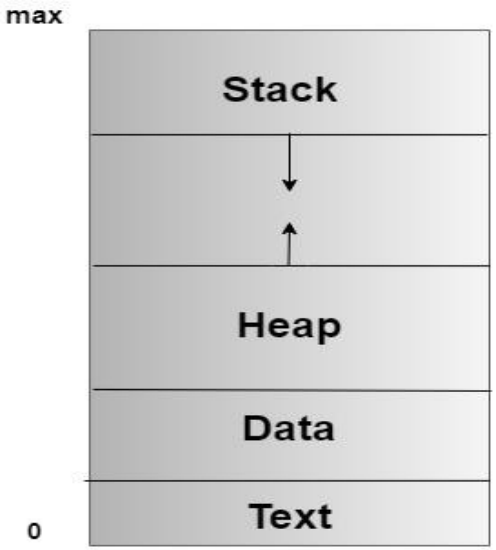


Figure: Process in the Memory

Process States

As a process executes, it changes *state*

new: The process is being created

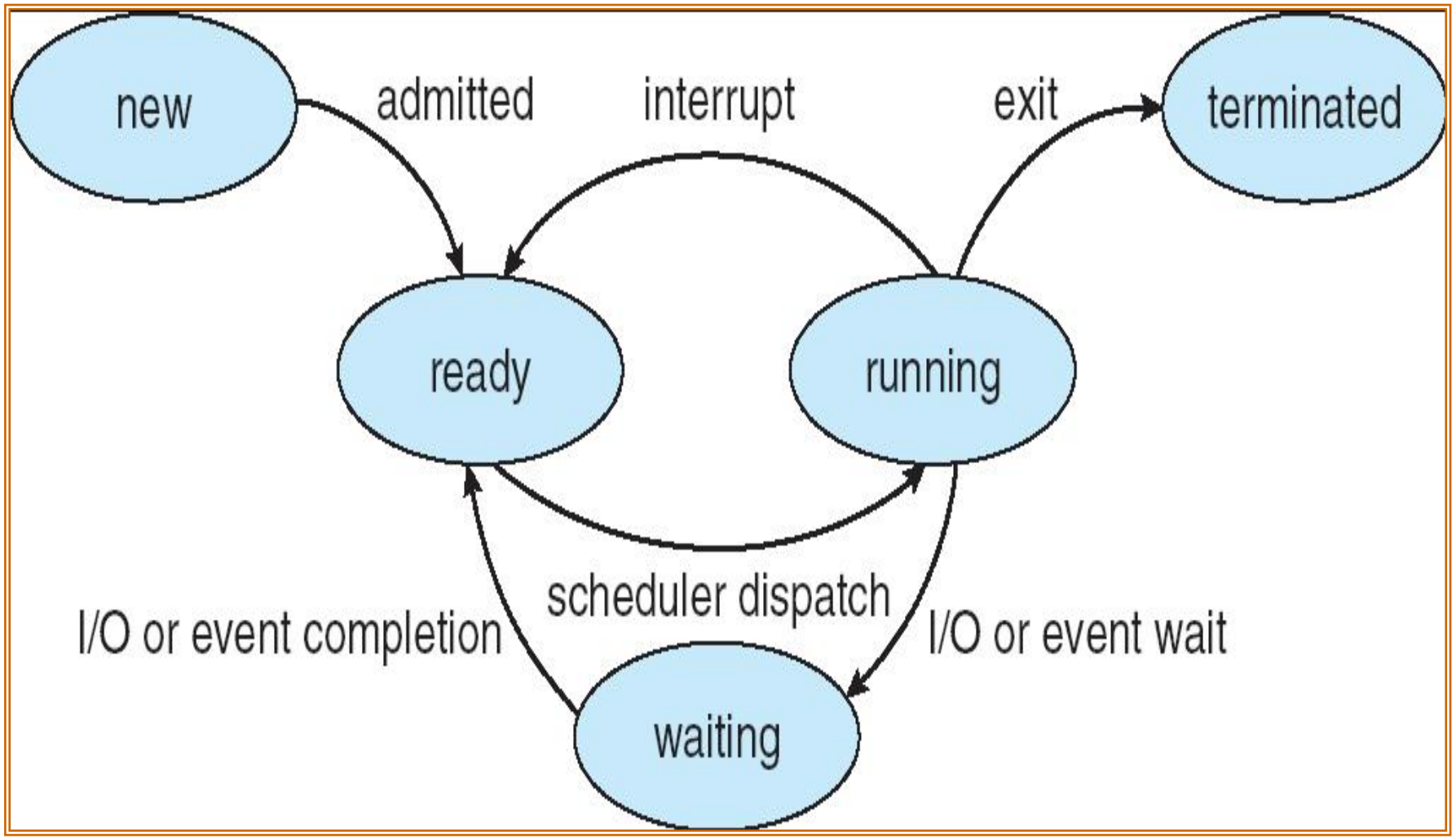
ready: The process is waiting to be assigned to a processor and is ready to get executed

running: Instructions are being executed

waiting: The process is waiting for some event to occur/ waiting for resources

terminated: The process has finished execution

Process States



Process Control Block (PCB)

Process Control Block (PCB, also called Task Controlling Block) is a data structure in the operating system kernel containing the information needed to manage a particular **process**.

The PCB is "the manifestation of a **process** in an operating system".

Process Control Block (PCB)

Information associated with each process:

1. **Process State-** new, ready….etc.
2. **Pointer-** to the parent process
3. **Program Counter-** next instruction to be executed.
4. **Process Number- Unique identification** number in OS
5. **CPU Registers-** Various CPU registers **where process** need to be **stored for execution** for running state.
6. **CPU Scheduling Information-** Process Priority and other info. required for scheduling

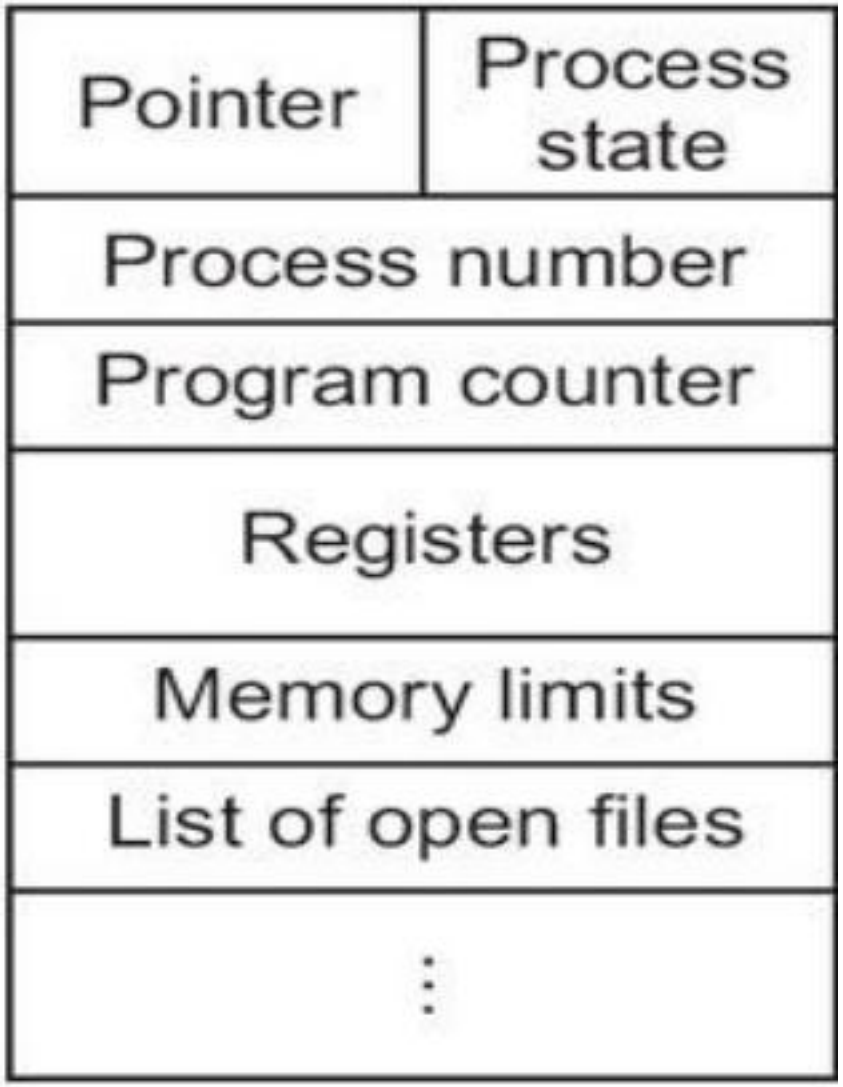
Process Control Block (PCB)

7. **Memory-Management Information-** Registers, Page tables (where process is saved)used by OS

8. **Accounting Information-** For how long CPU and other resources are allocated to process

9. **I/O Status Information-** List of devices allocated to the process

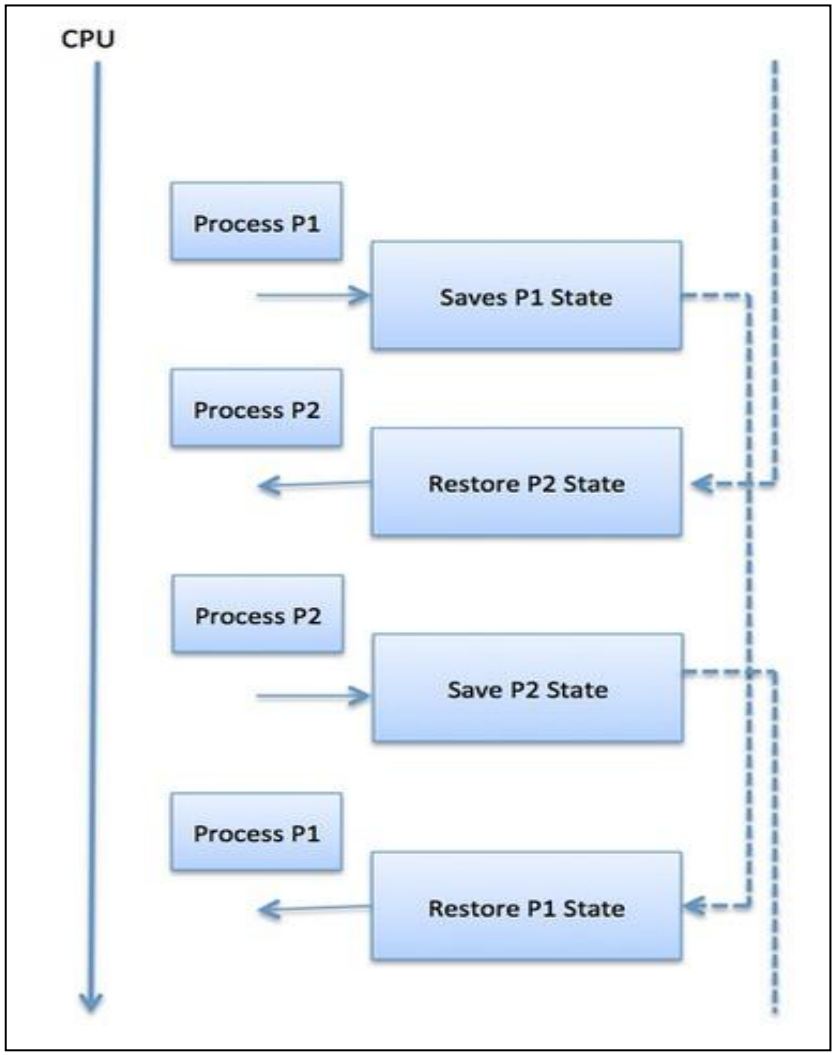
Process Control Block (PCB)



Process Scheduling

Context Switch

When CPU switches to another process, the system must save the state of the old process in the stack and load the saved state for the new process or reload the state of current process from stack.



Context Switch

Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **context switch**.

The **context** of a process is represented in the **Process Control Block(PCB)** of a process;

it includes the value of the CPU registers, the process state and memory-management information.

When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context Switch

Context switch time is **pure overhead**, because the **system does no useful work while switching**.

Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied

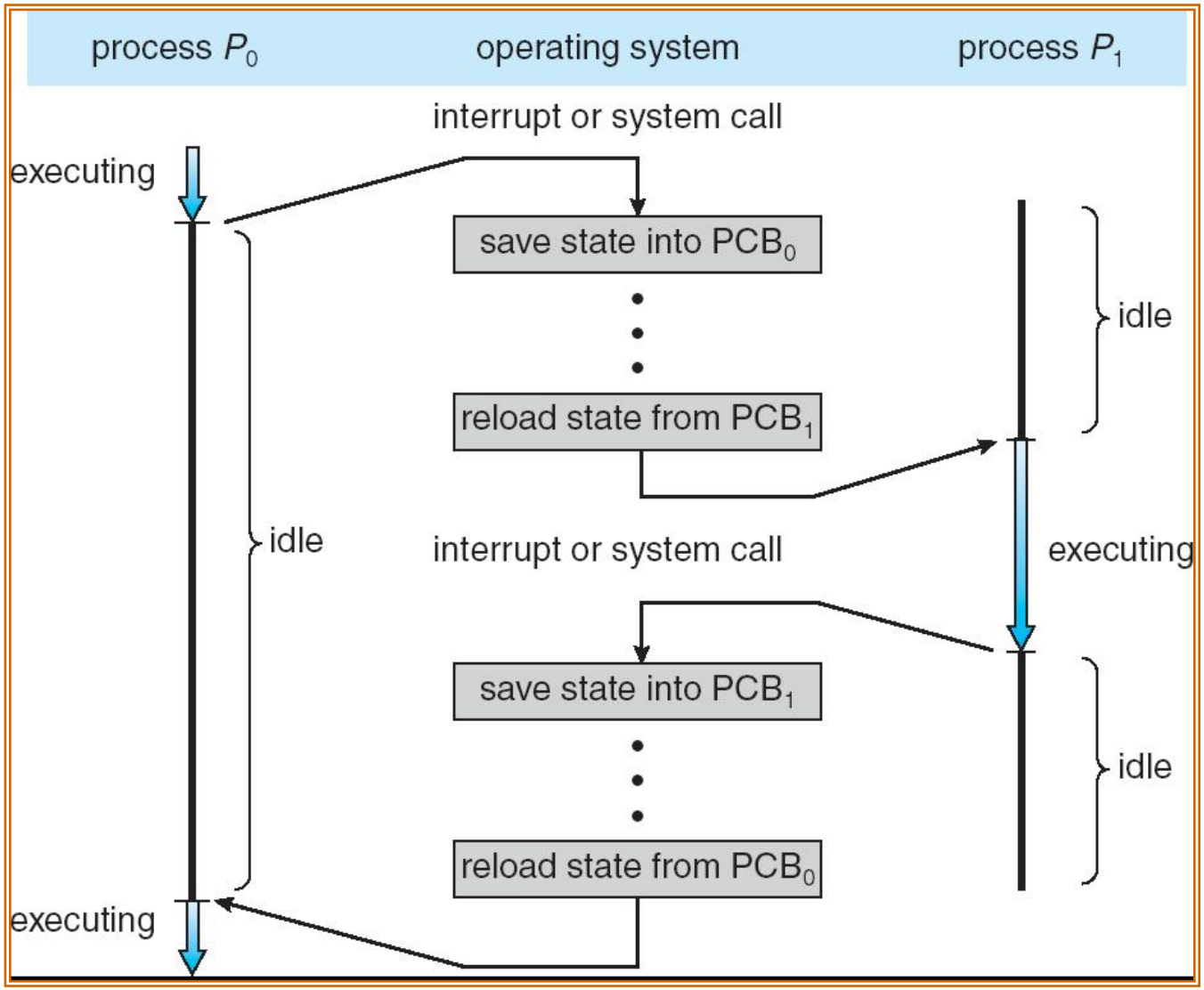
Typical speeds range from 1 to 1000 microseconds.

Context switching is expensive:

- Direct Cost: No. of cycles for load and store instructions
- Indirect Cost: When process run a lot of its data is stored on processor cache.

Must limit the frequency of context switching.

CPU Switch From Process to Process



Process Scheduling

The activity of determining which process in the ready state should be moved to the running state is known as Process Scheduling.

The prime aim of the process scheduling system is:

To keep the CPU busy all the time and to deliver minimum response time for all programs.

For achieving this, the scheduler must apply appropriate rules for swapping processes.

Process Scheduling

Schedulers fall into one of the two general categories:

Non pre-emptive scheduling: When the currently executing process gives up the CPU voluntarily.

Pre-emptive scheduling: When the operating system decides to favor another process, pre-empting the currently executing process.

Process Scheduling

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues.

The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.

When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

Process Scheduling Queues

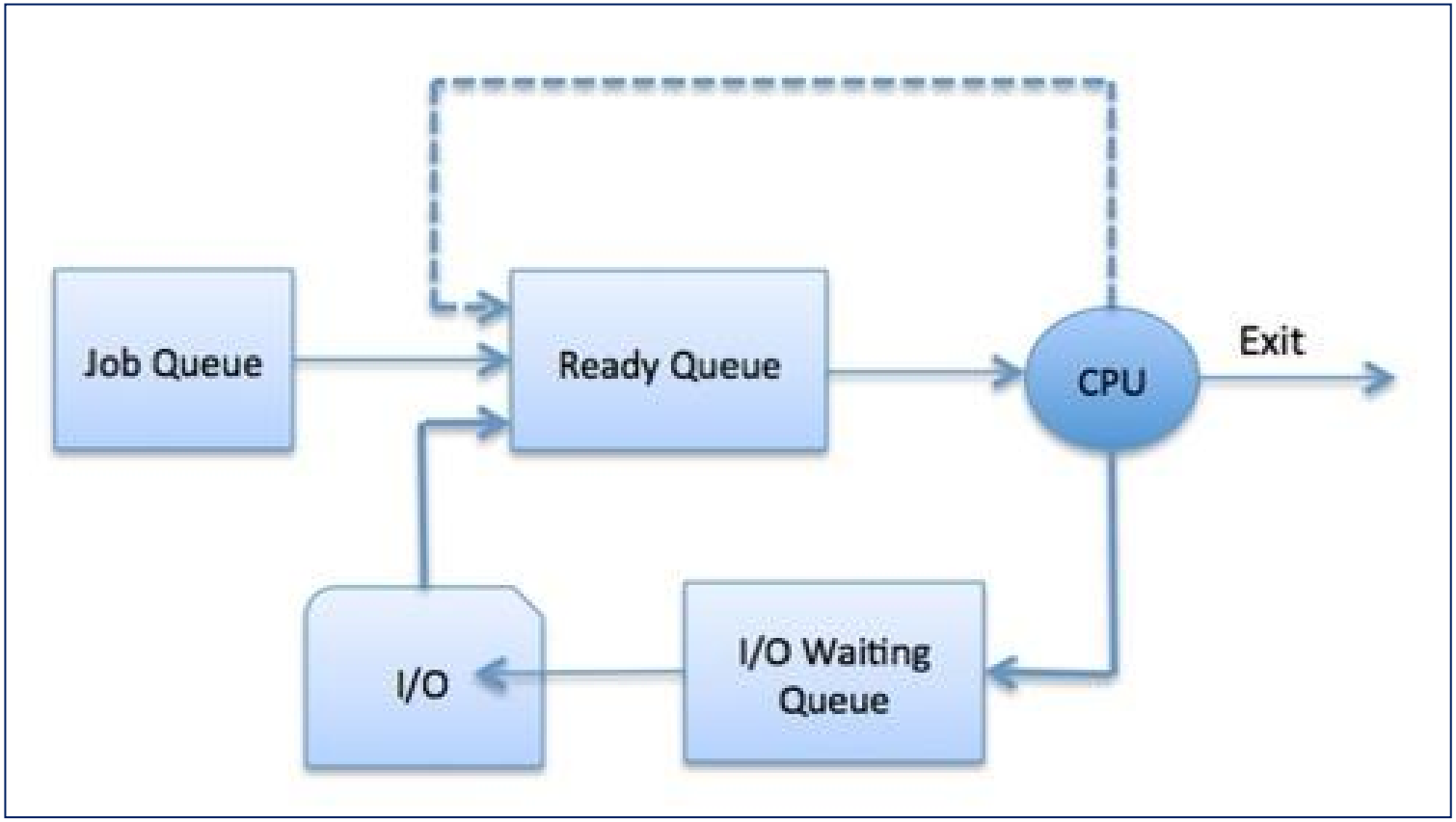
Job queue − This queue keeps all the processes in the system.

Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute.

A new process is always put in this queue.

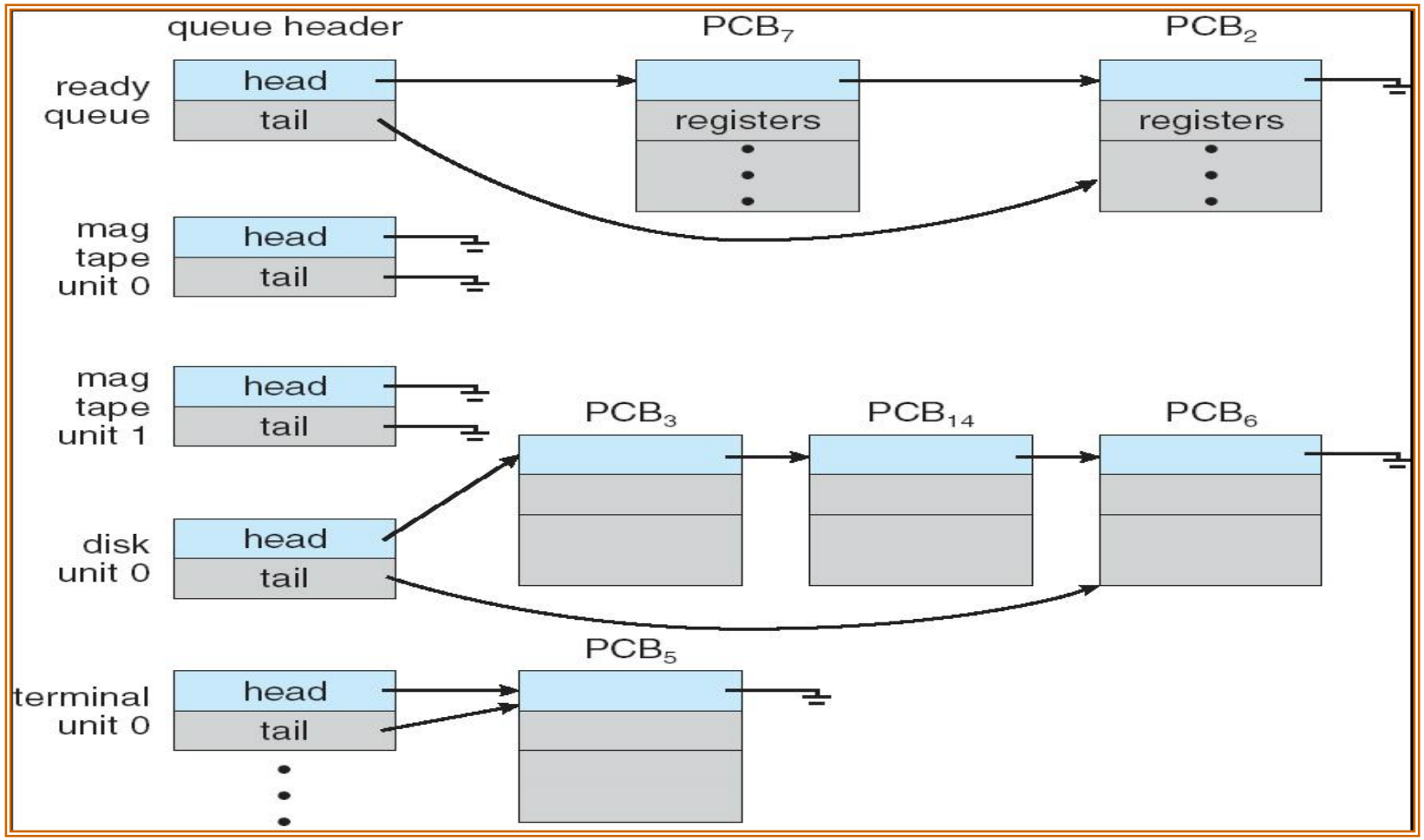
Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.

Process Scheduling Queues



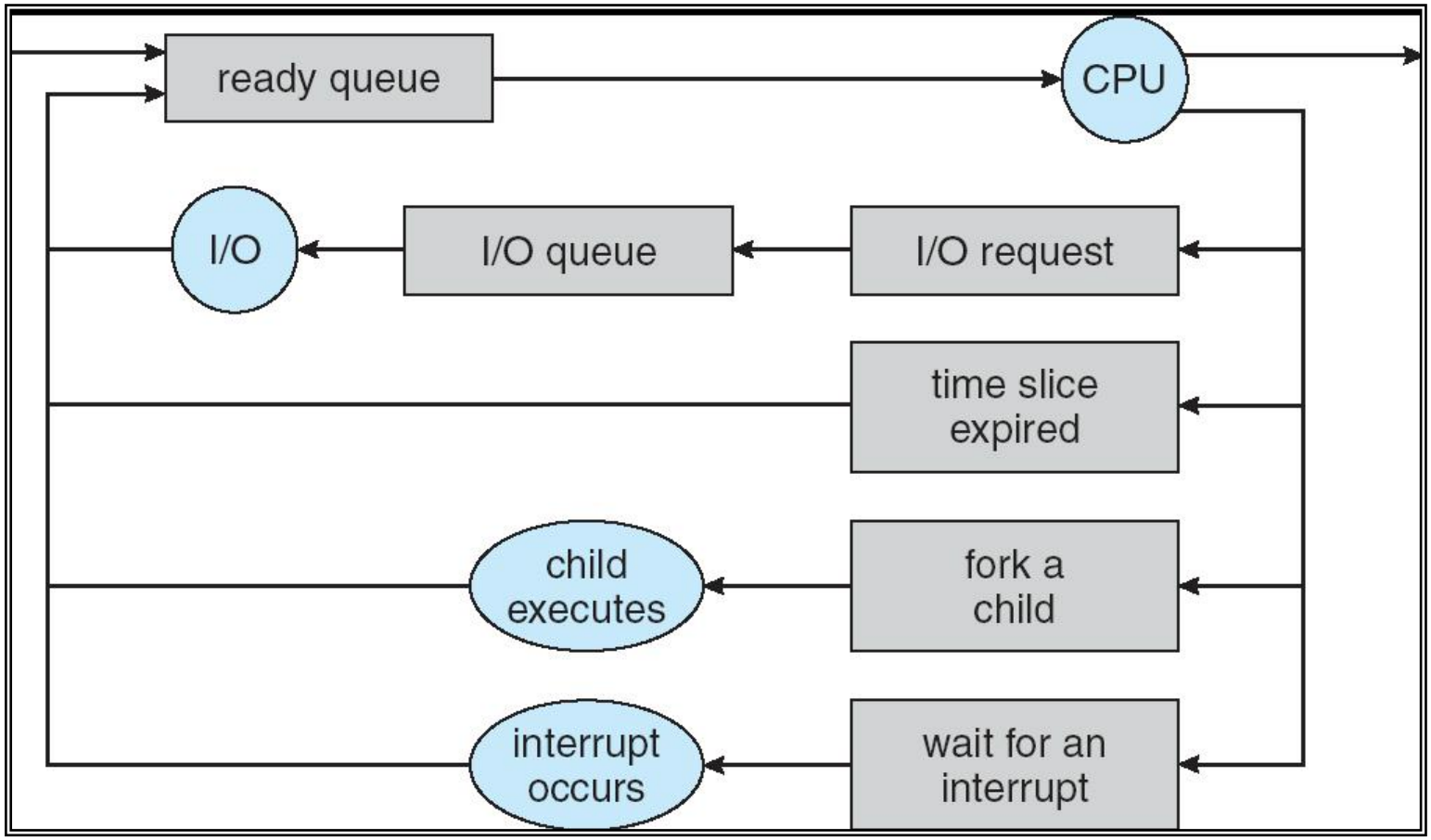
Ready Queue And Various I/O Device Queues

Ready Queue is implemented using Linked List



Representation of Process Scheduling

Queuing Diagram



Schedulers

Schedulers are special system software which handle process scheduling in various ways.

Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Schedulers are of three types −

Long-Term Scheduler

Short-Term Scheduler

Medium-Term Scheduler

Long Term Scheduler / Job Scheduler

It **determines which processes are admitted** to the system for processing.

Selects processes from pool (disk) and bring them into the ready queue

It **selects processes from the queue and loads them into memory** for execution.

When a process changes the state from new to ready, then there is use of long-term scheduler.

It controls **Degree of Multiprogramming (DoM)**

DoM: The degree of multiprogramming describes the maximum number of processes that a single-processor system can accommodate efficiently.

Long Term Scheduler / Job Scheduler

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor (CPU) bound.

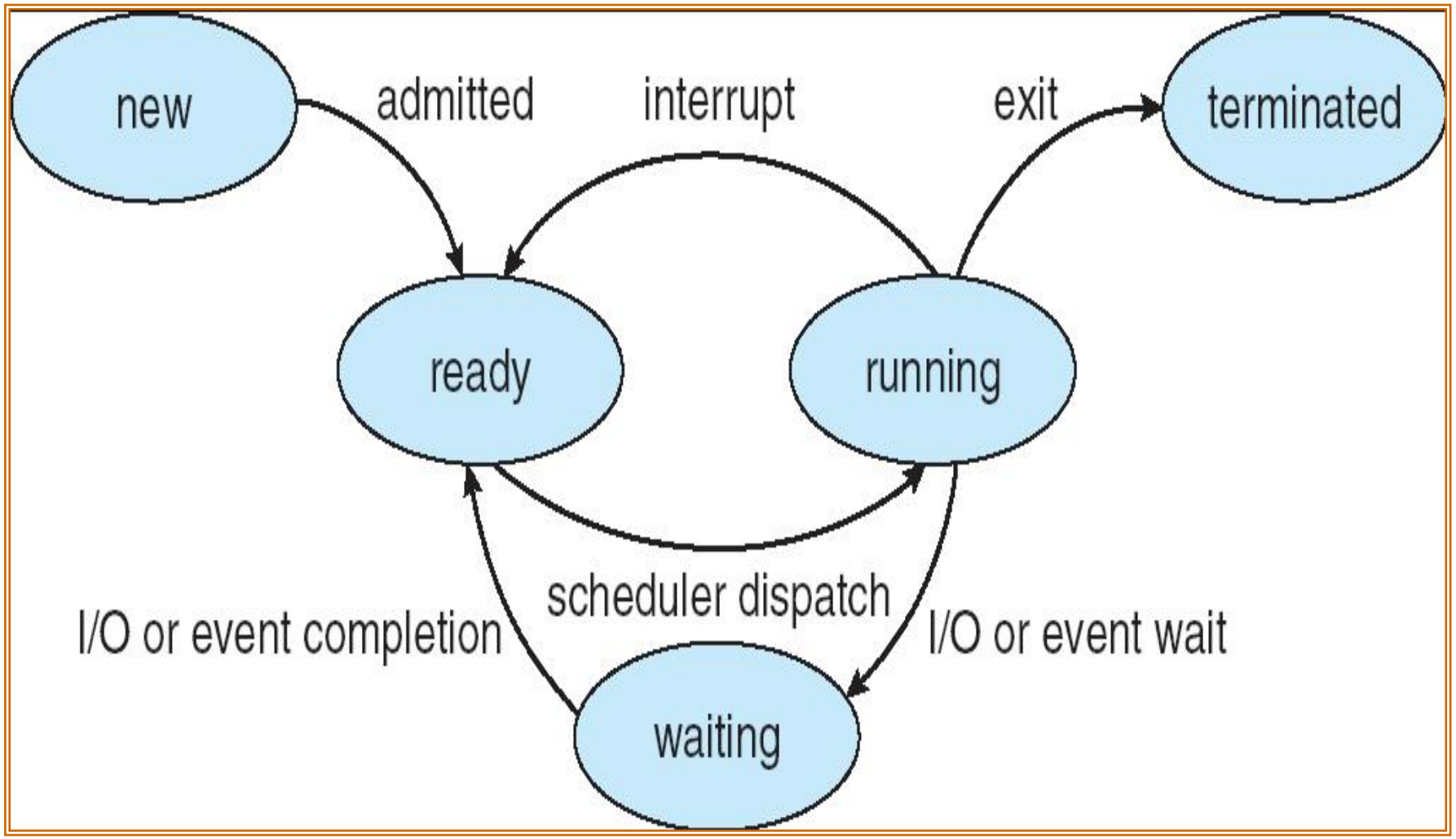
I/O BOUND: that spends its more time in doing I/O than computations.

CPU BOUND: that spends its more time doing computations.

LTS selects a good process.

Good Process: (I/O Bound + CPU Bound)

Process States



Short-Term Scheduler (CPU Scheduler)

Short-term schedulers, also known as dispatchers.

Selects which process should be executed next among the processes and allocates CPU to one of them.

(From Ready Queue Selects one process for execution and Allocate CPU **(Running Queue))**

It must select process for CPU execution frequently.

Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduling

Medium-term scheduling is a part of **swapping**.

Reduces DOM(Degree of Multiprogramming)

Processes are created and stored in Main Memory by Long Term Scheduler, But these processes has to wait for their turn to get execute.

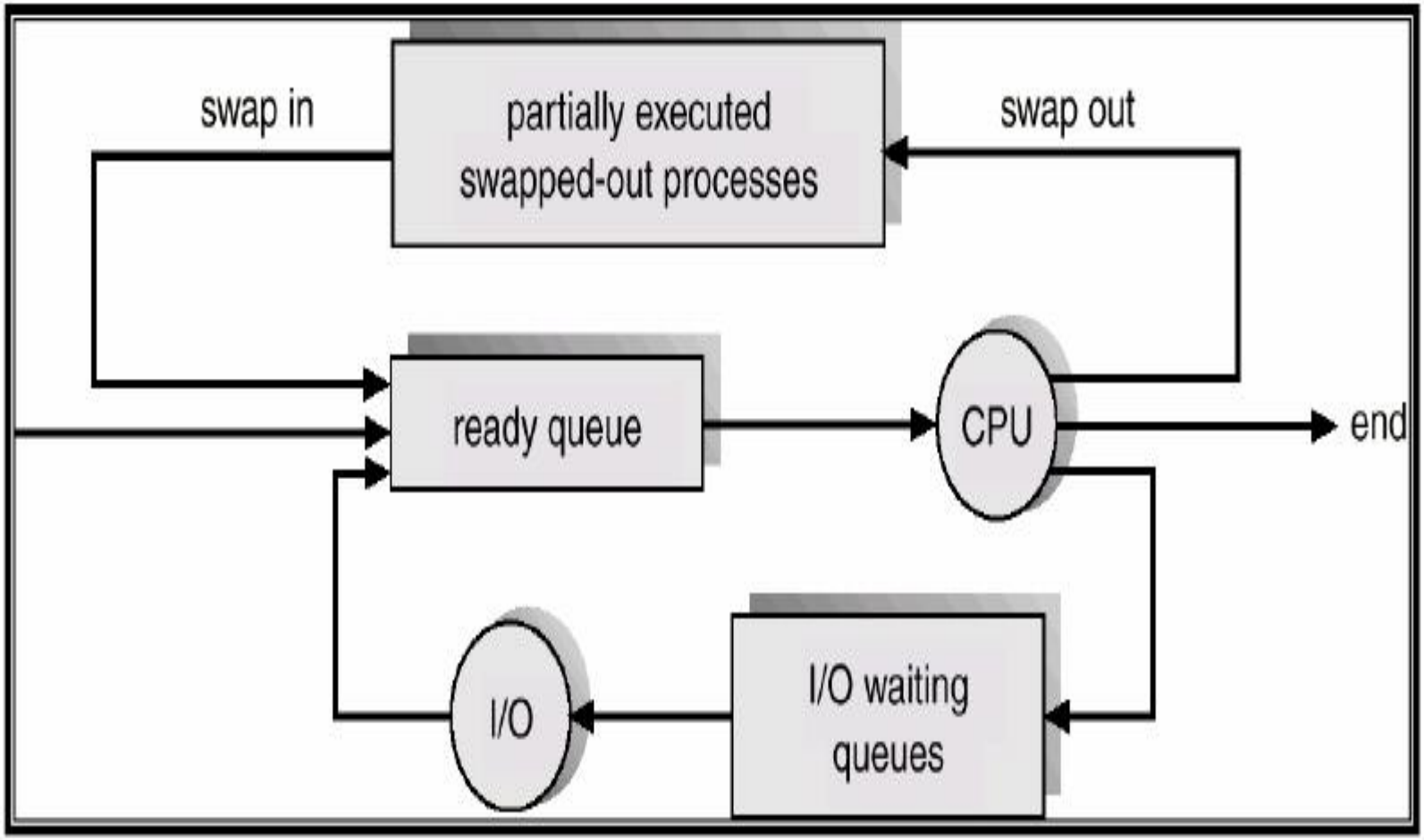
In Main Memory only those processes are kept that require execution.

When ready queue is empty,

MTS Swap-In and Swap-Out the processes from Main Memory and Secondary Memory.

Ready + Running + Waiting Queues are in Main Memory

Medium Term Scheduling



Process Operations

Operations on Process

1. Process Creation

2. Process Termination

Process Creation

Parent process create children processes, which, in turn create other processes, forming a tree of processes

The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.

Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

Process Creation (Cont.)

Resource sharing

- Parent and children **share all resources**
- **Children share subset** of parent's resources
- Parent and child **share no resources**

Execution Sequence

- Parent and children **execute concurrently**
- Parent waits until children terminate

Process Creation (Cont.)

Address Space

Address space may refer to a range of either physical or virtual **addresses** accessible to a processor or reserved for a process.

- Child process is duplicate of parent process (same program and data
- Child process has new program loaded into it

Each process is identified by its process identifier

UNIX examples

- **fork** () system call creates new process
- **exec**() system call used after a **fork** to replace the

fork()

System call **fork()** is used to create processes.

It takes **no arguments** and returns a process ID.

The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller.

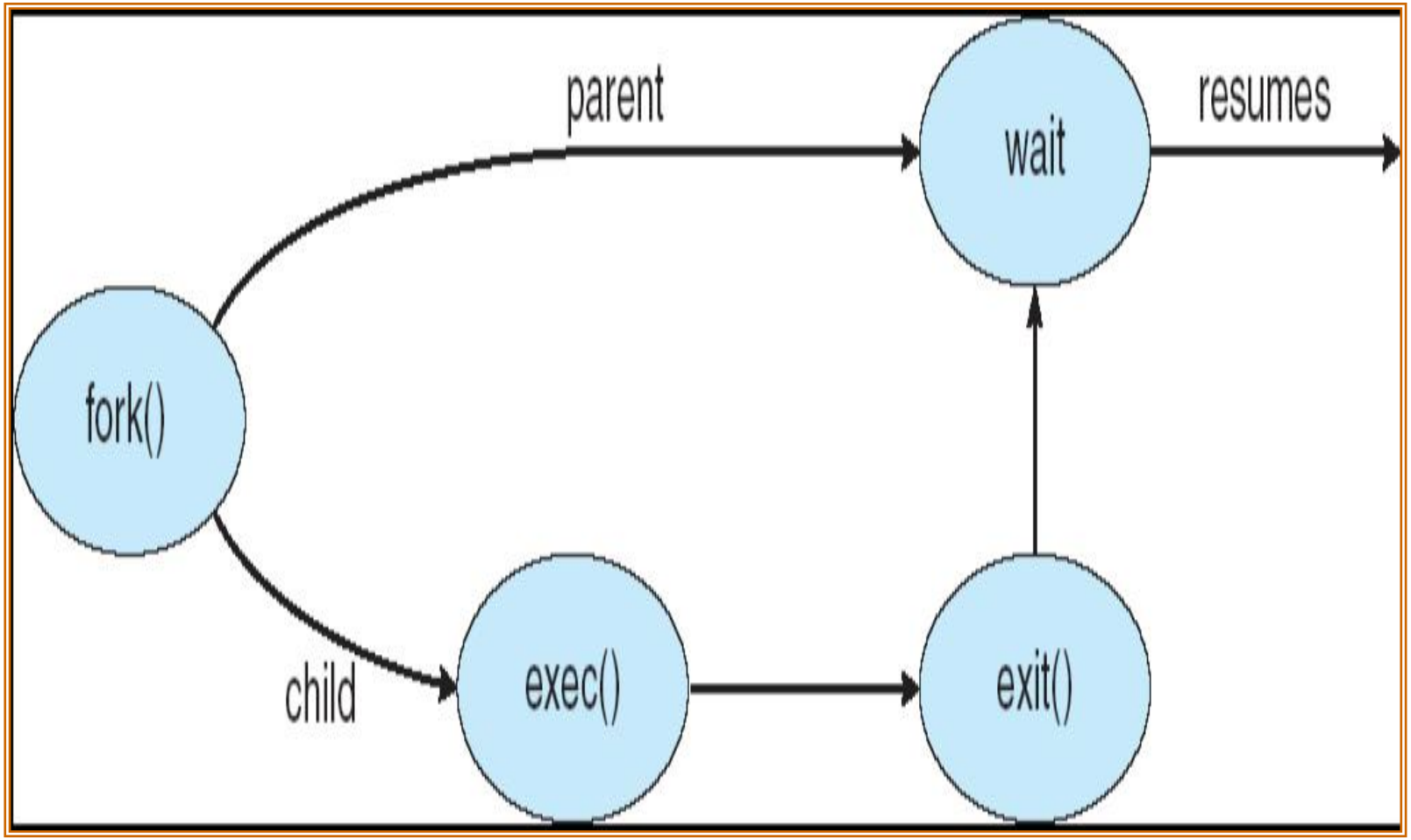
After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call.

This can be done by testing the returned value of **fork()**:

If **fork()** returns a **negative value**, the creation of a child process was **unsuccessful**.

fork() returns a zero to the newly created child process.

fork() returns a positive value, the ***process ID*** of the child process, to the parent.


parent
wait
resumes
fork()
child
exec()
exit()

Process Creation

There are two options for the parent process after creating the child :

1. **Wait for the child process to terminate before proceeding.**

   Parent process makes a wait() system call, for either a specific child process or for any particular child process, which causes the parent process to block until the wait() returns.

2. **Run concurrently with the child, continuing to process without waiting.**

There are also two possibilities **in terms of the address space** of the new process:

The child process is a duplicate of the parent process.

The child process has a program loaded into it.

Process Termination

Process executes last statement and asks the operating system to terminate it (by using **exit() system call**)

Process' resources are de-allocated by operating system

Parent may terminate execution of children processes (**abort**)

Why?

Child has **exceeded allocated resources**

Task assigned to child is **no longer required**

If parent is exiting/Terminated

Some operating system do not allow child to continue if its parent terminates

All children terminated - *cascading termination*

Process Termination

Processes may also be terminated by the system for a variety of **reasons**,

The inability of the system to deliver the necessary system resources.

In response to a KILL command or other unhandled process interrupts.

A parent may kill its children if the task assigned to them is no longer needed.

If the parent does not exit, the system may or may not allow the child to continue without a parent (**orphaned processes** are generally inherited by init, which then proceeds to kill them.)

Process Termination

When a process ends, all of its system resources are freed up. The process **termination status** and execution times are **returned to the parent** if the parent is waiting for the child to terminate,

or eventually returned to init if the process already became an orphan.

The processes which are trying to terminate but cannot do so because their parent is not waiting for them are termed **zombies**. These are eventually **inherited by init process as orphans** and killed off.

INTERPROCESS COMMUNICATION

Process Communicates with each other frequently. The method of communication between two or more processes to exchange the information is called Interprocess-Communication (IPC) and such messages are called as interprocess messages.

IPC allows the process to communicate and to synchronize their actions without sharing the same address space. IPC is very helpful in distributed system. In distributed system actually process resides on different computer connected in a network and the process here will communicate with each other.

For example → chatting on internet through World wide web. (www)

Advantages of IPC →

* Process does not need to guess the size of shared data area required for IPC.
* Process need not define a complex protocal for communication.
* This message resides in buffer of the system till it is delivered so nobody can temper the message.
* The processes may exist in different Computer Systems.
* If data area is smaller than the message then OS gives warning to the system. Operating System blocks a process execution if no message is present.

COOPERATING PROCESSES → The Concurrent processes executing in Content Switching are either independent or cooperative processes. An independent process is one, if it does not affect or be affected by the other processes executing in the system. In other words any process that does not share any data with any other process is independent. While a prosses is said to be Cooperating if it can affect or get affected by other processes executing in the System. Likewise, any process sharing data with other processes is a Cooperating process. An environment provides the process cooperation for the following reasons:

1- Information sharing → It is because several users may be interested in the same piece of information, we should provide an environment to allow concurrent access to these types of resources.

2- Computational speedup → If a particular task is required to run faster, it is broken into Sub-tasks, each of which will be executed in parallel with the others. This can be achieved with multiple processing elements.

3- Modularity → The System can be Constructed into functions into Separate processes or threads. modular fashion, dividing the system

4- Convenience → An individual user may also have many task on which to work at one time.

"Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronize the actions."