

Operating Systems with Linux

(MCA-105)

Unit - 1

by

Dr. Sunil Pratap Singh

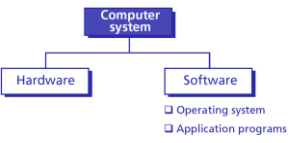
(Associate Professor, BVICAM, New Delhi)

2023

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.1

Computer System

- A computer is a system composed of two major components:

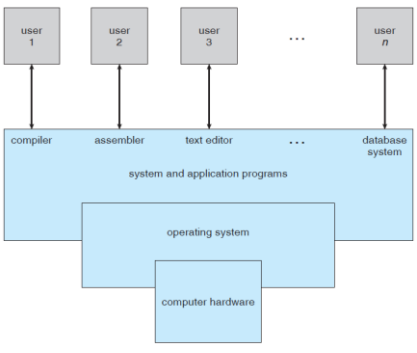


```

graph TD
    CS[Computer system] --> H[Hardware]
    CS --> S[Software]
    S --> OS[Operating system]
    S --> AP[Application programs]
            
```
- Computer hardware is the physical equipment.
- Software is the collection of programs that allows the hardware to do its job.
 - Computer software is divided into two broad categories:
 - Application programs - use the computer hardware to solve users' problems.
 - Operating system - controls the access to hardware by users.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.2


Abstract View of Components of a Computer System



```

graph TD
    U1[user 1] --> SA[system and application programs]
    U2[user 2] --> SA
    U3[user 3] --> SA
    Un[user n] --> SA
    subgraph SA [system and application programs]
        C[compiler]
        A[assembler]
        TE[text editor]
        DB[database system]
    end
    SA --- OS[operating system]
    OS --- CH[computer hardware]
            
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.3



Operating System: Common Definitions

- An operating system is an interface between the hardware of a computer and the user (programs or humans).
- An operating system is a program (or a set of programs) that facilitates the execution of other programs.
- An operating system acts as a general manager supervising the activity of each component in the computer system.
 - As a general manager, the operating system checks that hardware and software resources are used efficiently, and when there is a conflict in using a resource, the operating system mediates to solve it.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.4



Design Goals

- Major design goals of an operating system are:
 - Efficient (efficient use of hardware)
 - Convenient (easy use of resources)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.5



Bootstrap Process

- As per the definitions, the operating system provides support for other programs.
 - It is responsible for loading other programs into memory for execution.
 - However, the operating system itself is a program that needs to be loaded into the memory and be run.

How is this dilemma solved? → Bootstrap Process

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.6

Bootstrap Process (contd...)

1. Bootstrap program runs
2. Operating system is loaded
3. Operating system runs

- A very small section of memory (ROM) holds a small program called the **bootstrap program**.
- When the computer is turned on, the CPU counter is set to the first instruction of this bootstrap program and executes the instructions in this program.
- This program is only responsible for loading the operating system itself, or that part of it required to start up the computer, into RAM memory.
- When loading is done, the program counter in the CPU is set to the first instruction of the operating system in RAM and the operating system is executed.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.7

Operating System: User View

- **Case 1:** When the user sits in front of a PC, consisting of a monitor, keyboard, mouse, and system unit.
 - Such a system is designed for one user to **monopolize its resources**.
 - The goal is to maximize the work (or play) that the user is performing.

In this case, the operating system is designed mostly for **ease of use**, with **some attention paid to performance** and **none paid to resource utilization**.

 - As performance is important to user, such systems are optimized for **single-user experience**.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.8

Operating System: User View (contd...)

- **Case 2:** When the user sits at a terminal connected to a mainframe or a minicomputer.
 - In such system, other users are accessing the same computer through other terminals.
 - These users share resources and may exchange information.

In this case, the operating system is designed to **maximize resource utilization** - to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.9

 **Operating System: User View (contd...)**

- **Case 3: When the users sit at workstations connected to networks of other workstations and servers.**
 - In such system, users have dedicated resources at their disposal, but they also share resources such as networking and servers - file, compute, and print servers.

In this case, the operating system is designed to **compromise** between **individual usability** and **resource utilization**.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.10

 **Operating System: User View (contd...)**

- **Case 4: When the users work with standalone handheld computers, may or may not be connected to networks.**
 - Such systems have limitations of power, speed, and interface, and they perform relatively few remote operations.

In this case, the operating system is designed mostly for **individual usability**, but **performance per unit of battery life is important** as well.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.11

 **Operating System: User View (contd...)**

- **Case 5: When computers have little or no user view.**
 - Embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status.

In this case, the operating system is designed primarily to **run without user intervention**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.12

Operating System: System View

- From the Computer's point of view, the operating system is the program most intimately involved with the hardware.

In this context, an operating system can be viewed as a resource allocator.

- A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on.
- The operating system acts as the manager of these resources.
- Resource allocation is especially important where many users access the same computer.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.13

Operating System: System View (contd...)

- From the Computer's point of view, the operating system controls the various I/O devices and user programs.

In this context, an operating system can be viewed as a control program which manages the execution of user programs to prevent errors and improper use of the computer.

- It is especially concerned with the operation and control of I/O devices.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.14

Kernel

- A Kernel is a computer program that is the heart and core of an Operating System.
- Every operating system - whether it is Windows, Mac, Linux, or Android, has a core program known as Kernel which acts as the 'boss' for the whole system.
- It is the most important part of an Operating System.
- Whenever a system starts, the Kernel is the first program that is loaded after the boot-loader because the Kernel has to handle the rest of the thing of the system for the Operating System.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.15

Kernel (contd...)

- The Kernel remains in the memory until the Operating System is shut-down.
- The Kernel is responsible for disk management, memory management, task management, i/o management, etc.
- When a process makes a request to the Kernel, then it is called **System Call**.
- When we launch a program, the user interface sends a request to Kernel. The Kernel then sends a request to CPU, Memory to assign processing power, memory, and other things so the application can run smoothly in the front end.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.16

Kernel Space and User Space

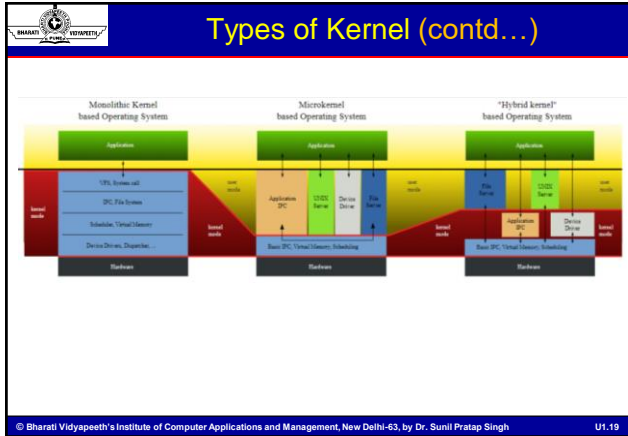
- A Kernel is provided with a protected **Kernel Space** which is a separate area of memory.
 - This area is not accessible by other application programs.
 - The code of the Kernel is loaded into this protected Kernel Space.
- The memory used by other applications is called the **User Space**.
- Communication between Kernel Space and User Space is a bit slower because these are two different spaces in the memory.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.17

Types of Kernel

- **Monolithic Kernel**
 - In monolithic kernel, all the services (**user services** and **kernel services**) of operating system are implemented in the **same memory space**.
- **Microkernel**
 - In Microkernel, the **user services** and **kernel services** are implemented into different spaces, i.e. we use **User Space** and **Kernel Space** in case of Microkernels.
- **Hybrid Kernel**
 - A Hybrid Kernel is a **combination** of both Monolithic Kernel and Microkernel. It makes the use of the speed of Monolithic Kernel and the modularity of Microkernel.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.18



Difference b/w Monolithic and Microkernel


BASIS	MONOLITHIC KERNEL	MICROKERNEL
Basic	Both, user services and kernel services are kept in the same address space.	User services and kernel services are kept in separate address space.
Size	Monolithic kernel is larger than microkernel.	Microkernel are smaller in size.
Execution	Fast execution (through system call)	Slow execution (through message passing)
Extendable	The monolithic kernel is hard to extend.	The microkernel is easily extendable.
Security	If a service crashes, the whole system crashes in monolithic kernel.	If a service crashes, it does effect on working of microkernel.
Example	Linux, Unix, Microsoft Windows (95, 98, Me), DOS, OpenVMS, etc.	Symbian, Singularity, PikeOS, Minix, etc.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.20

Difference b/w Operating System and Kernel

OPERATING SYSTEM	KERNEL
Operating System is a system software.	Kernel is system software which is part of operating system.
Operating System provides interface b/w user and hardware.	Kernel provides interface b/w application and hardware.
It also provides protection and security.	It's main purpose is memory management, disk management, process management and task management.
All system needs operating system to run.	All operating system needs kernel to run.
It is the first program to load when computer boots up.	It is the first program to load when operating system loads.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.21



Components of Operating System

- An operating system has at least four duties:
 - Memory Manager,
 - Process Manager,
 - Device Manager, and
 - Storage Manager.
- Many organizations that have a department that is not necessarily under any specific manager, an operating system also has such a component, which is usually called a **user interface**.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.22



User Interface

- Each operating system has a user interface, a **program** that **accepts requests** from users (processes) and **interprets** them for the rest of the operating system.
- A user interface in some operating systems, such as **UNIX**, is called a **shell**.
- In others, it is called a **window** to denote that it is menu driven and has a **GUI (graphical user interface)** component.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.23



Memory Manager

- Memory allocation must be managed to prevent applications from running out of memory.
- Operating systems can be divided into two broad categories of memory management:
 - Monoprogramming
 - Multiprogramming

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.24

BHARATI VIDYAPEETH UNIVERSITY
Monoprogramming

- In monoprogramming, most of the memory capacity is dedicated to a single program (the data to be processed by a program can be considered as part of the program).
- Only a small part is needed to hold the operating system.

- The job of the memory manager is straightforward here. It loads the program into memory, runs it, and replaces it with the next program.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.25

BHARATI VIDYAPEETH UNIVERSITY
Problems with Monoprogramming

- The program must fit into memory.
 - If the size of memory is less than the size of the program, the program cannot be run.
- When one program is being run, no other program can be executed.
 - A program, during its execution, often needs to receive data from input devices and needs to send data to output devices. Input/output devices are slow compared with the CPU, so when the input/output operations are being carried out, the CPU is idle.
 - It cannot serve another program because this program is not in memory.
 - This is a very inefficient use of memory and CPU time.

Monoprogramming belongs to the past, but it is important to understand multiprogramming.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.26

BHARATI VIDYAPEETH UNIVERSITY
Multiprogramming

- In multiprogramming, more than one program is in memory at the same time, and they are executed concurrently, with the CPU switching rapidly between the programs.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.27

Categories of Multiprogramming

Multiprogramming

Nonswapping

- Partitioning
- Paging

Swapping

- Demand paging
- Demand segmentation

- Nonswapping** - The program remains in memory for the duration of execution.
- Swapping** - During execution, the program can be swapped between memory and disk one or more times.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.28

Responsibilities of Memory Manager

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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.29

Process Manager

- Modern operating systems use three terms that refer to a set of instructions:
 - Program
 - Job
 - Process
- Program:**
 - A program is a non-active set of instructions stored on disk (or tape).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.30

Process Manager: Job

- **Job:**
 - A program becomes a job from the moment it is selected for execution until it has finished running and becomes a program again.
 - During this time a job may or may not be executed.
 - It may be located on disk waiting to be loaded to memory, or it may be loaded into memory and waiting for execution by the CPU.
 - When a job has finished executing (either normally or abnormally), it becomes a program and once again resides on the disk.
 - Every job is a program, but every program is not a job.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.31

Process Manager: Process

- **Process:**
 - A process is a program in execution.
 - It is a program that has started but has not finished.
 - In other words, a process is a job that is being run in memory.
 - It has been selected among other waiting jobs and loaded into memory.
 - A process may be executing or it may be waiting for CPU time.
 - As long as the job is in memory, it is a process.
 - Every process is a job, but every job is not a process.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.32

Relationship between a Program, a Job, and a Process: State Diagram

In the running state, one of three things can happen:

- The process executes until it needs I/O resources.
- The process exhausts its allocated time slot.
- The process terminates.

- In the first case, the process goes into the **waiting state** and waits until I/O is complete.
- In the second case, it goes directly to the **ready state**.
- In the third case, it goes into the **terminated state** and is no longer a process.

```

graph TD
    subgraph Program
        P[Inactive on the disk]
    end
    subgraph Job
        J[Hold]
        T[Terminated]
    end
    subgraph Process
        R[Ready]
        W[Waiting]
        CPU[Gets access to CPU]
    end
    P -- "Becomes a job" --> J
    J -- "Becomes a program" --> P
    J -- "Enters memory" --> R
    R -- "Time slot exhausted" --> W
    W -- "I/O satisfied" --> R
    R -- "CPU" --> CPU
    CPU --> T
    CPU -- "Exits" --> P
    
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.33

Process Manager: Schedulers

- To move a job or process from one state to another, the process manager uses two schedulers:
 - Job Scheduler, and
 - Process Scheduler

```

    graph LR
        subgraph JobScheduler [Job scheduler]
            Hold((Hold))
            Ready((Ready))
            Terminated((Terminated))
            JobScheduler[Job scheduler]
            JobScheduler -- Schedules --> Ready
            Ready --> JobScheduler
            JobScheduler --> Terminated
        end
        subgraph ProcessScheduler [Process Scheduler]
            Ready2((Ready))
            Running((Running))
            Waiting((Waiting))
            ProcessScheduler[Process Scheduler]
            ProcessScheduler -- Schedules --> Running
            Running --> ProcessScheduler
            ProcessScheduler --> Waiting
            Waiting --> ProcessScheduler
        end
        JobScheduler --> Ready2
        Ready2 --> Running
        Running --> Ready2
        Waiting --> Ready2
    
```

- The **job scheduler** moves a job from the hold state to the ready state or from the running state to the terminated state.
- The **process scheduler** moves a process from one state to another.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.34

Process Manager: Queuing

- To handle multiple processes and jobs, the process manager uses queues (waiting lists).
- An operating system can have several queues.

```

    graph LR
        Start((Start)) --> JobQueue[Job queue]
        JobQueue --> ReadyQueue[Ready queue]
        ReadyQueue --> CPU[CPU]
        CPU --> Done((Done))
        CPU -- "To memory" --> ReadyQueue
        CPU -- "Time slot exhausted" --> ReadyQueue
        CPU -- "Needs I/O" --> IOQueue[I/O queue]
        IOQueue -- "I/O satisfied" --> ReadyQueue
    
```

- The process manager can have different policies for selecting the next job or process from a queue: it could be first in, first out (FIFO), shortest length first, highest priority first, and so on.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.35

Process Manager: Process Synchronization

- The main idea behind process management is to **synchronize different processes with different resources**.
- Whenever resources can be used by more than one user (or process, in this case), we can have two problematic situations:
 - Deadlock** - Deadlock occurs when the operating system does not put resource restrictions on processes.
 - Starvation** - Starvation is the opposite of deadlock. It can happen when the operating system puts too many resource restrictions on a process.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.36

Process Manager: Deadlock

- Deadlock occurs if the operating system allows a process to start running without first checking to see if the required resources are ready, and allows a process to hold a resource as long as it wants.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.37

Process Manager: Starvation

a. Process A needs both File1 and File2 b. Process A still needs both File1 and File2


c. Process A still needs both File1 and File2 (starving)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.38

Responsibilities of Process Manager

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.39



Responsibilities of Device Manager (I/O Manager)

- Monitors every input/output device constantly to ensure that the device is functioning properly. The manager also needs to know when a device has finished serving one process and is ready to serve the next process in the queue.
- Maintains a queue for each input/output device or one or more queues for similar input/output devices. For example, if there are two fast printers in the system, the manager can have one queue for each or one queue for both.
- Controls the different policies for accessing input/output devices. For example, it may use FIFO for one device and shortest length first for another.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.40



Storage Manager

- To make the computer system convenient for users, the operating system provides a uniform, logical view of information storage.
- The operating system maps files onto physical media and accesses these files via the storage devices.
- File-System Management
- Mass-Storage (Disk) Management


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.41



File-System Management

- Computers can store information on several different types of physical media.
- Magnetic disk, optical disk, and magnetic tape are the most common.
- Each of these media has its own characteristics and physical organization.
- Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.
- These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.42



File-System Management (contd..)

- A file is a collection of related information defined by its creator.
- Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files are normally organized into directories to make them easier to use.
- When multiple users have access to files, it may be desirable to control by whom and in what ways (for example, read, write, append) files may be accessed.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.43



Responsibilities of File-System Management

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.44



Mass-Storage (Disk) Management

- The computer system must provide secondary storage to back up main memory because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost.
- Modern computer systems use disks as the principal storage medium for both, programs and data.
- Most programs – including compilers, assemblers, word processors, editors, and formatters - are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing.
- Hence, the proper management of disk storage is of central importance to a computer system.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.45

Responsibilities of Mass-Storage Management

- Free-space management
- Storage allocation
- Disk scheduling

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.46

Services of Operating System

- An operating system provides an environment for the execution of programs.
- It provides certain services to programs and to the users of those programs.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.47

Services of Operating System (contd..)

User Interface:

- Almost all operating systems have a User Interface (UI).
- User Interface can take several forms:
 - **Command-line Interface (CLI)** - Uses text commands and a method for entering them.
 - **Batch Interface** - Commands and directives to control those commands are entered into files, and those files are executed.
 - **Graphical User Interface (GUI)** – It is the most commonly used UI. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.48

 **Services of Operating System (contd..)**


Program Execution:

- Load a program into memory and run that program.
- The program must be able to end its execution, either normally or abnormally (indicating error).

I/O Operations:

- The operating system provides a means to do I/O. For efficiency and protection, users usually cannot control I/O devices directly.
- A running program may require I/O, which may involve a file or an I/O device.
- For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.49

 **Services of Operating System (contd..)**


File System:

- The operating system provides services to read and write files and directories.
- Some programs include permissions management to allow or deny access to files or directories based on file ownership.

Communication:

- The operating system provides services for exchanging information between processes.
- Communication may occur **between processes that are executing on the same computer** or **between processes that are executing on different computer systems tied together by a computer network**.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.50

 **Services of Operating System (contd..)**

Resource Allocation:

- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- Many different types of resources are managed by the operating system.
- Some resources (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.51




Services of Operating System (contd..)

Accounting:

- The operating system provides services to keep track of which users use how much and what kinds of computer resources.
- This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics
- Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.52




Services of Operating System (contd..)

Protection and Security:

- Protection involves ensuring that all access to system resources is controlled.
- When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself.
- Security of the system from outsiders is also important.
- Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.
- It extends to defending external I/O devices including modems, network adapters, etc. from invalid access attempts.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.53



Computer-System Architecture

- A computer system may be organized in a number of different ways.
 - We can categorize roughly according to the number of general-purpose processors used.
 - Single-Processor Systems
 - Multiprocessor Systems
 - Clustered Systems


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.54



Single-Processor Systems

- Most systems use a single processor.
- These systems range from PDAs through mainframes.
- **There is one main CPU** capable of executing a general-purpose instruction set, including instructions from user processes.
- **Almost all systems have other special-purpose processors as well.**
 - They may come in the form of device-specific processors such as disk, keyboard, and graphics controllers.
 - The special-purpose processors run a limited instruction set and do not run user processes.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.55




Single-Processor Systems (contd...)

Examples of Special-purpose Processors:

- PCs contain a disk-controller microprocessor which receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm.
 - This arrangement relieves the main CPU of the overhead of disk scheduling.
- PCs contain a keyboard microprocessor to convert the keystrokes into codes to be sent to the CPU.
- **The operating system cannot communicate with these processors; they do their jobs autonomously.**
- **The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.56



Multiprocessor Systems

- Multiprocessor systems have two or more processors in close communication, sharing the computer resources.
- **Main Advantages of Multiprocessor Systems:**
 - **Increased Throughput**
 - By increasing the number of processors, we expect to get more work done in less time.
 - The speed-up ratio with N processors is not N, however; rather, it is less than N.
 - **Economy of Scale**
 - Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies
 - **Increased Reliability**
 - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.57

Types of Multiprocessor Systems

- **Asymmetric Multiprocessing (AMP)**
 - Each processor is assigned a specific task.
 - A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks.
 - This scheme defines a master–slave relationship.
 - The master processor schedules and allocates work to the slave processors.
 - **Example:** Sun’s Operating System (Version 4) – SunOS, IOS
- **Symmetric Multiprocessing (SMP)**
 - Each processor performs all tasks within the operating system.
 - All processors are peers; no master–slave relationship exists between processors.
 - Each processor has its own set of registers, as well as a cache; however, all processors share physical memory.
 - **Examples:** Sun’s Operating System (Version 5) – Solaris, Windows, Unix, Linux

© Bharati Vidyapeeth’s Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.58

Windows: Symmetric Multiprocessing

The screenshot shows a Microsoft Docs page titled "Multiprocessor-Safe". The page content includes:

- The Microsoft Windows NT-based operating system is designed to run uniformly on uniprocessor and symmetric multiprocessor (SMP) platforms, and kernel-mode drivers should be designed to do likewise.
- On any multiprocessor platform, the following conditions exist:
 - All CPUs are identical, and either all or none of the processors must have identical capabilities.
 - All CPUs share memory and have uniform access to memory.
 - In a symmetric platform, every CPU can access memory, take an interrupt, and access local resources. (By contrast, in an asymmetric multiprocessor machine, one CPU takes all interrupts for a set of subordinate CPUs.)

© Bharati Vidyapeeth’s Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.59


Multiprocessor Systems (contd...)

- A recent trend in CPU design is to include multiple computing cores on a single chip.

Figure: A dual-core design with two cores placed on the same chip.

- In essence, these are multiprocessor chips.
- They can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication.
- In addition, one chip with multiple cores uses significantly less power than multiple single-core chips.
- Multicore systems are especially well suited for server systems such as database and Web servers.


© Bharati Vidyapeeth’s Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.60



Operating-System Operations

- The operating system's nature is **interrupt driven**.
- **Interrupts** are the signals that are triggered by the software or the hardware when any process or an event needs immediate attention.
 - If we move our mouse on the screen, an interrupt is getting generated.
 - If we press any key on a keyboard, an interrupt is getting generated.
 - If we print a document, an interrupt is getting generated.
 - If we power off the monitor screen, an interrupt is getting generated.
 - There are many such occasion such that operating system is waiting for some interrupts to take place and for attending the interrupts.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.61



Operating-System Operations (contd..)

- **Hardware** may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.
- **Software** may trigger an interrupt by executing a special operation called a system call.
- **Interrupts, when generated by the hardware or the software have to be attended by the operating system.**
- The operating system has some processes that are operating and these processes are present in the queue, so **when the operating system receives the interrupt signals, all the processes in the queue that were running are made to halt and CPU is made to attend to the signal of the interrupt.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.62



Operating-System Operations (contd..)


- A **Trap** is a software-generated interrupt which can be caused by some reason such as:
 - Error in the instruction (for example, division by zero or invalid memory access).
 - User program initiates a specific service request from the operating system (for example, the user program requires printing something to the screen; it would invoke a trap and the operating system will perform writing that data to the screen).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.63

 **Operating-System Operations (contd..)**


- The **system calls** are example of a **trap** where the operating system is asked by the program to request a particular service and the operating system then generates an interrupt to access the services for the program.
 - Example: The statement `printf("%s\n", str);` will invoke the write function to print the output to the standard output which is the monitor. This will invoke a trap and it will pass the control to the trap handler. Then, the user mode changes to kernel mode and the OS executes the write call. After completing the task, the control is transferred back to the user mode from the kernel mode.
- The traps are more likely to be caused by the execution of the current instructions and therefore **the traps are also known as synchronous events**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.64

 **Operating-System Operations (contd..)**

- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
 - The user program that is up for execution on CPU generally uses library calls to issue system calls. The function of the library routine is to check the arguments that are provided by the application and then build a data structure to convey the arguments from the application to the kernel of the operating system and then execute special instructions which are called the trap or the software interrupts.
 - These special instructions or traps contain operands that help to identify exactly which kernel service the application arguments are demanding. So, when the process is made to execute the traps then the interrupt saves the state of the user code and then switches to the supervisor mode and then dispatches the right kernel routine that can implement the requested service.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.65

 **OS Operations: Dual Mode Operation**

- To ensure the proper execution of the operating system, it is important to distinguish between the execution of operating-system code and user-defined code.
- We need two separate modes of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).
- Most computer systems provide hardware support that allows us to differentiate among various modes of execution.
- A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: **kernel (0)** or **user (1)**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.66

OS Operations: Dual Mode Operation (contd..)

- With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.
- When the computer system is executing on behalf of a user application, the system is in user mode.
- When a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.

The diagram shows a horizontal line separating the 'user process' (top) from the 'kernel' (bottom). In the user process, a box labeled 'user process executing' has an arrow pointing to 'calls system call'. This arrow crosses the line and is labeled 'trap mode bit = 0'. In the kernel, a box labeled 'execute system call' has an arrow pointing back up to 'return from system call'. This arrow crosses the line and is labeled 'return mode bit = 1'. On the right side of the diagram, 'user mode (mode bit = 1)' is indicated for the top section and 'kernel mode (mode bit = 0)' for the bottom section.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.67

OS Operations: Dual Mode Operation (contd..)


- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0).
- The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.
- The dual mode of operation provides with the means for protecting the operating system from errant users.
- This protection is achieved by designating some of the machine instructions that may cause harm as privileged instructions.
- The hardware allows privileged instructions to be executed only in kernel mode.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.68

OS Operations: Dual Mode Operation (contd..)

- Recent versions of the Intel CPU do provide dual-mode operation.
- Most contemporary operating systems - such as Windows, as well as Unix, Linux, and Solaris - take advantage of dual-mode feature and provide greater protection for the operating system.
- The lack of a hardware-supported dual mode can cause serious shortcomings in an operating system.
 - For instance, MS-DOS was written for the Intel 8088 architecture, which has no mode bit and therefore no dual mode.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.69



OS Operations: Timer

- The operating system should maintains control over the CPU.
- It can not be allowed that a user program gets stuck in an infinite loop or to fail to call system services and never return control to the operating system.
- **A timer can be set to interrupt the computer after a specified period.**
- A timer is generally implemented by a fixed-rate clock and a counter.
- The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.
- **Before turning over control to the user, the operating system ensures that the timer is set to interrupt.**
 - **If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.70



Evolution of Operating Systems: Serial Processing

- In earliest computers (from the late-1940s to the mid-1950s), the programmer interacted directly with the computer hardware.
 - There was no Operating System.
 - The computers were run from a console consisting of **display lights**, **toggle switches**, some form of **input device**, and a **printer**.
 - Programs in machine code were loaded via the input device (e.g., a card reader).
 - If an error halted the program, the error condition was indicated by the lights.
 - If the program proceeded to a normal completion, the output appeared on the printer.
 - These early systems presented two main problems:
 - **Scheduling**
 - **Setup Time**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.71



Serial Processing (contd...)

- **Scheduling**
 - Most installations used a hardcopy sign-up sheet to reserve computer time. A user could sign up for a block of time in multiples of a half hour or so. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.
- **Setup Time**
 - A considerable amount of time was spent just in setting up the program to run. A single program, called a job, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions.
 - **This mode of operation is termed Serial Processing, reflecting the fact that users have access to the computer in series.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.72

Simple Batch Systems

- To **improve utilization**, the concept of a **batch operating system** was developed.
- The first batch operating system (and the first OS of any kind) was developed in the mid-1950s by General Motors for use on an IBM 701.
- The main idea behind the simple batch-processing scheme was the use of a **software**, known as the **monitor**.
- With this type of OS, the user no longer has direct access to the processor.
 - The user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
 - Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.73

Simple Batch Systems (contd...)

- The monitor reads in jobs one at a time from the input device.
- As it is read in, the current job is placed in the user program area, and control is passed to this job.
- When the job is completed, it returns control to the monitor, which immediately reads in the next job.
- The results of each job are sent to an output device (such as a printer).

```

            graph TD
                subgraph Monitor
                    IP[Interrupt processing]
                    DD[Device drivers]
                    JS[Job sequencing]
                    CLIC[Control language interpreter]
                end
                subgraph UserProgramArea [User program area]
                    UPA[ ]
                end
                Monitor --- Boundary
                Boundary --- UPA
            
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.74

Simple Batch Systems (contd...)

- The monitor performs a scheduling function.
 - A batch of jobs is queued up, and jobs are executed as rapidly as possible, with no intervening idle time.
- The monitor improves job setup time as well.
 - With each job, instructions are included in a primitive form of job control language (JCL).
 - JCL is a special type of programming language used to provide instructions to the monitor.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.75

Simple Batch Systems (contd...)

- With a batch operating system, processor time alternates between execution of user programs and execution of the monitor.
- There have been two sacrifices:
 - Some main memory is given over to the monitor.
 - Some processor time is consumed by the monitor.
 - Both of these are forms of overhead.
 - Despite this overhead, the simple batch system improves utilization of the computer.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.76

Multiprogrammed Batch Systems

- Even with the automatic job sequencing provided by a simple batch operating system, the processor is often idle.
- The problem is that I/O devices are slow compared to the processor.

Read one record from file	15 μ s
Execute 100 instructions	1 μ s
Write one record to file	15 μ s
Total	31 μ s

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

System Utilization Example

- In this example, the computer spends over 96% of its time waiting for I/O devices to finish transferring data to and from the file.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.77

Multiprogrammed Batch Systems (contd...)

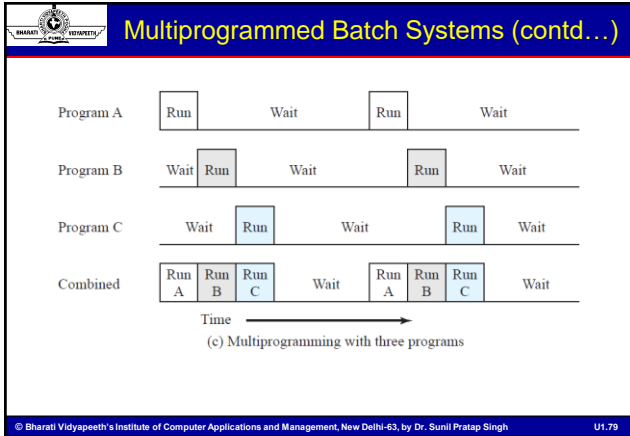
Time →

(a) Uniprogramming

Time →


(b) Multiprogramming with two programs

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.78



-
- In multiprogrammed systems, several jobs are kept in main memory and they are executed concurrently, with the CPU switching rapidly between the jobs.
 - Multiprogramming is the central theme of modern operating systems.
 - When several jobs are kept in main memory, it requires some form of memory management.
 - In addition, if several jobs are ready to run, the processor must decide which one to run, this decision requires an algorithm for scheduling.
- © Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.80


-
- The most notable additional feature that is useful for multiprogramming is the hardware that supports I/O interrupts and DMA (direct memory access).
 - With interrupt-driven I/O or DMA, the processor can issue an I/O command for one job and proceed with the execution of another job while the I/O is carried out by the device controller.
 - When the I/O operation is complete, the processor is interrupted and control is passed to an interrupt-handling program in the OS.
 - The OS will then pass control to another job.
- © Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.81



Programmed I/O

- The problem with the **Programmed I/O** is that **the processor has to wait** a long time for the input/output module of concern to be ready for either reception or transmission of more data.
 - The processor, while waiting, must repeatedly interrogate the status of the I/O module. **As a result the level of performance of entire system is degraded.**
- An alternative approach for this is **Interrupt-driven I/O**.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.82



Interrupt-driven I/O

- **Interrupt driven I/O** means CPU **issues commands to the I/O module** then **proceeds with its normal work until interrupted by I/O device** on completion of its work.
 - The I/O module will then interrupt the processor to request service, when it is ready to exchange data with the processor.
 - The processor then executes the data transfer as before and then resumes its former processing.
- **Interrupt-driven I/O still consumes a lot of time because every data has to pass with processor.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.83



Limitations of Programmed and Interrupt-driven I/O

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.84



Direct Memory Access (DMA)

- When large volumes of data are to be moved, a more efficient technique is required: **Direct Memory Access**.
- In DMA mechanism, **CPU grants I/O module authority to read from or write to memory without involvement**.
 - The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an I/O module.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.85



Working of DMA

- When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending the following information:
 - Whether a read or write is requested.
 - The address of the I/O devices.
 - Starting location in memory to read from or write to.
 - The number of words to be read or written.
- The processor then continues with other work. **It has delegated this I/O operation to the DMA module, and that module will take care of it.**
 - The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor.
 - When the transfer is complete, the DMA module sends an interrupt signal to the processor. **Thus the processor is involved only at the beginning and at the end of the transfer.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.86



Time-Sharing Systems

- With the use of multiprogramming, batch processing can be quite efficient.
- However, for many jobs, **it is desirable to provide a mode in which the user interacts directly with the computer.**
- Indeed, for some jobs, such as **transaction processing, an interactive mode is essential.**
- Today, the requirement for an interactive computing facility can be met by the use of a **dedicated personal computer or workstation.**
- **The interactive computing facility was not available in the 1960s, when most computers were big and costly.**
- **Instead, time sharing was developed.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.87



Time-Sharing Systems (contd...)

- When the concept of multiprogramming is used to handle **multiple interactive jobs**, it is referred to as time sharing.
 - Processor time is shared among multiple users.
 - Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short quantum of computation.
 - The main objective of time-sharing systems is **minimize response time**.
 - **Compatible Time-Sharing System (CTSS)** was one of the first time-sharing operating systems, developed at MIT in 1961 for the IBM 709.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.88



Compatible Time-Sharing System (CTSS)

- The system ran on a computer with 32,000 words of main memory, with the monitor consuming 5000 of that.
- A system clock generated interrupts at a rate of approximately one every 0.2 seconds.
- At each clock interrupt, the OS regained control and could assign the processor to another user.
- **This technique is known as time slicing.**
- Thus, at regular time intervals, the current user would be preempted and another user loaded in.
- To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in.
- Subsequently, the old user program code and data were restored in main memory when that program was next given a turn.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.89



Real-Time Embedded Systems

- Embedded computers are found everywhere - from car engines and manufacturing robots to microwave ovens.
- Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.
- **These embedded systems vary considerably.**
 - Some are general-purpose computers, running standard operating systems—such as UNIX—with special-purpose applications to implement the functionality.
 - Others are hardware devices with a special-purpose embedded operating system providing just the functionality desired.
 - Yet others are hardware devices with application-specific integrated circuits (ASICs) that perform their tasks without an operating system.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.90



Real-Time Embedded Systems (contd...)

- Embedded systems almost always run real-time operating systems.
- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application.
- Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.91



Real-Time Embedded Systems (contd...)

- A real-time system has well-defined, fixed time constraints.
 - Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building.
- A real-time system functions correctly only if it returns the correct result within its time constraints.
- Contrast this system with a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or a batch system, which may have no time constraints at all.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.92



Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide the users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Distributed systems depend on networking for their functionality.
- Networks are characterized based on the distances between their nodes.
 - SAN (Small-Area Network)
 - LAN (Local-Area Network)
 - MAN (Metropolitan-Area Network)
 - WAN (Wide-Area Network)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.93

Distributed Operating Systems (contd...)

- Various inter-connected computers communicate each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
 - These are referred as **loosely coupled systems** or distributed systems.
 - The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.
 - The different systems communicate closely enough to provide the illusion that only a single operating system controls the network.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.94

Computing Environments

- Client-Server Computing
- Peer-to-Peer Computing

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.95

Client-Server Computing

- It is a form of specialized distributed system.

```

    graph TD
      C1[client] --- Network[ ]
      C2[client] --- Network
      C3[client] --- Network
      Dots[...] --- Network
      C4[client] --- Network
      Network --- S[server]
      style Network width:0px,height:0px
    
```

- Many of today's systems act as **server systems** to satisfy requests generated by client systems.
- Server systems can be broadly categorized as:
 - Compute Servers, and
 - File Servers

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.96

Compute-Server System

- It provides an interface to which a client can send a request to perform an action (for example, read data, encrypt the plain text, etc.).
 - In response, the server executes the action and sends back results to the client.
 - A server running a database (program) that responds to client requests for accessing data is an example of such a system.
 - A server running a container (program) that responds to client requests for performing computation (business logic) is also an example of compute-server system.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.97

File-Server System

- It provides a file-system interface where clients can create, update, read, and delete files.
- File servers may be categorized by the method of access:
 - Internet File Servers
 - An example of such a system is a Web server that delivers files to clients running Web browsers.
 - Internet file servers are frequently accessed by FTP or by HTTP.
 - LAN Servers
 - Any computer can be configured to be a host and act as a file server.
 - A file server might be a dedicated network-attached storage (NAS) device that also serves as a remote hard disk drive for other computers.
 - LAN servers are accessed by SMB (Server Message Block – used in Windows and Mac) and NFS (Network File System – used in Unix-like Systems) protocol.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.98

Peer-to-Peer (P2P) Computing

- It is another structure for a distributed system.
- In this model, clients and servers are not distinguished from one another.
- All nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.
- In P2P system, services can be provided by several nodes distributed throughout the network.
- To participate in a P2P system, a node must first join the network of peers.
- Once a node has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.99

Working of Nodes in P2P Computing Environment

- Approach - 1
- When a node joins a network, it registers its service with a centralized lookup service on the network.
- Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service.
- The remainder of the communication takes place between the client and the service provider.
- Approach - 2
- A peer acting as a client first discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network.
- The node (or nodes) providing that service responds to the peer making the request.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.100

System Call

- The system call is the means by which a process requests a specific kernel service.
- System calls provide an interface to the services made available by an operating system.
- System calls are generally available as routines written in C and C++.
- Certain low-level tasks (for example, tasks where hardware must be accessed directly) may need to be written using assembly-language instructions.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.101

How System Calls are Used? - Example

A simple program to read data from one file and copy them to another file.

source file → destination 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

In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define the two files.

Once the two file names are obtained, the program must open the input file and create the output file. Each of these operations requires another system call.

There are also possible error conditions for each operation. When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another sequence of system calls) and then terminate abnormally (another system call).

If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (another system call).

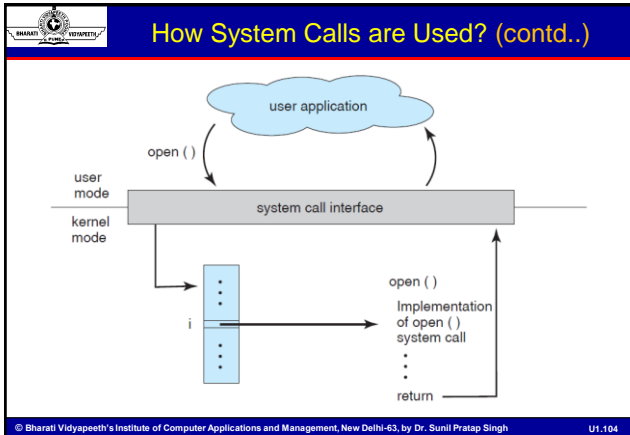
Frequently, systems execute thousands of system calls per second.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.102

How System Calls are Used? (contd..)

- Most programmers never focus on detailed working levels of System Calls.
 - Typically, application developers design the programs according to an Application Programming Interface (API).
 - The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
 - Three of the most common APIs available to application programmers are the Win32 API for Windows systems, the POSIX (Portable Operating System Interface) API for POSIX-based systems (all versions of UNIX, Linux, and MacOS X), and the Java API for designing programs that run on the Java virtual machine.
- Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer.
- For example, the Win32 function CreateProcess() actually calls the NtCreateProcess() system call in the Windows kernel.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.103



How System Calls are Used? (contd..)

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:

```

#include <stdio.h>
int main()
{
    ...
    printf("Greetings");
    return 0;
}
    
```

The diagram shows a box for 'user mode' containing the C code snippet. An arrow labeled 'printf()' points from the code to a box labeled 'standard C library'. Below the library, a box labeled 'write()' is connected to a cloud labeled 'write() system call'. An arrow labeled 'write()' points from the library to the system call. From the system call, an arrow labeled 'return' points back to the library. From the library, an arrow labeled 'return 0;' points back to the user mode.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.105

Why Programmers prefer API rather than System Calls

- **Program Portability**
 - An application programmer designing a program using an API can expect his program to compile and run on any system that supports the same API.
- **Difficult to Work with System Calls**
 - Actual system calls can often be more detailed and difficult to work with than the API available to an application programmer.
- **System Call Interface in Programming Languages**
 - The support system (a set of functions built into libraries included with a compiler) for most programming languages provides a system-call interface that serves as the link to system calls made available by the operating system.
 - The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.106

Types of System Calls

- **Process Control**
- **File Management**
- **Device Management**
- **Information Maintenance**
- **Communications**
- **Protection**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.107

Types of System Calls

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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.108

Process

- A process is a program in execution.
- A process is the unit of work in systems.
- Systems consist of a collection of processes:
 - **Operating-system processes** execute system code, and
 - **User processes** execute user code
 - All these processes may execute concurrently.
- A process need certain resources - such as CPU time, memory, files, and I/O devices - to accomplish its task.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.109

Process (contd...)

- Modern operating systems support processes that have multiple threads.
- The operating system is responsible for the following activities in connection with process and thread management:
 - The **creation** and **deletion** of both user and system processes;
 - The **scheduling** of processes; and
 - The provision of mechanisms for **synchronization**, **communication**, and **deadlock handling** for processes.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.110

Process Concept

- What to call all the CPU activities?
 - A batch system executes **jobs**;
 - Time-shared system has **user programs**, or **tasks**.
 - The terms **job** and **process** are used almost interchangeably.
- Much of operating-system theory and terminology was developed during a time when the major activity of operating systems was **job processing**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.111

Process in Memory

- Informally, a process is a program in execution.
- However, a process is more than the program code. It includes:
 - Text Section (Program Code)
 - Stack Section (Temporary Data such as Function Parameters, Return Addresses, and Local Variables)
 - Data Section (Global Variables)
 - Heap (Memory that is Dynamically Allocated during Process Run Time)
- Process also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.

The diagram shows a vertical stack of memory sections. From top to bottom: 'stack' (grey), 'heap' (grey), 'data' (grey), and 'text' (grey). A downward arrow is in the stack section, and an upward arrow is in the heap section. The top is labeled 'max' and the bottom is labeled '0'.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.112

Process vs. Program

- A program by itself is not a process.
- A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file)
- A process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.
- Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog.exe or a.out.)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.113

Process Concept (contd...)

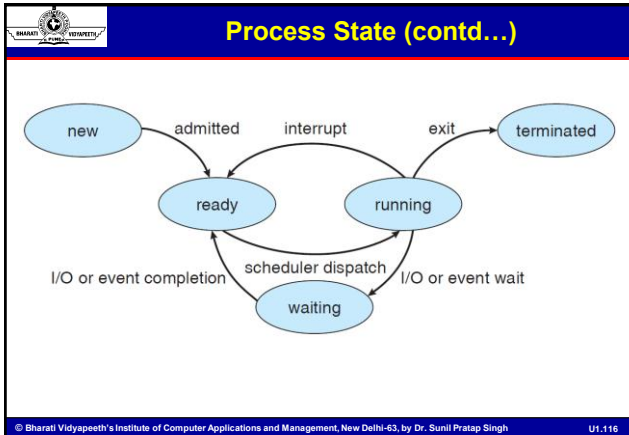
- Although two processes may be associated with the same program, they are considered two separate execution sequences.
 - The same user may invoke many copies of the Web Browser program.
 - Each of these is a separate process.
 - Although the text sections are equivalent, the data, heap, and stack sections vary.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.114

Process State

- As a process executes, it changes state.
- Each process may be in one of the following states:
 - New** - The process is being created.
 - Running** - Instructions are being executed.
 - Waiting** - The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
 - Ready** - The process is waiting to be assigned to a processor.
 - Terminated** - The process has finished execution.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.115



Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB) – A Data Structure**.
- PCB contains many information, including:**
 - Process State** (The state may be new, ready, running, etc.)
 - Program Counter** (The counter indicates the address of the next instruction to be executed for this process.)
 - CPU Registers** (The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, etc.)
 - CPU-scheduling Information**
 - Accounting Information** (This information includes amount of CPU and real time used, job/process numbers, etc.)
 - I/O Status Information** (This information includes the list of I/O devices allocated to the process, a list of open files, etc.)

process state
process number
program counter
registers
memory limits
list of open files
• • •

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.117

Process Representation

- The process control block in the Linux operating system is represented by the C structure `task_struct`.
- All active processes are represented using a doubly linked list of `task_struct`, and the kernel maintains a pointer - `current` - to the process currently executing on the system.

The diagram illustrates a doubly linked list of `task_struct` process information. Three boxes, each labeled 'struct task_struct process information', are connected by curved arrows pointing in both directions. A pointer labeled 'current (currently executing process)' points to the middle box.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.118

Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for execution on the CPU.
- For a single-processor system, there will never be more than one running process.
- If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.119

CPU Switching from Process to Process

The diagram shows the interaction between process P_0 , the operating system, and process P_1 . P_0 is initially executing. An 'interrupt or system call' occurs, leading to 'save state into PCB₀'. The OS then 'reload state from PCB₁' and starts executing P_1 . P_0 becomes idle. Later, another 'interrupt or system call' occurs, leading to 'save state into PCB₁'. The OS then 'reload state from PCB₀' and resumes executing P_0 . P_1 becomes idle.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.120

Scheduling Queues

- As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
 - This queue is generally stored as a linked list.
 - A ready-queue header contains pointers to the first and final PCBs in the list.
 - Each PCB includes a pointer field that points to the next PCB in the ready queue.
- The process may have to wait for shared device, such as a disk. The list of processes waiting for a particular I/O device is called a **device queue**.
 - Each device has its own device queue.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.121

Ready Queue and various I/O Device Queues

The diagram illustrates the structure of scheduling queues. At the top, a 'ready queue' header with 'head' and 'tail' pointers is shown. It points to a linked list of PCBs (Process Control Blocks), specifically PCB₇ and PCB₂. Each PCB contains 'registers' and a pointer to the next PCB. Below this, several I/O device queues are shown: 'mag tape unit 0', 'mag tape unit 1', 'disk unit 0', and 'terminal unit 0'. Each device queue has its own 'head' and 'tail' pointers and points to a specific PCB (PCB₃, PCB₁₄, PCB₆, and PCB₅ respectively).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.122

Process Scheduling: Queuing Diagram

- Two types of queues are present: the **ready queue** and a set of **device queues**.
- The circles represent the resources that serve the queues.
- The arrows indicate the flow of processes in the system.

The diagram shows the flow of processes through different states and resources. A 'ready queue' leads to the 'CPU'. From the 'CPU', processes can move to an 'I/O request' state, which then enters an 'I/O queue' and finally to an 'I/O' resource (circle). From the 'I/O' resource, processes can move to 'child executes' or 'interrupt occurs' states. From 'child executes', processes can move to 'wait for an interrupt' and then to 'interrupt occurs'. From 'interrupt occurs', processes can move back to the 'ready queue'. From 'wait for an interrupt', processes can move to 'time slice expired' and then back to the 'ready queue'. From 'time slice expired', processes can move to 'fork a child' and then to 'child executes'.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.123

Schedulers

- A process migrates among the various scheduling queues throughout its lifetime.
- The operating system selects processes from these queues with appropriate scheduler.
- Often, in a **batch system**, more processes are submitted than can be executed immediately.
 - These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution.
 - The **long-term scheduler**, or **job scheduler**, selects processes from this pool and loads them into memory for execution.
 - The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute and allocates the CPU to one of them.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.124

Short-term Scheduler

- The short-term scheduler must select a new process for the CPU frequently.
- Often, the short-term scheduler executes at least once every 100 milliseconds.
- Because of the short time between executions, the short-term scheduler must be fast.
 - If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then $10/(100 + 10) = 9$ percent of the CPU is being used (wasted) simply for scheduling the work.
 - Short term scheduler controls **multitasking**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.125

Long-term Scheduler

- The long-term scheduler executes much less frequently.
 - Minutes may separate the creation of one new process and the next.
- The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).
 - If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
 - Thus, the long-term scheduler may need to be invoked only when a process leaves the system.
 - Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.126

Long-term Scheduler (contd...)

- Most processes can be described as either **I/O bound** or **CPU bound**.
 - An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
 - A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.
 - If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.
 - If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
 - The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.127

Medium-term Scheduler

- Some **time-sharing systems** may introduce an additional scheduler (**medium-term scheduler**).
- Sometimes it can be advantageous to **remove processes from memory**, and thus **reduce the degree of multiprogramming**.
- Later, the **process can be reintroduced into memory**, and its execution can be continued where it left off.
- **This scheme is called swapping.**
- The process is swapped out, and is later swapped in, by the medium-term scheduler.
- Swapping may be necessary to improve the **process mix** or because a change in memory requirements has **overcommitted available memory, requiring memory to be freed up.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.128


Medium-term Scheduler

```

            graph LR
            subgraph "partially executed swapped-out processes"
            P[ ]
            end
            subgraph "ready queue"
            R[ ]
            end
            subgraph "CPU"
            C(( ))
            end
            subgraph "I/O waiting queues"
            I[ ]
            end
            subgraph "I/O"
            IO(( ))
            end
            R --> C
            C --> P
            C --> I
            I --> IO
            IO --> R
            P --> R
            
```

Addition of medium-term scheduling to the queuing diagram


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.129



Context Switch

- When an interrupt occurs, the system needs to **save the current context of the process running on the CPU** so that it can restore that context when its processing is done.
- **Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.**
- This task is known as a context switch.
- The context is represented in the PCB of the process.
- **Context-switch time is pure overhead.**
- Context-switch times are highly dependent on hardware support.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.130



Operations on Processes: Creation

- A process may create several new processes during the course of execution.
 - The creating process is called a **parent process**.
 - The new processes are called the **children of that process**.
 - Each of these new processes may in turn create other processes, forming a **tree of processes**.
- Most operating systems identify processes according to a **unique process identifier (or pid)**, which is typically an integer number.
- When a process creates a new process, two possibilities exist for execution:
 - **The parent continues to execute concurrently with its children.**
 - **The parent waits until some or all of its children have terminated.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.131



Process Creation (contd...)

- There are also two possibilities for the address space of the new process:
 - **The child process is a duplicate of the parent process (it has the same program and data as the parent).**
 - **The child process has a new program loaded into it.**
- In **UNIX**, using **fork() system call**, the new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process.
- In **WINDOWS**, using **spawn() system call**, a specified program is loaded into the address space of the child process at process creation.
 - **UNIX systems implement this as a second step, using exec() system call.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.132

Operations on Processes: Termination

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using appropriate system call.
- At that point, the process may return a status value (typically an integer) to its parent process.
- All the resources of the process - including physical and virtual memory, open files, and I/O buffers - are deallocated by the operating system.
- Termination can occur in other circumstances as well.
 - A process can cause the termination of another process via an appropriate system call.
 - Usually, such a system call can be invoked only by the parent of the process that is to be terminated.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.133

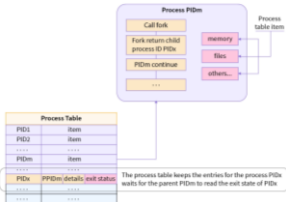
Process Termination (contd...)

- A parent may terminate the execution of one of its children for a variety of reasons:
 - The child has exceeded its usage of some of the resources that it has been allocated.
 - The task assigned to the child is no longer required.
 - The operating system does not allow a child to continue if its parent terminates.
- Some systems do not allow a child to exist if its parent has terminated.
 - In such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated.
- In UNIX, if the parent terminates, however, all its children have assigned as their new parent the `init` process. Thus, the children still have a parent to collect their status and execution statistics.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.134

Zombie Process in Operating System

- Zombie process is also known as "dead" process.
 - Ideally when a process completes its execution, its entry from the process table should be removed but this does not happen in case of zombie a process.



- **Analogy:** Zombie, mythological, is a dead person revived physically. Similarly, a zombie process in Operating System is a dead process (completed its execution) but is still revived (it's entry is present in memory).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.135

Reasons for Cooperation

- **Information Sharing**
 - Several users may be interested in the same piece of information (for instance, a shared file)
 - We must provide an environment to allow concurrent access to such information.
- **Computation Speedup**
 - We may break a task into subtasks, each of which will be executing in parallel with the others.
 - Such a speedup can be achieved only with multiple processing elements.
- **Modularity**
 - Dividing the system functions into separate processes or threads.
- **Convenience**
 - An individual user may work on many tasks at the same time.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.139

Interprocess Communication (contd...)

- Cooperating processes require an Interprocess Communication (IPC) mechanism.
 - IPC allows cooperating processes to exchange data and information.
- **Models of Interprocess Communication:**
 - **Shared Memory**
 - **Message Passing**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.140

Models of Interprocess Communication

Message Passing

Shared Memory


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.141



Shared-Memory System

- Communicating processes establish a region of shared memory.
 - A shared-memory region resides in the address space of the process that creates the shared-memory region.
 - Other processes that wish to communicate using this shared-memory region must attach shared address space to their address space.
 - Normally, the operating system tries to prevent one process from accessing another process's memory.
 - Shared memory requires that two or more processes agree to remove this restriction.
 - The form of data and location are determined by cooperating processes and are not under the operating system's control.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.142




Shared-Memory System (contd...)

Producer-Consumer Problem: A Paradigm for Cooperating Processes

- A producer process produces information that is consumed by a consumer process.
 - A Web server produces HTML, images, etc., client computer consume the information through the browser.
 - Compiler code makes assembly code consumed by an assembler.
- The use of shared memory is a solution of producer-consumer problem.
 - A buffer is maintained in the shared memory region, that is filled by a producer and emptied by the consumer.
 - A producer can produce one item while the consumer is consuming another item.
 - The producer and consumer are synchronized, so that the consumer could not try to consume an item that has not yet been produced.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.143



Shared-Memory System (contd...)

- Two types of buffers can be used:
 - **Unbounded Buffer**
 - The consumer waits for a new item, however, there is no restriction on the producer to produce items.
 - **Bounded Buffer**
 - Fixed buffer size.
 - If the buffer is empty, the consumer must wait for a new item.
 - When the buffer is full, the producer waits until it can produce new items.
 - A bounded buffer is implemented using a circular queue of an array.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.144

Message-Passing Systems

- It is useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.
- A message-passing facility provides at least two operations: `send(message)` and `receive(message)`.
- If processes **P** and **Q** want to communicate, they must send messages to and receive messages from each other.
- A communication link exists between the communicating processes.
 - Logically, the link can be implemented with several methods:
 - Direct or Indirect Communication
 - Synchronous or Asynchronous (Blocking or Non-blocking) Communication
 - Automatic or Explicit Buffering

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.145

Direct Communication

- Each process that wants to communicate must explicitly **name** the recipient or sender of the communication.
 - `send(P, message)` - Send a message to process P.
 - `receive(Q, message)` - Receive a message from process Q.
- **Properties of Communication Link:**
 - A link is established between every pair of processes that want to communicate.
 - The processes need to know only each other's **identity** to communicate.
 - A link is associated with exactly two processes.
 - Between each pair of processes, there exists exactly one link.
- **Changing the identifier of a process may necessitate examining all other process definitions.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.146

Indirect Communication

- The messages are sent to and received from **mailboxes** or **ports**.
 - A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
 - Each mailbox has a unique identification.
 - A process can communicate with some other process via a number of different mailboxes.
 - **Two processes can communicate only if the processes have a shared mailbox.**
 - `send(A, message)` - Send a message to mailbox A.
 - `receive(A, message)` - Receive a message from mailbox A.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.147

Indirect Communication (contd...)

- **Properties of Communication Link:**
 - A link is established between a pair of processes only if both members of the pair have a shared mailbox (port).
 - A link may be associated with more than two processes.
 - Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.
- A mailbox may be owned either by a process or by the operating system.
 - Suppose that processes P1, P2, and P3 all share mailbox A. Process P1 sends a message to A, while both P2 and P3 execute a receive() from A. Which process will receive the message sent by P1?
 - The answer depends on the scheme that we choose: (a) Allow a link to be associated with at most two processes. (b) Allow at most one process at a time to execute a receive operation. (c) Allow the system to select arbitrarily which process will receive the message.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.148

Synchronization

- Communication between processes takes place through calls to `send()` and `receive()` primitives.
- There are different design options for implementing each primitive.
- Message passing may be either **blocking** or **non-blocking** - also known as **synchronous** and **asynchronous**.
 - **Blocking send** - The sending process is blocked until the message is received by the receiving process or by the mailbox.
 - **Non-blocking send** - The sending process sends the message and resumes operation.
 - **Blocking receive** - The receiver blocks until a message is available.
 - **Non-blocking receive** - The receiver retrieves either a valid message or a null.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.149

Buffering

- Whether communication is direct or indirect, **messages exchanged by communicating processes reside in a temporary queue.**
- Such queues can be implemented in three ways:
 - **Zero Capacity** - This queue cannot keep any message waiting in it. For this, a sending process must be blocked until the receiving process receives the message. **It is also known as no buffering.**
 - **Bounded Capacity** - This queue has finite length n. Thus it can have n messages waiting in it. If the queue is not full, new message can be placed in the queue, and a sending process is not blocked. **It is also known as automatic buffering.**
 - **Unbounded Capacity** - The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.150

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Threads: Motivation

- Many software packages that run on modern desktop PCs are multithreaded.
- An application typically is implemented as a separate process with several threads of control.
 - A Web browser might have one thread display images or text while another thread retrieves data from the network.
 - A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.
 - Web server runs as a single process and creates a separate thread for each client request.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.151

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Single-threaded and Multithreaded Process


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.152

BHARATI VIDYAPEETH'S INSTITUTE OF COMPUTER APPLICATIONS AND MANAGEMENT

Benefits of Multithreading

- **Responsiveness**
 - A multithreaded application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation.
 - A multithreaded web browser could allow user interaction in one thread while an image was being loaded in another thread.
- **Resource Sharing**
 - Threads share the memory and the resources of the process to which they belong by default.
 - The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
- **Economy**
 - Threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.153



UNIX

- Unix is considered as the mother of most of the operating systems.
- Earlier, the operating systems were designed with a particular machine in mind.
 - They were invariably written in a low-level language.
 - The systems were fast but restricted to the hardware they were designed for.
- The development of UNIX was started in 1969, by Thompson and Ritchie, at AT&T.
 - Initially, they designed a small system with limited utilities.
 - In 1973, they rewrote the entire system in C language.
- **Unix is a family of multi-programing, multi-tasking and multi-user computer operating systems.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.154



Berkeley Software Distribution (BSD)

- A U. S. Govt. decree (subsequently revoked) prevented AT&T from selling computer software.
- Therefore, AT&T distributed UNIX to academic and research institutions **at nominal cost, but without any support.**
- The University of California, Berkeley created a UNIX of its own.
 - They called it **BSD UNIX (Berkeley Software Distribution)**
 - Berkeley filled the gaps left behind by AT&T, and rewrite the whole OS.
 - They created standard editor of UNIX system (vi) and a popular shell (C shell).
- BSD offered UNIX for free to many companies.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.155



Other Systems

- **Sun used the BSD system as a foundation for developing their own brand of UNIX (SunOS, now known as Solaris).**
- Other companies had their own brands, and **UNIX finally turned commercial.**
 - IBM -----> AIX
 - HP -----> HP-UX
- As each vendor modified and enhanced UNIX to create its own version, the **original UNIX lost its identity as a separate product.**
- AT&T sold its UNIX business to Novell, who later turned over the UNIX trademark to a standard body called X/OPEN, now merged with **The Open Group.**


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh
U1.156



Other Systems (contd...)

- UNIX is a registered trademark of **The Open Group**.
 - **The Open Group** refers to a family of computer operating systems and tools conforming to **The Open Group Base Specification**, Issue 7 (also known as POSIX.1-2008 or IEEE Std 1003.1 - 2008).
 - To use the Unix trademark, an operating system vendor must pay a licensing fee and annual trademark royalties to The Open Group.
 - Officially licensed Unix operating systems (and their vendors) include Mac (Apple), Solaris (Sun/Oracle), AIX (IBM), HP-UX (Hewlett-Packard), etc.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.157



UNIX Philosophy

- Make each program do one thing well.
 - Reusable software tools: 1 tool = 1 function
- Expect the output of every program to become the input of another, yet unknown, program to combine simple tools to perform complex tasks
- Everything seen as a file.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.158



UNIX-like Systems

- The operating systems that behave like UNIX systems and provide similar utilities, but do not conform to UNIX specification or are not licensed by The Open Group, are commonly known as Unix-like systems.
- These include a wide variety of Linux distributions, for example:
 - Red Hat Enterprise Linux
 - Ubuntu
 - CentOS
 - Several Descendants of the Berkeley Software Distribution OS
 - FreeBSD
 - OpenBSD
 - NetBSD

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.159

Linux

- Linux and Unix are different but they do have a relationship with each other as Linux is derived from Unix.
- Linux is a Unix-like kernel.
- Linux is a Unix clone.
- Linus Torvalds is the father of Linux – the free Unix.
- Linux is just the kernel and not the complete OS.
 - The Linux kernel is generally packaged in Linux distributions which thereby makes it a complete OS.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.160

Linux (contd...)

- Linux is distributed under the GNU General Public License which makes it mandatory for developers and sellers to make the source code public.
 - GNU, also known as Free Software Foundation, have written many important Linux tools .
 - Development of GNU/Linux began in 1984, when the Free Software Foundation began development of a free Unix-like operating system called GNU.
 - Today, development on Linux is carried out at several locations across the globe at the behest of the Free Software Foundation.
 - The most popular GNU/Linux distributions include Red Hat, Caldera, Debian (Ubuntu) and Mandrake.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.161

Architecture of GNU/Linux

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.162

Features of GNU/Linux

- Portable
- Open Source
- Multiuser
- Multiprogramming
- Hierarchical File System
- Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.163

Linux File System

- In Linux, everything is seen as a file.
- Files are ordered in a tree structure starting with the root directory.
- This root directory can be considered as the start of the file system, and it further branches out various other subdirectories.
- The root is denoted with a forward slash '/'.


```

graph TD
    Root["/(Root)"] --> etc
    Root --> bin
    Root --> usr
    Root --> tmp
    Root --> dev
    bin --> public
    bin --> misc
    bin --> staff
    usr --> software
    usr --> doc
    usr --> john
    usr --> mary
    usr --> bill
    usr --> carl
            
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.164

Linux File System (contd...)

- Linux is **cAsE sEnSiTiVe**
 - Each file in a given directory must be unique.

Windows

Linux

- In Linux, we can have 2 files with the same name in the same directory, provided they use different cases.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.165

Types of Files

- In Linux and UNIX, everything is a file.
- If something is not a file, then it must be running as a process on the system.
- **General Files (Ordinary Files)** - They can contain image, video, program or simply text. They can be in ASCII or a Binary format.
- **Directory Files** - These files are a warehouse for other file types. We can have a directory file within a directory (sub-directory). We can take them as 'Folders' found in Windows operating system.
- **Device Files** - In MS Windows, devices like Printers, CD-ROM, and hard drives are represented as drive letters like G: H:. In Linux, there are represented as files. For example, if the hard drive has three partitions, they would be named and numbered as `/dev/sda1`, `/dev/sda2` and `/dev/sda3`.
- All device files reside in the directory `/dev/`

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.166

Linux File System (contd...)


- Files are referenced by names.
- **/home/sunil**
 - First `/` represents the root directory.
 - It is an absolute pathname, which is a sequence of directory names separated by slashes.
 - An absolute pathname shows a file's location with reference to the top, i.e., root.
- **Shortcuts**
 - .. (Parent Directory)
 - .(Current Directory)
 - ~ (Home Directory)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.167

Windows Vs. Linux

Windows	Linux
Windows uses different data drives like C: D: E to stored files and folders.	Unix/Linux uses a tree like a hierarchical file system.
Windows has different drives like C: D: E	There are no drives in Linux
Hard drives, CD-ROMs, printers are considered as devices	Peripherals like hard drives, CD-ROMs, printers are also considered files in Linux/Unix
There are 4 types of user account types 1) Administrator, 2) Standard, 3) Child, 4) Guest	There are 3 types of user account types 1) Regular, 2) Root and 3) Service Account
Administrator user has all administrative privileges of computers.	Root user is the super user and has all administrative privileges.
In Windows, you cannot have 2 files with the same name in the same folder	Linux file naming convention is case sensitive. Thus, sample and SAMPLE are 2 different files in Linux/Unix operating system.
In windows, My Documents is default home directory.	For every user /home/username directory is created which is called his home directory.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.168



Commands in Linux

- Linux commands are words that when keyed in cause some action to take place.
- Commands are actually files containing programs often written in C. These files are contained in directories.
- Seldom more than four characters in length like ls, who, ps.
- Lower case
- Case sensitive


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.169



Command Interpreter/Shell

- Shell is a program that acts as the interface between user and the Linux system.
- It allows us to enter commands for the operating system to execute.
- There are multiple shells in Linux:
 - Bash (standard shell in Linux)
 - Bourne shell (sh)
 - C Shell (csh)
 - Korn shell (ksh)
- When we key in a command at the command prompt, the shell searches for a file matching the command name in a list of directories.
- If it is found, then the program corresponding to the file is executed. If it is not found, a command not found error is given.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.170



Types of Commands

- External Commands
 - External commands are those which exist as separate files programs.
 - Most commands in Linux are of this nature.
 - Example: **ps**
- Internal Commands
 - The commands whose implementation is written within the shell.
 - They do not exist as a separate file.
 - Example: **echo**
- With **type** command, we can determine whether a command is external or internal.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.171



Structure of Commands

- Command can be one word or multiple words separated by white spaces.
- In multiple words command:
 - **First word:** the **actual name of the command**
 - **Other words:** **arguments** (can be options or expressions or file names)
 - A command can perform different task depending on the option specified.
 - The options are generally preceded by a by a single or double minus sign, called **Short** or **Long** option respectively.
 - **Commands can be combined together as per need.**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Sunil Pratap Singh U1.172



Operating Systems with Linux

(MCA-105)

Unit - 1

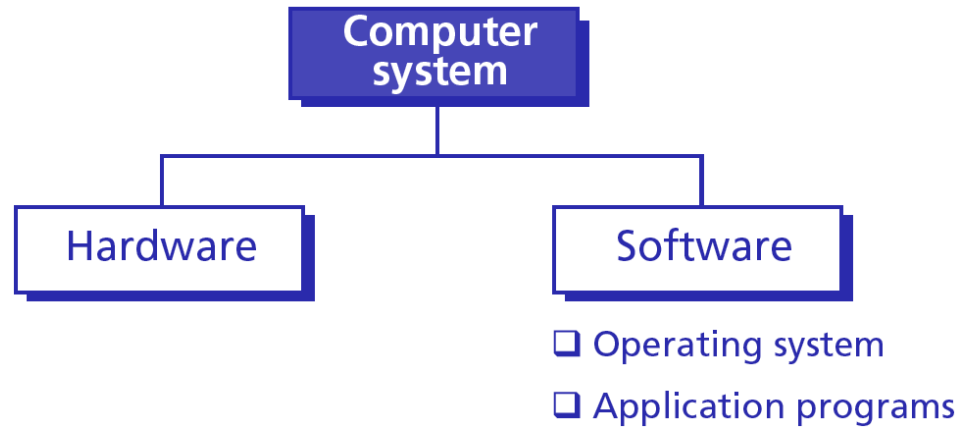
by

Dr. Sunil Pratap Singh
(Associate Professor, BVICAM, New Delhi)

2023

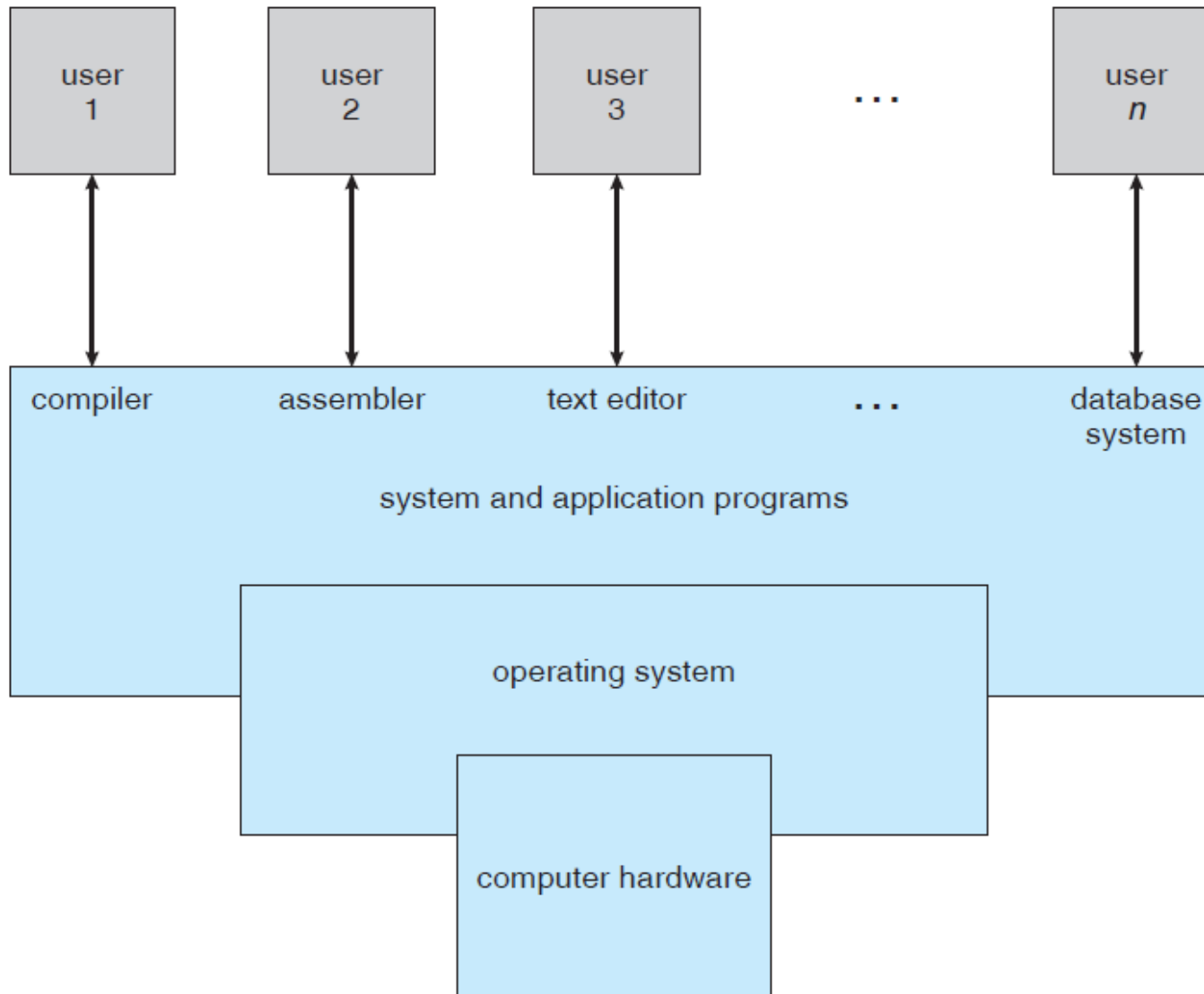
Computer System

- A computer is a system composed of two major components:



- Computer hardware is the physical equipment.
- Software is the collection of programs that allows the hardware to do its job.
 - Computer software is divided into two broad categories:
 - **Application programs** - use the computer hardware to solve users' problems.
 - **Operating system** - controls the access to hardware by users.

Abstract View of Components of a Computer System



Operating System: Common Definitions

- An operating system is an interface between the hardware of a computer and the user (programs or humans).
- An operating system is a program (or a set of programs) that facilitates the execution of other programs.
- An operating system acts as a general manager supervising the activity of each component in the computer system.
 - As a general manager, the operating system checks that hardware and software resources are used efficiently, and when there is a conflict in using a resource, the operating system mediates to solve it.

Design Goals

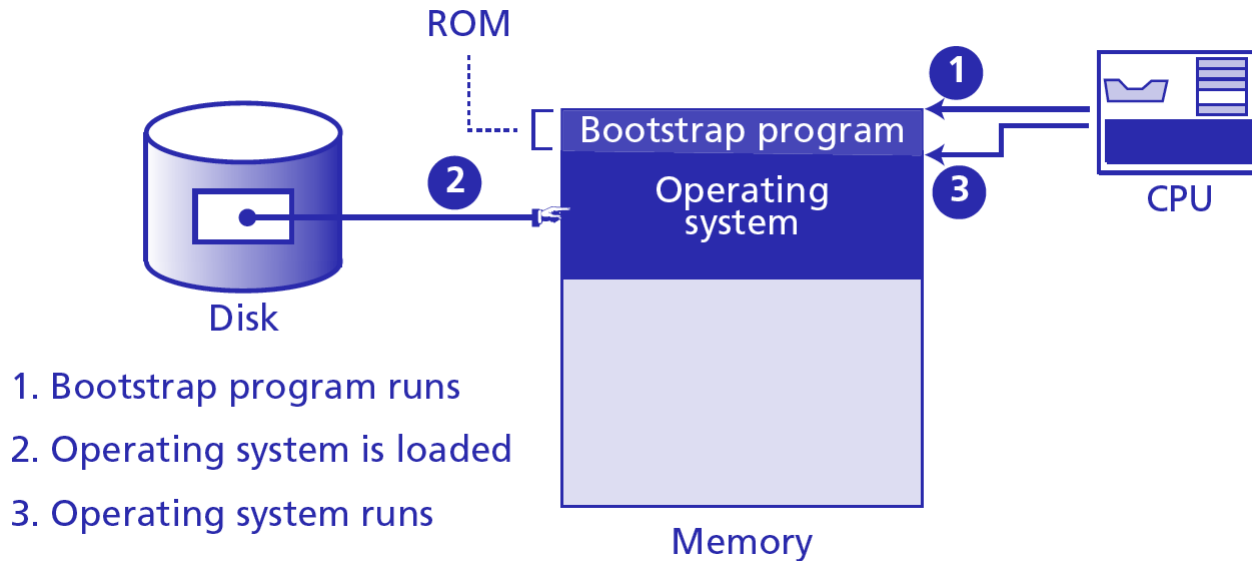
- Major design goals of an operating system are:
 - Efficient (efficient use of hardware)
 - Convenient (easy use of resources)

Bootstrap Process

- As per the definitions, the operating system provides support for other programs.
 - It is responsible for loading other programs into memory for execution.
 - However, the operating system itself is a program that needs to be loaded into the memory and be run.

How is this dilemma solved? → Bootstrap Process

Bootstrap Process (contd...)



- A very small section of memory (ROM) holds a small program called the **bootstrap program**.
- When the computer is turned on, the CPU counter is set to the first instruction of this bootstrap program and executes the instructions in this program.
- This program is only responsible for loading the operating system itself, or that part of it required to start up the computer, into RAM memory.
- When loading is done, the program counter in the CPU is set to the first instruction of the operating system in RAM and the operating system is executed.

Operating System: User View

- Case 1: When the user sits in front of a PC, consisting of a monitor, keyboard, mouse, and system unit.
 - Such a system is designed for one user to **monopolize its resources**.
 - The goal is to maximize the work (or play) that the user is performing.

In this case, the operating system is designed mostly for ease of use, with some attention paid to performance and none paid to resource utilization.

- As performance is important to user, such systems are optimized for single-user experience.

Operating System: User View (contd...)

- Case 2: When the user sits at a terminal connected to a mainframe or a minicomputer.
 - In such system, other users are accessing the same computer through other terminals.
 - These users share resources and may exchange information.

In this case, the operating system is designed to **maximize resource utilization** - to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.

Operating System: User View (contd...)

- Case 3: When the users sit at workstations connected to networks of other workstations and servers.
 - In such system, users have dedicated resources at their disposal, but they also share resources such as networking and servers - file, compute, and print servers.

In this case, the operating system is designed to **compromise** between **individual usability** and **resource utilization**.

Operating System: User View (contd...)

- Case 4: When the users work with standalone handheld computers, may or may not be connected to networks.
 - Such systems have limitations of power, speed, and interface, and they perform relatively few remote operations.

In this case, the operating system is designed mostly for individual usability, but performance per unit of battery life is important as well.



Operating System: User View (contd...)

- Case 5: When computers have little or no user view.
 - Embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status.

In this case, the operating system is designed primarily to run without user intervention.

Operating System: System View

- From the Computer's point of view, the operating system is the program most intimately involved with the hardware.

In this context, an operating system can be viewed as a **resource allocator**.

- A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on.
- The operating system acts as the manager of these resources.
- Resource allocation is especially important where many users access the same computer.

- From the Computer's point of view, the operating system controls the various I/O devices and user programs.

In this context, an operating system can be viewed as a **control program** which manages the execution of user programs to prevent errors and improper use of the computer.

- It is especially concerned with the operation and control of I/O devices.

Kernel

- A **Kernel** is a **computer program** that is the heart and **core** of an Operating System.
- Every operating system - whether it is Windows, Mac, Linux, or Android, has a **core program** known as **Kernel** which acts as the 'boss' for the whole system.
- It is the **most important part** of an Operating System.
- Whenever a system starts, the Kernel is the **first program** that is loaded after the boot-loader because the Kernel has to handle the rest of the thing of the system for the Operating System.

Kernel (contd...)

- The Kernel remains in the memory until the Operating System is shut-down.
- The Kernel is responsible for disk management, memory management, task management, i/o management, etc.
- When a process makes a request to the Kernel, then it is called **System Call**.
- When we launch a program, the user interface sends a request to Kernel. The Kernel then sends a request to CPU, Memory to assign processing power, memory, and other things so the application can run smoothly in the front end.

Kernel Space and User Space

- A Kernel is provided with a protected **Kernel Space** which is a separate area of memory.
 - This area is not accessible by other application programs.
 - The code of the Kernel is loaded into this protected Kernel Space.
- The memory used by other applications is called the **User Space**.
- Communication between Kernel Space and User Space is a bit slower because these are two different spaces in the memory.

Types of Kernel

- **Monolithic Kernel**

- In monolithic kernel, all the services (**user services** and **kernel services**) of operating system are implemented in the **same memory space**.

- **Microkernel**

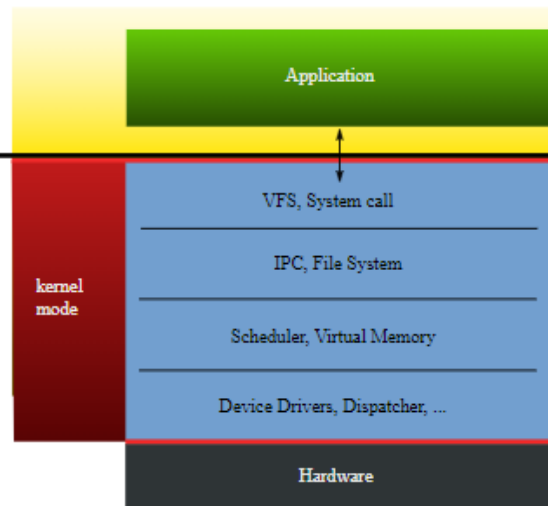
- In Microkernel, the **user services** and **kernel services** are implemented into different spaces, i.e. we use **User Space** and **Kernel Space** in case of Microkernels.

- **Hybrid Kernel**

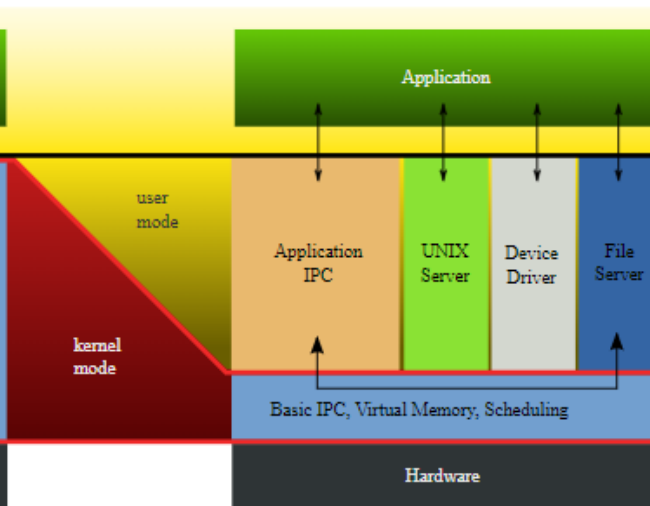
- A Hybrid Kernel is a **combination** of both Monolithic Kernel and Microkernel. It makes the use of the speed of Monolithic Kernel and the modularity of Microkernel.

Types of Kernel (contd...)

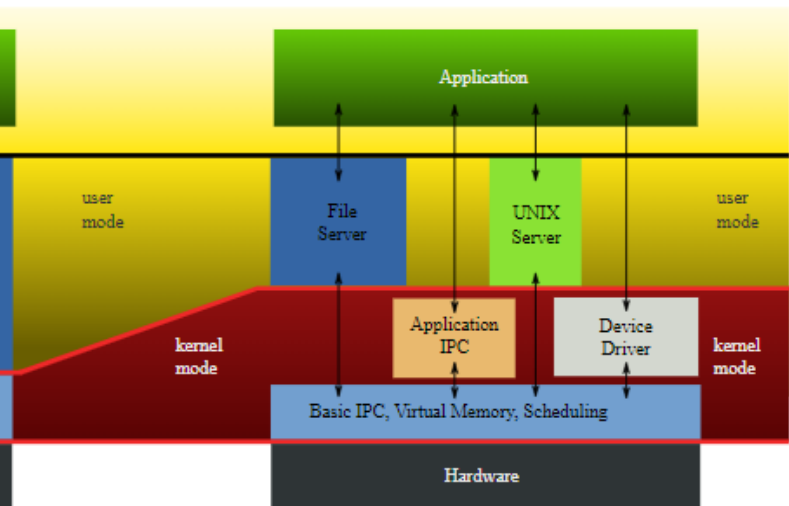
Monolithic Kernel based Operating System



Microkernel based Operating System



"Hybrid kernel" based Operating System



Difference b/w Monolithic and Microkernel

BASIS	MONOLITHIC KERNEL	MICROKERNEL
Basic	Both, user services and kernel services are kept in the same address space.	User services and kernel, services are kept in separate address space.
Size	Monolithic kernel is larger than microkernel.	Microkernel are smaller in size.
Execution	Fast execution (through system call)	Slow execution (through message passing)
Extendable	The monolithic kernel is hard to extend.	The microkernel is easily extendible.
Security	If a service crashes, the whole system crashes in monolithic kernel.	If a service crashes, it does effect on working of microkernel.
Example	Linux, Unix, Microsoft Windows (95, 98, Me), DOS, OpenVMS, etc.	Symbian, Singularity, PikeOS, Minix, etc.

Difference b/w Operating System and Kernel

OPERATING SYSTEM	KERNEL
Operating System is a system software.	Kernel is system software which is part of operating system.
Operating System provides interface b/w user and hardware.	Kernel provides interface b/w application and hardware.
It also provides protection and security.	It's main purpose is memory management, disk management, process management and task management.
All system needs operating system to run.	All operating system needs kernel to run.
It is the first program to load when computer boots up.	It is the first program to load when operating system loads.

Components of Operating System

- An operating system has at least four duties:
 - Memory Manager,
 - Process Manager,
 - Device Manager, and
 - Storage Manager.

- Many organizations that have a department that is not necessarily under any specific manager, an operating system also has such a component, which is usually called a **user interface**.

User Interface

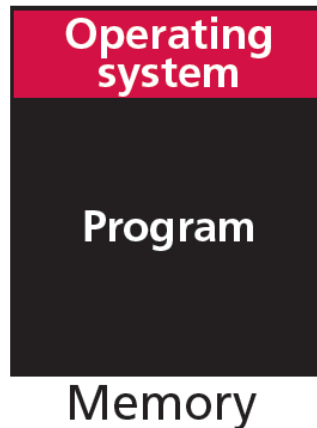
- Each operating system has a user interface, a **program** that **accepts requests** from users (processes) and **interprets** them for the rest of the operating system.
- A user interface in some operating systems, such as **UNIX**, is called a **shell**.
- In others, it is called a **window** to denote that it is menu driven and has a **GUI (graphical user interface)** component.

Memory Manager

- Memory allocation must be managed to prevent applications from running out of memory.
- Operating systems can be divided into two broad categories of memory management:
 - Monoprogramming
 - Multiprogramming

Monoprogramming

- In monoprogramming, **most of the memory capacity is dedicated to a single program** (the data to be processed by a program can be considered as part of the program).
- Only a small part is needed to hold the operating system.



- The job of the memory manager is straightforward here. **It loads the program into memory, runs it, and replaces it with the next program.**

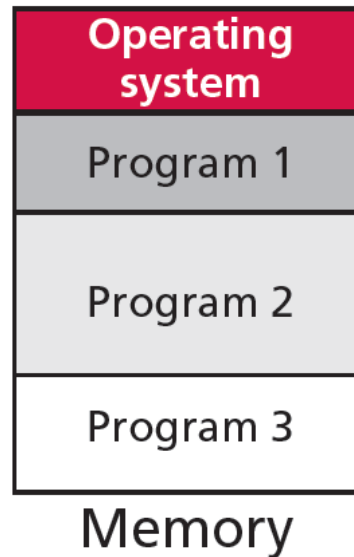
Problems with Monoprogramming

- The program must fit into memory.
 - If the size of memory is less than the size of the program, the program cannot be run.
- When one program is being run, no other program can be executed.
 - A program, during its execution, often needs to receive data from input devices and needs to send data to output devices. Input/output devices are slow compared with the CPU, so when the input/output operations are being carried out, the CPU is idle.
 - It cannot serve another program because this program is not in memory.
 - This is a very inefficient use of memory and CPU time.

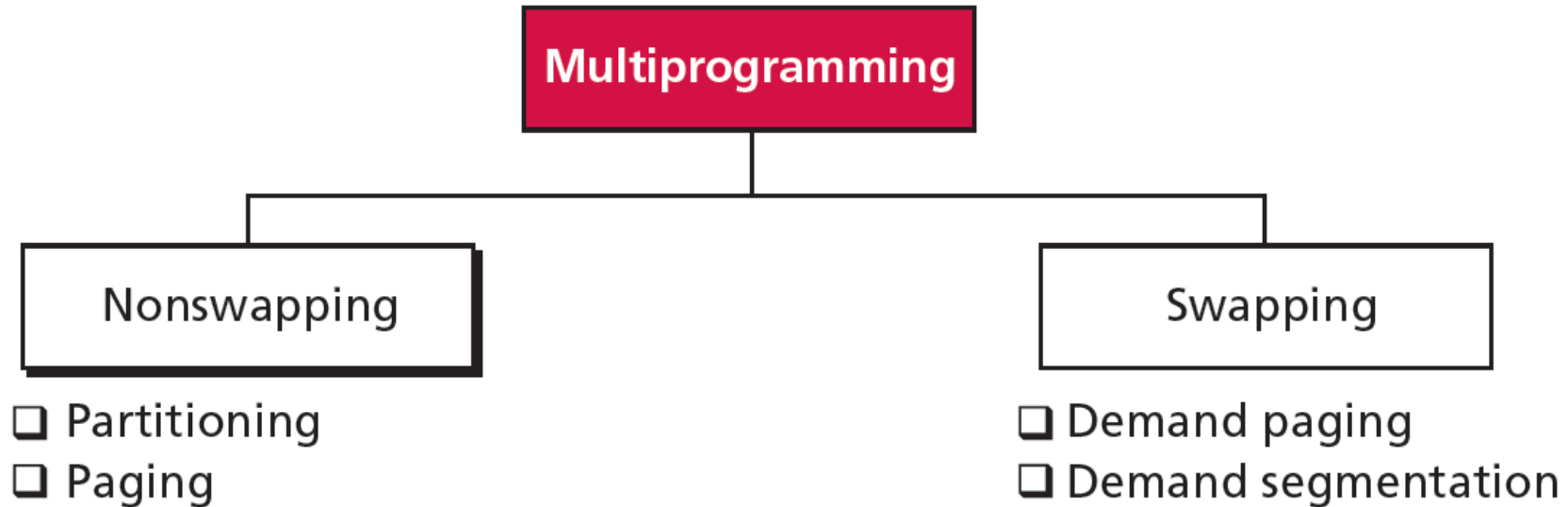
Monoprogramming belongs to the past, but it is important to understand multiprogramming.

Multiprogramming

- In multiprogramming, more than one program is in memory at the same time, and they are executed concurrently, with the CPU switching rapidly between the programs.



Categories of Multiprogramming



- **Nonswapping** - The program remains in memory for the duration of execution.
- **Swapping** - During execution, the program can be swapped between memory and disk one or more times.



Responsibilities of Memory Manager

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

Process Manager

- Modern operating systems use three terms that refer to a set of instructions:
 - Program
 - Job
 - Process
- **Program:**
 - A program is a non-active set of instructions stored on disk (or tape).

Process Manager: Job

- **Job:**

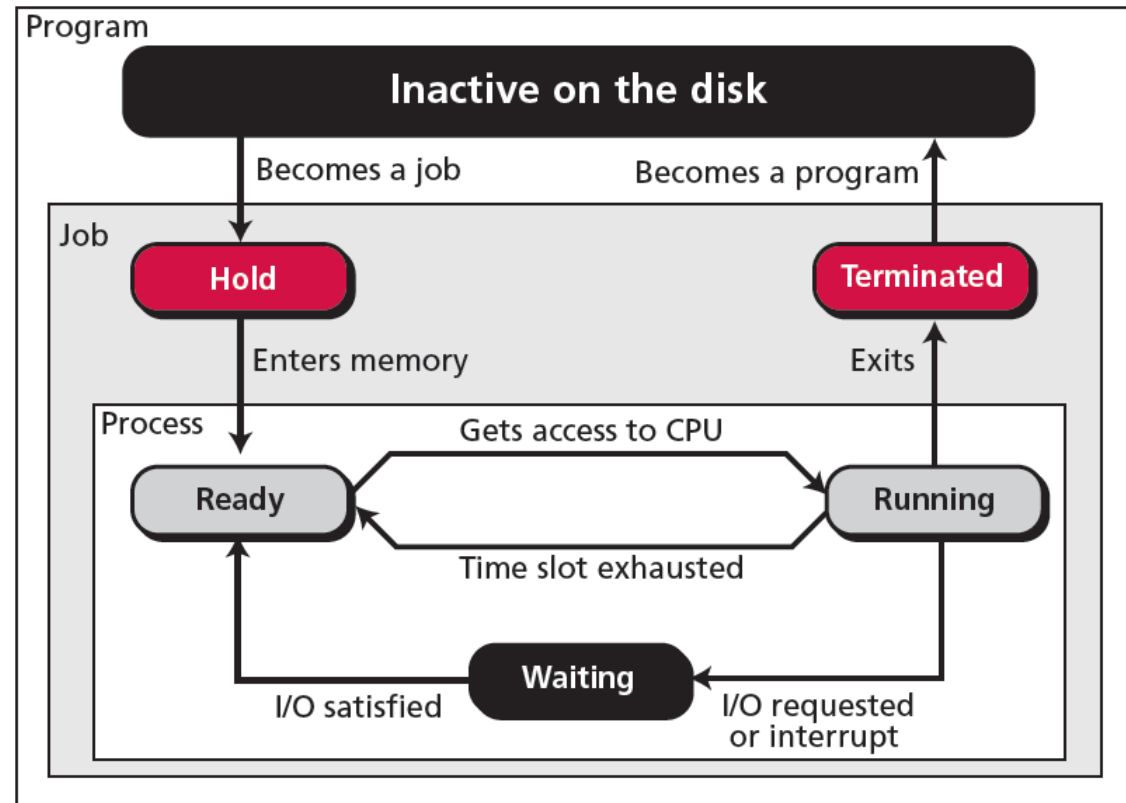
- A **program becomes a job** from the moment it is selected for execution until it has finished running and becomes a program again.
- During this time a job may or may not be executed.
- It may be located on disk waiting to be loaded to memory, or it may be loaded into memory and waiting for execution by the CPU.
- When a job has finished executing (either normally or abnormally), it becomes a program and once again resides on the disk.
- **Every job is a program, but every program is not a job.**

- **Process:**

- A process is a program in execution.
- It is a program that has started but has not finished.
- In other words, a process is a job that is being run in memory.
- It has been selected among other waiting jobs and loaded into memory.
- A process may be executing or it may be waiting for CPU time.
- As long as the job is in memory, it is a process.
- **Every process is a job, but every job is not a process.**

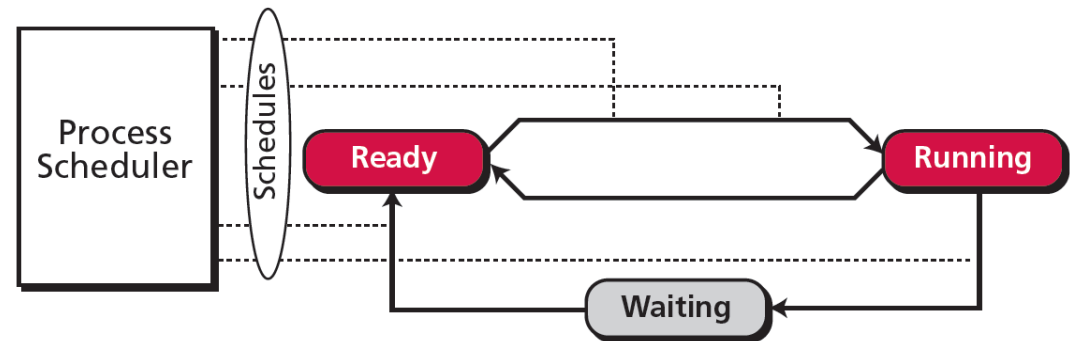
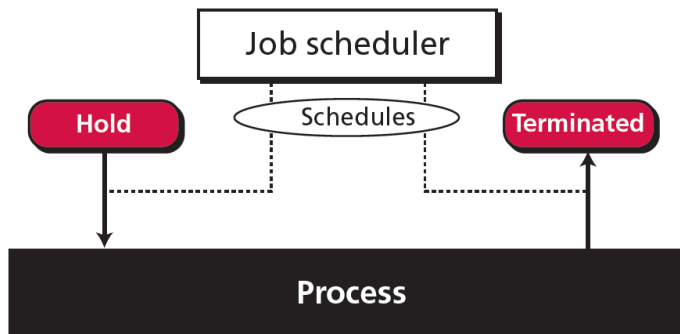
In the running state, one of three things can happen:

- The process executes until it needs I/O resources.
 - The process exhausts its allocated time slot.
 - The process terminates.
- In the first case, the process goes into the **waiting state** and waits until I/O is complete.
 - In the second case, it goes directly to the **ready state**.
 - In the third case, it goes into the **terminated state** and is no longer a process.



Process Manager: Schedulers

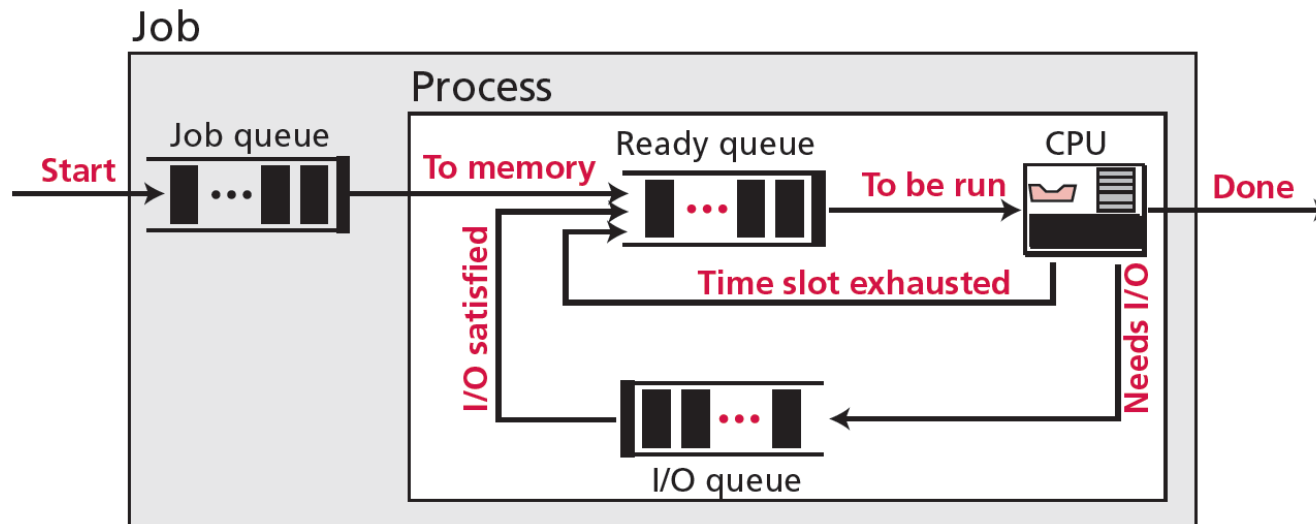
- To move a job or process from one state to another, the process manager uses two schedulers:
 - Job Scheduler, and
 - Process Scheduler



- The **job scheduler** moves a job from the hold state to the ready state or from the running state to the terminated state.
- The **process scheduler** moves a process from one state to another.

Process Manager: Queuing

- To handle multiple processes and jobs, the process manager uses queues (waiting lists).
- An operating system can have several queues.



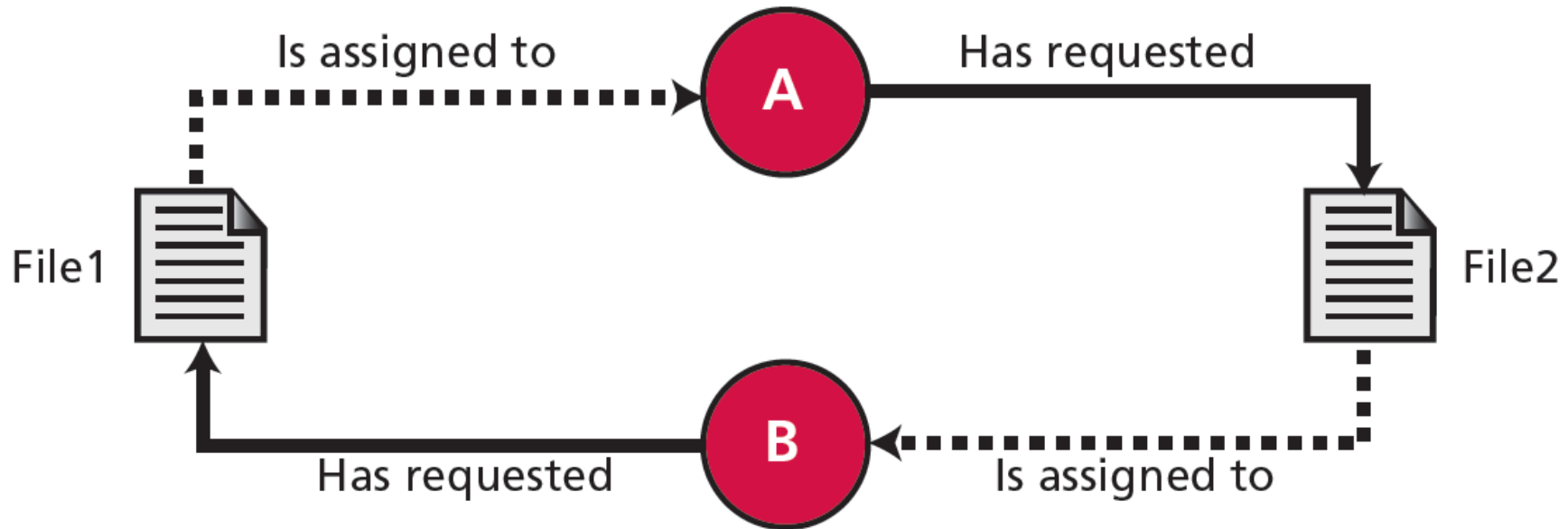
- The process manager can have different policies for selecting the next job or process from a queue: it could be first in, first out (FIFO), shortest length first, highest priority first, and so on.

Process Manager: Process Synchronization

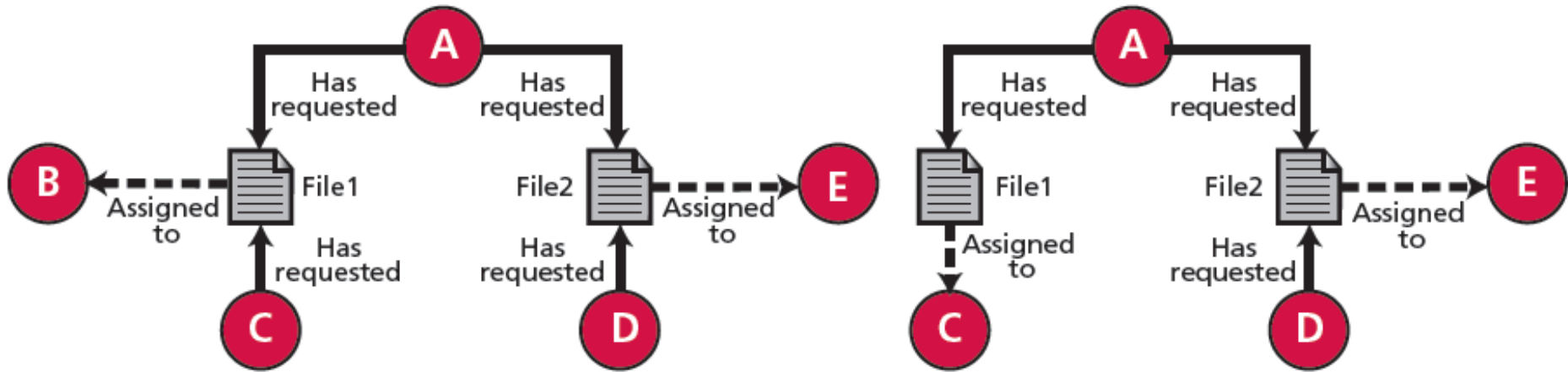
- The main idea behind process management is to **synchronize different processes with different resources**.
- Whenever resources can be used by more than one user (or process, in this case), we can have two problematic situations:
 - **Deadlock** - Deadlock occurs when the operating system does not put resource restrictions on processes.
 - **Starvation** - Starvation is the opposite of deadlock. It can happen when the operating system puts too many resource restrictions on a process.

Process Manager: Deadlock

- Deadlock occurs if the operating system allows a process to start running without first checking to see if the required resources are ready, and allows a process to hold a resource as long as it wants.

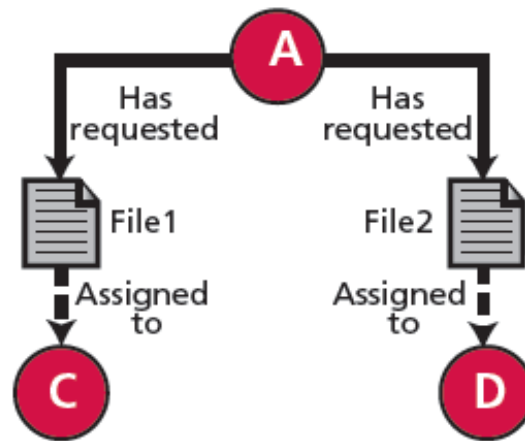


Process Manager: Starvation



a. Process A needs both File1 and File2

b. Process A still needs both File1 and File2



c. Process A still needs both File1 and File2 (starving)

Responsibilities of Process Manager

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

Responsibilities of Device Manager (I/O Manager)

- Monitors every input/output device constantly to ensure that the device is functioning properly. The manager also needs to know when a device has finished serving one process and is ready to serve the next process in the queue.
- Maintains a queue for each input/output device or one or more queues for similar input/output devices. For example, if there are two fast printers in the system, the manager can have one queue for each or one queue for both.
- Controls the different policies for accessing input/output devices. For example, it may use FIFO for one device and shortest length first for another.

Storage Manager

- To make the computer system convenient for users, the operating system provides a uniform, logical view of information storage.
- The operating system maps files onto physical media and accesses these files via the storage devices.
- File-System Management
- Mass-Storage (Disk) Management

File-System Management

- Computers can store information on several different types of physical media.
- Magnetic disk, optical disk, and magnetic tape are the most common.
- Each of these media has its own characteristics and physical organization.
- Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.
- These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).



File-System Management (contd..)

- A file is a collection of related information defined by its creator.
- Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files are normally organized into directories to make them easier to use.
- When multiple users have access to files, it may be desirable to control by whom and in what ways (for example, read, write, append) files may be accessed.

Responsibilities of File-System Management

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

Mass-Storage (Disk) Management

- The computer system must provide secondary storage to back up main memory because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost.
- Modern computer systems use disks as the principal storage medium for both, programs and data.
- Most programs – including compilers, assemblers, word processors, editors, and formatters - are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing.
- Hence, the proper management of disk storage is of central importance to a computer system.

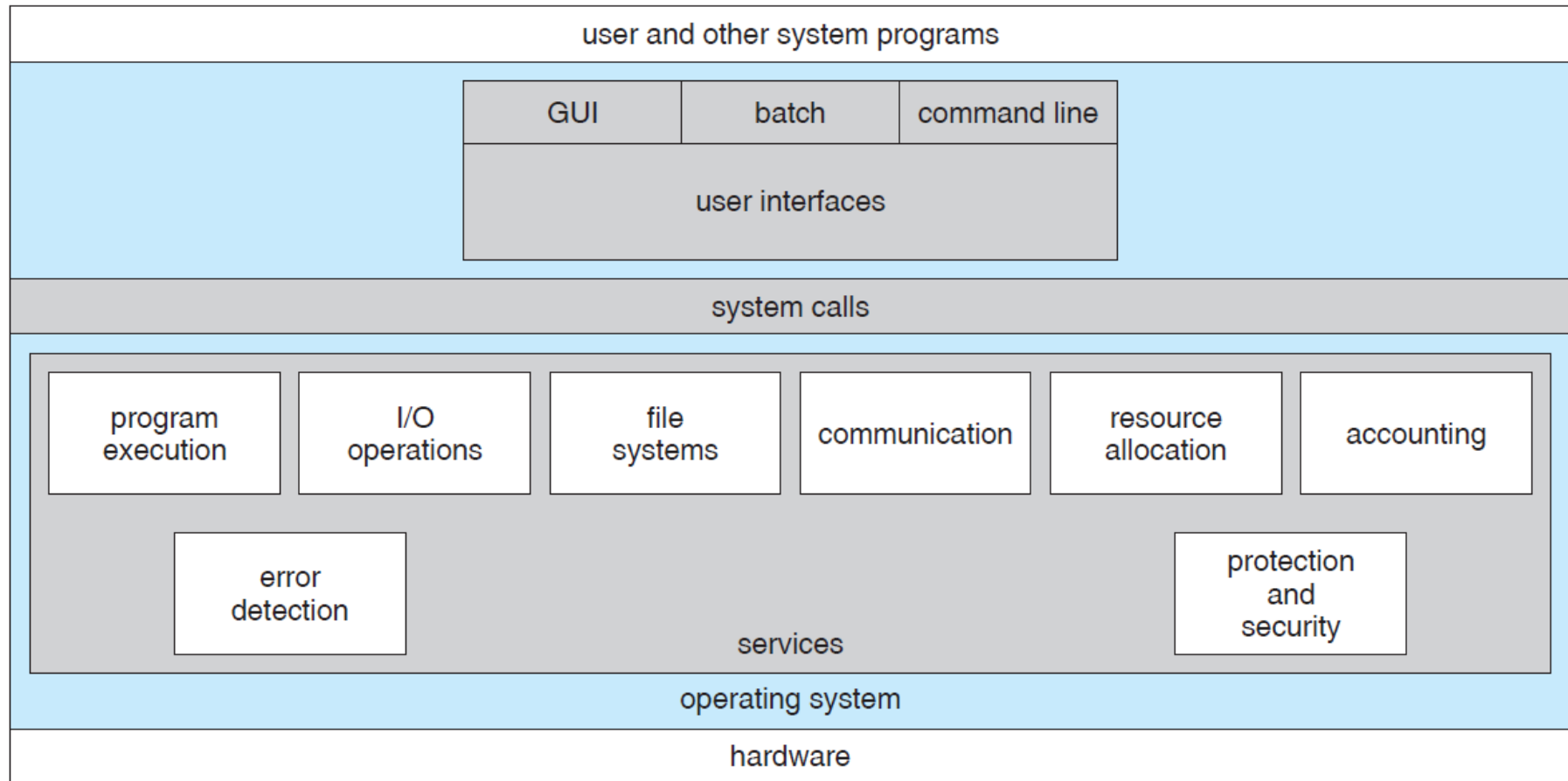


Responsibilities of Mass-Storage Management

- Free-space management
- Storage allocation
- Disk scheduling

Services of Operating System

- An operating system provides an environment for the execution of programs.
- It provides certain services to programs and to the users of those programs.



User Interface:

- Almost all operating systems have a User Interface (UI).
- User Interface can take several forms:
 - **Command-line Interface (CLI)** - Uses text commands and a method for entering them.
 - **Batch Interface** - Commands and directives to control those commands are entered into files, and those files are executed.
 - **Graphical User Interface (GUI)** – It is the most commonly used UI. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

Program Execution:

- Load a program into memory and run that program.
- The program must be able to end its execution, either normally or abnormally (indicating error).

I/O Operations:

- The operating system provides a means to do I/O. For efficiency and protection, users usually cannot control I/O devices directly.
- A running program may require I/O, which may involve a file or an I/O device.
- For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen).

File System:

- The operating system provides services to read and write files and directories.
- Some programs include permissions management to allow or deny access to files or directories based on file ownership.

Communication:

- The operating system provides services for exchanging information between processes.
- Communication may occur **between processes that are executing on the same computer** or **between processes that are executing on different computer systems tied together by a computer network.**

Resource Allocation:

- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- Many different types of resources are managed by the operating system.
- Some resources (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.

Accounting:

- The operating system provides services to keep track of which users use how much and what kinds of computer resources.
- This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics
- Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and Security:

- Protection involves ensuring that all access to system resources is controlled.
- When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself.
- Security of the system from outsiders is also important.
- Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.
- It extends to defending external I/O devices including modems, network adapters, etc. from invalid access attempts.



Computer-System Architecture

- A computer system may be organized in a number of different ways.
 - We can categorize roughly according to the number of general-purpose processors used.
 - Single-Processor Systems
 - Multiprocessor Systems
 - Clustered Systems

Single-Processor Systems

- Most systems use a single processor.
- These systems range from PDAs through mainframes.
- **There is one main CPU** capable of executing a general-purpose instruction set, **including instructions from user processes**.
- **Almost all systems have other special-purpose processors as well.**
 - They may come in the form of device-specific processors such as disk, keyboard, and graphics controllers.
 - The special-purpose processors run a limited instruction set and do not run user processes.

Examples of Special-purpose Processors:

- PCs contain a disk-controller microprocessor which receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm.
 - This arrangement relieves the main CPU of the overhead of disk scheduling.
- PCs contain a keyboard microprocessor to convert the keystrokes into codes to be sent to the CPU.
- The operating system cannot communicate with these processors; they do their jobs autonomously.
- The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor.

Multiprocessor Systems

- Multiprocessor systems have two or more processors in close communication, sharing the computer resources.
- **Main Advantages of Multiprocessor Systems:**
 - **Increased Throughput**
 - By increasing the number of processors, we expect to get more work done in less time.
 - The speed-up ratio with N processors is not N , however; rather, it is less than N .
 - **Economy of Scale**
 - Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies
 - **Increased Reliability**
 - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

Types of Multiprocessor Systems

- **Asymmetric Multiprocessing (AMP)**

- Each processor is assigned a specific task.
- A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks.
- This scheme defines a master–slave relationship.
- The master processor schedules and allocates work to the slave processors.
- **Example:** Sun’s Operating System (Version 4) – SunOS, IOS

- **Symmetric Multiprocessing (SMP)**

- Each processor performs all tasks within the operating system.
- All processors are peers; no master–slave relationship exists between processors.
- Each processor has its own set of registers, as well as a cache; however, all processors share physical memory.
- **Examples:** Sun’s Operating System (Version 5) – Solaris, Windows, Unix, Linux

docs.microsoft.com/en-us/windows-hardware/drivers/kernel/multiprocessor-safe

Microsoft | Docs | Documentation | Learn | Q&A | Code Samples | Shows | Events


Windows Hardware Developer | Explore | Downloads | Events | Samples | Support | Dashboard

Docs / Windows / Windows Drivers / Kernel-Mode Driver Architecture

Filter by title

- Kernel-Mode Driver Architecture Design Guide
- Overview
 - Windows Components
 - Types of Windows Drivers
 - Design Goals for Kernel-Mode Drivers
 - Portable
 - Configurable
 - Always Preemptible and Always Interruptible
 - Multiprocessor-Safe**
 - Object-Based
 - Packet-Driven I/O with Reusable IRPs

Multiprocessor-Safe

Article • 12/08/2020 • 3 minutes to read •  [Is this page helpful?](#)

The Microsoft Windows NT-based operating system is designed to run uniformly on uniprocessor and symmetric multiprocessor (SMP) platforms, and kernel-mode drivers should be designed to do likewise.

In any Windows multiprocessor platform, the following conditions exist:

- All CPUs are identical, and either all or none of the processors must have identical coprocessors.
- All CPUs share memory and have uniform access to memory.
- In a *symmetric* platform, every CPU can access memory, take an interrupt, and access I/O control registers. (By contrast, in an *asymmetric* multiprocessor machine, one CPU takes all interrupts for a set of subordinate CPUs.)

Multiprocessor Systems (contd...)

- A recent trend in CPU design is to include multiple computing cores on a single chip.

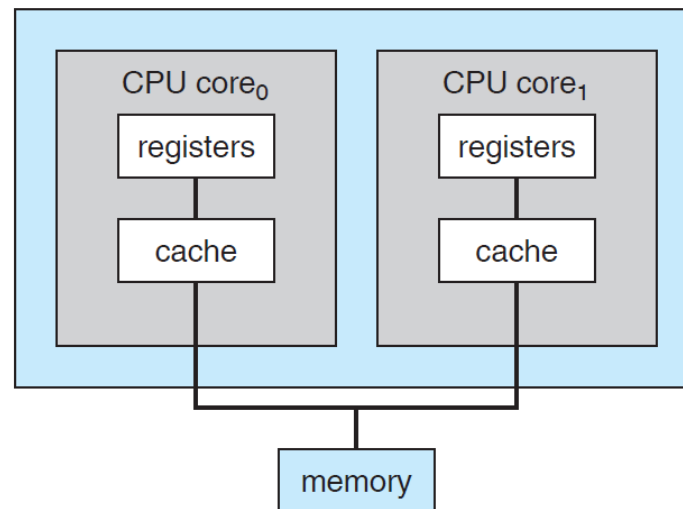


Figure: A dual-core design with two cores placed on the same chip.

- In essence, these are multiprocessor chips.
- They can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication.
- In addition, one chip with multiple cores uses significantly less power than multiple single-core chips.
- Multicore systems are especially well suited for server systems such as database and Web servers.

Operating-System Operations

- The operating system's nature is **interrupt driven**.
- **Interrupts** are the signals that are triggered by the software or the hardware when any process or an event needs immediate attention.
 - If we move our mouse on the screen, an interrupt is getting generated.
 - If we press any key on a keyboard, an interrupt is getting generated.
 - If we print a document, an interrupt is getting generated.
 - If we power off the monitor screen, an interrupt is getting generated.
 - There are many such occasion such that operating system is waiting for some interrupts to take place and for attending the interrupts.

Operating-System Operations (contd..)

- **Hardware** may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.
- **Software** may trigger an interrupt by executing a special operation called a system call.
- **Interrupts, when generated by the hardware or the software have to be attended by the operating system.**
- The operating system has some processes that are operating and these processes are present in the queue, so **when the operating system receives the interrupt signals, all the processes in the queue that were running are made to halt and CPU is made to attend to the signal of the interrupt.**

Operating-System Operations (contd..)

- A **Trap** is a software-generated interrupt which can be caused by some reason such as:
 - Error in the instruction (for example, division by zero or invalid memory access).
 - User program initiates a specific service request from the operating system (for example, the user program requires printing something to the screen; it would invoke a trap and the operating system will perform writing that data to the screen).

Operating-System Operations (contd..)

- The **system calls are example of a trap** where the operating system is asked by the program to request a particular service and the operating system then generates an interrupt to access the services for the program.
 - Example: The statement `printf(“%s\n”, str);` will invoke the write function to print the output to the standard output which is the monitor. This will invoke a trap and it will pass the control to the trap handler. Then, the user mode changes to kernel mode and the OS executes the write call. After completing the task, the control is transferred back to the user mode from the kernel mode.
- The traps are more likely to be caused by the execution of the current instructions and therefore **the traps are also known as synchronous events.**

Operating-System Operations (contd..)

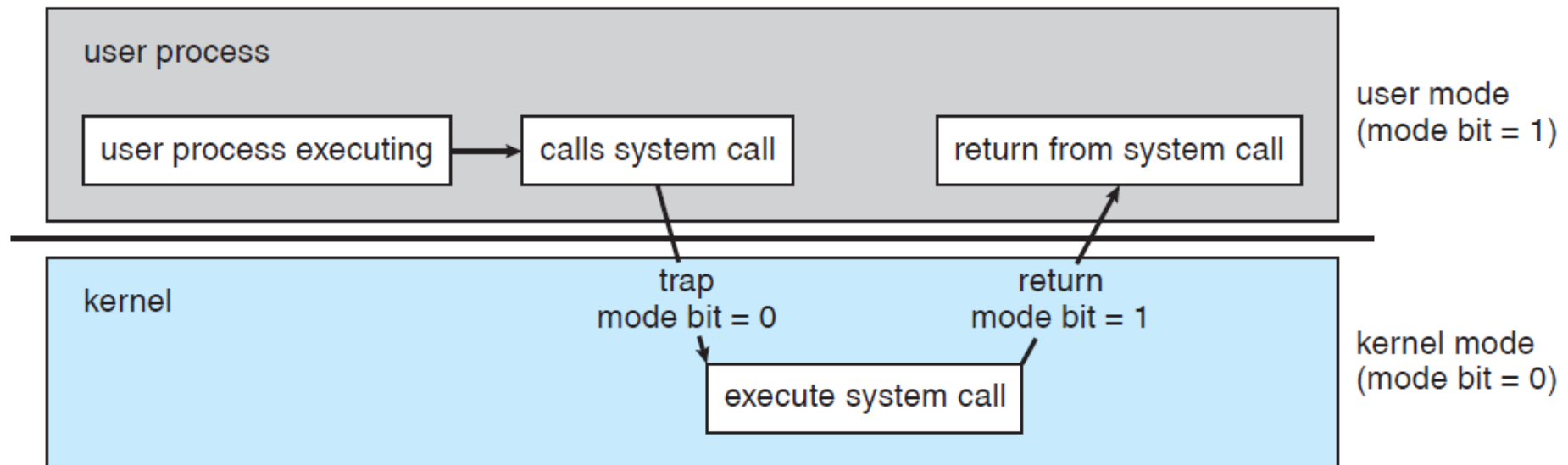
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
 - The user program that is up for execution on CPU generally uses library calls to issue system calls. The function of the library routine is to check the arguments that are provided by the application and then build a data structure to convey the arguments from the application to the kernel of the operating system and then execute special instructions which are called the trap or the software interrupts.
 - These special instructions or traps contain operands that help to identify exactly which kernel service the application arguments are demanding. So, when the process is made to execute the traps then the interrupt saves the state of the user code and then switches to the supervisor mode and then dispatches the right kernel routine that can implement the requested service.

OS Operations: Dual Mode Operation

- To ensure the proper execution of the operating system, it is important to distinguish between the execution of operating-system code and user-defined code.
- We need two separate modes of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).
- Most computer systems provide hardware support that allows us to differentiate among various modes of execution.
- A bit, **called the mode bit**, is added to the hardware of the computer to indicate the current mode: **kernel (0)** or **user (1)**.

OS Operations: Dual Mode Operation (contd..)

- With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.
- When the computer system is executing on behalf of a user application, the system is in user mode.
- When a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.



OS Operations: Dual Mode Operation (contd..)

- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0).
- The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.
- The dual mode of operation provides with the means for protecting the operating system from errant users.
- This protection is achieved by designating some of the machine instructions that may cause harm as **privileged instructions**.
- The hardware allows **privileged instructions** to be executed only in kernel mode.

OS Operations: Dual Mode Operation (contd..)

- Recent versions of the Intel CPU do provide dual-mode operation.
- Most contemporary operating systems - such as Windows, as well as Unix, Linux, and Solaris - take advantage of dual-mode feature and provide greater protection for the operating system.
- The lack of a hardware-supported dual mode can cause serious shortcomings in an operating system.
 - For instance, MS-DOS was written for the Intel 8088 architecture, which has no mode bit and therefore no dual mode.

OS Operations: Timer

- The operating system should maintains control over the CPU.
- It can not be allowed that a user program gets stuck in an infinite loop or to fail to call system services and never return control to the operating system.
- A timer can be set to interrupt the computer after a specified period.
- A timer is generally implemented by a fixed-rate clock and a counter.
- The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.
- Before turning over control to the user, the operating system ensures that the timer is set to interrupt.
 - If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.

- In earliest computers (from the late-1940s to the mid-1950s), the programmer interacted directly with the computer hardware.
 - There was no Operating System.
 - The computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
 - Programs in machine code were loaded via the input device (e.g., a card reader).
 - If an error halted the program, the error condition was indicated by the lights.
 - If the program proceeded to a normal completion, the output appeared on the printer.
 - These early systems presented two main problems:
 - Scheduling
 - Setup Time

Serial Processing (contd...)

- **Scheduling**

- Most installations used a hardcopy sign-up sheet to reserve computer time. A user could sign up for a block of time in multiples of a half hour or so. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.

- **Setup Time**

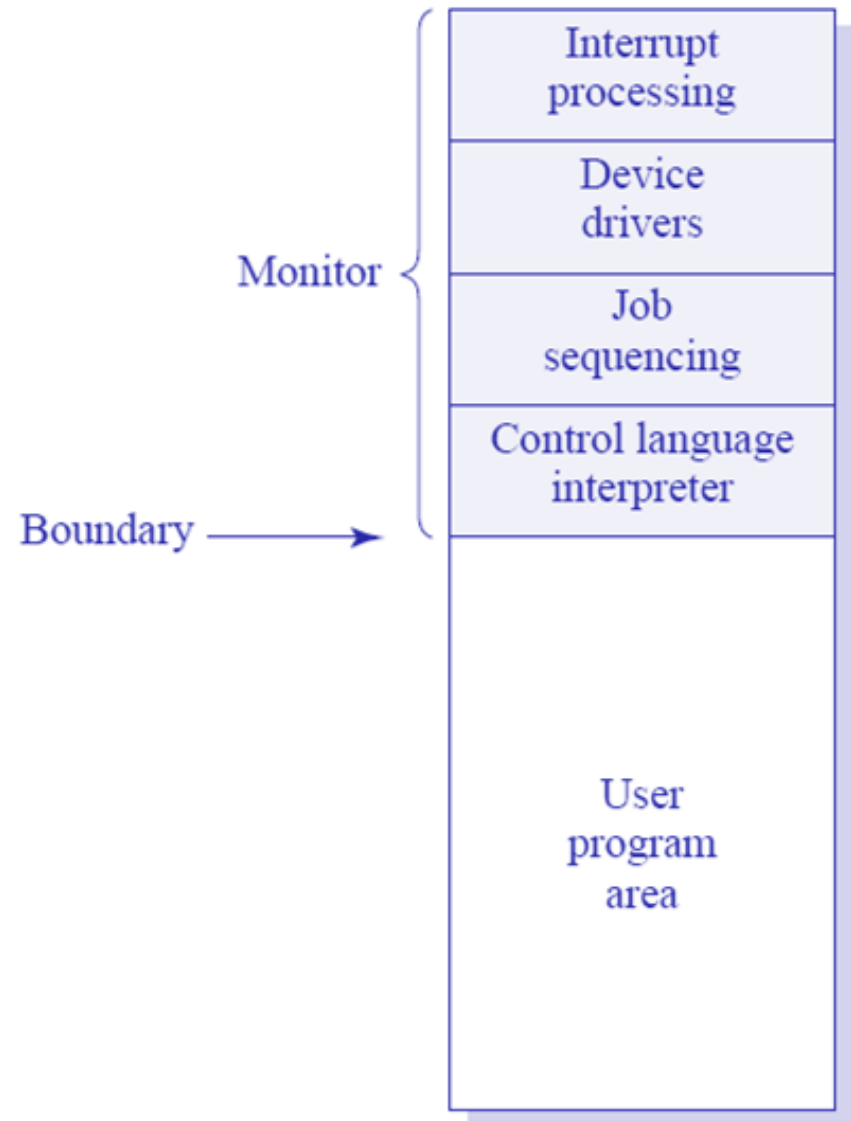
- A considerable amount of time was spent just in setting up the program to run. A single program, called a job, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions.
- This mode of operation is termed **Serial Processing**, reflecting the fact that users have access to the computer in series.

Simple Batch Systems

- **To improve utilization**, the concept of a **batch operating system** was developed.
- The first batch operating system (and the first OS of any kind) was developed in the mid-1950s by General Motors for use on an IBM 701.
- The main idea behind the simple batch-processing scheme was the use of a **software**, known as the **monitor**.
- **With this type of OS, the user no longer has direct access to the processor.**
 - The user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
 - Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.

Simple Batch Systems (contd...)

- The monitor reads in jobs one at a time from the input device.
- As it is read in, the current job is placed in the user program area, and control is passed to this job.
- When the job is completed, it returns control to the monitor, which immediately reads in the next job.
- The results of each job are sent to an output device (such as a printer).



Simple Batch Systems (contd...)

- The monitor performs a scheduling function.
 - A batch of jobs is queued up, and jobs are executed as rapidly as possible, with no intervening idle time.
- The monitor improves job setup time as well.
 - With each job, instructions are included in a primitive form of job control language (JCL).
 - JCL is a special type of programming language used to provide instructions to the monitor.

Simple Batch Systems (contd...)

- With a batch operating system, processor time alternates between execution of user programs and execution of the monitor.
- There have been two sacrifices:
 - Some main memory is given over to the monitor.
 - Some processor time is consumed by the monitor.
 - Both of these are forms of overhead.
 - Despite this overhead, the simple batch system improves utilization of the computer.

Multiprogrammed Batch Systems

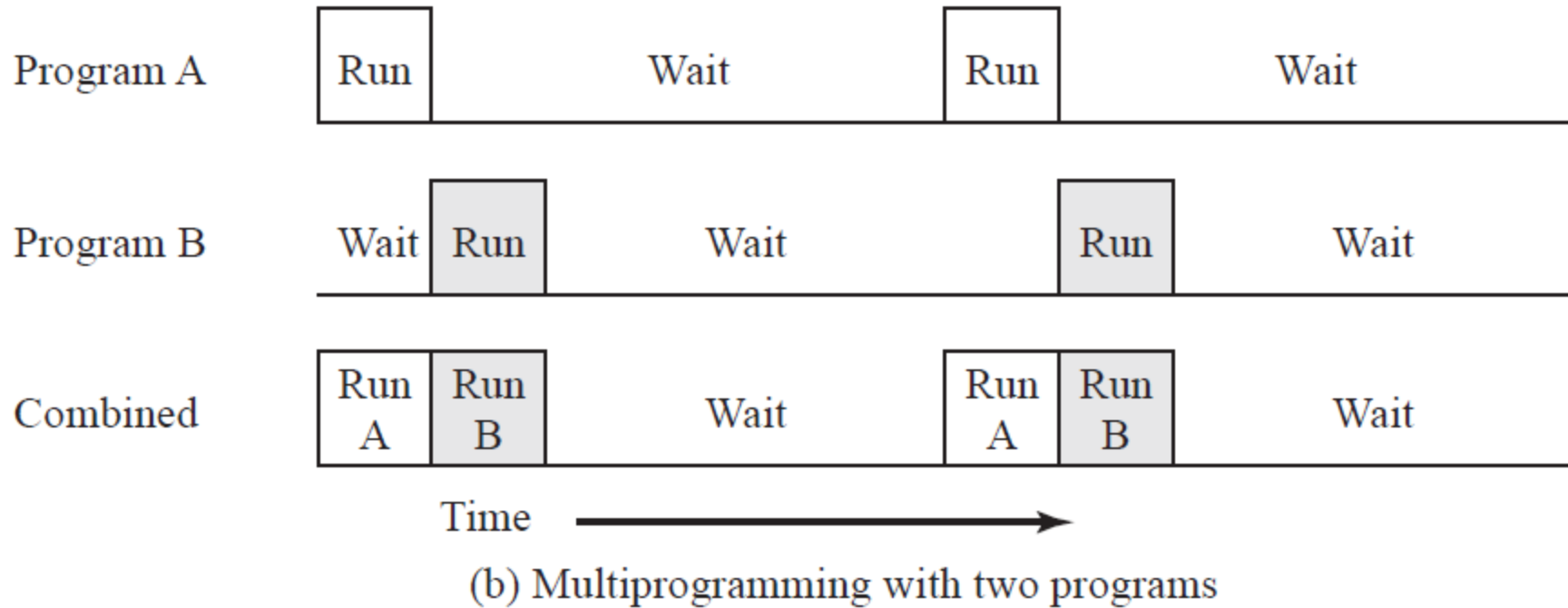
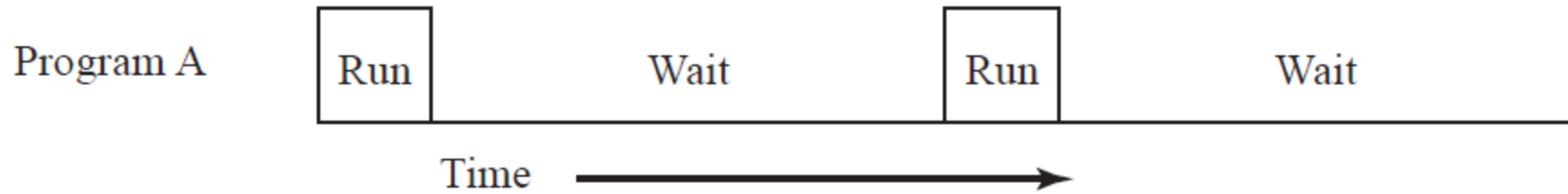
- Even with the automatic job sequencing provided by a simple batch operating system, **the processor is often idle.**
- The problem is that I/O devices are slow compared to the processor.

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	<u>31 μs</u>
Percent CPU Utilization = $\frac{1}{31} = 0.032 = 3.2\%$	

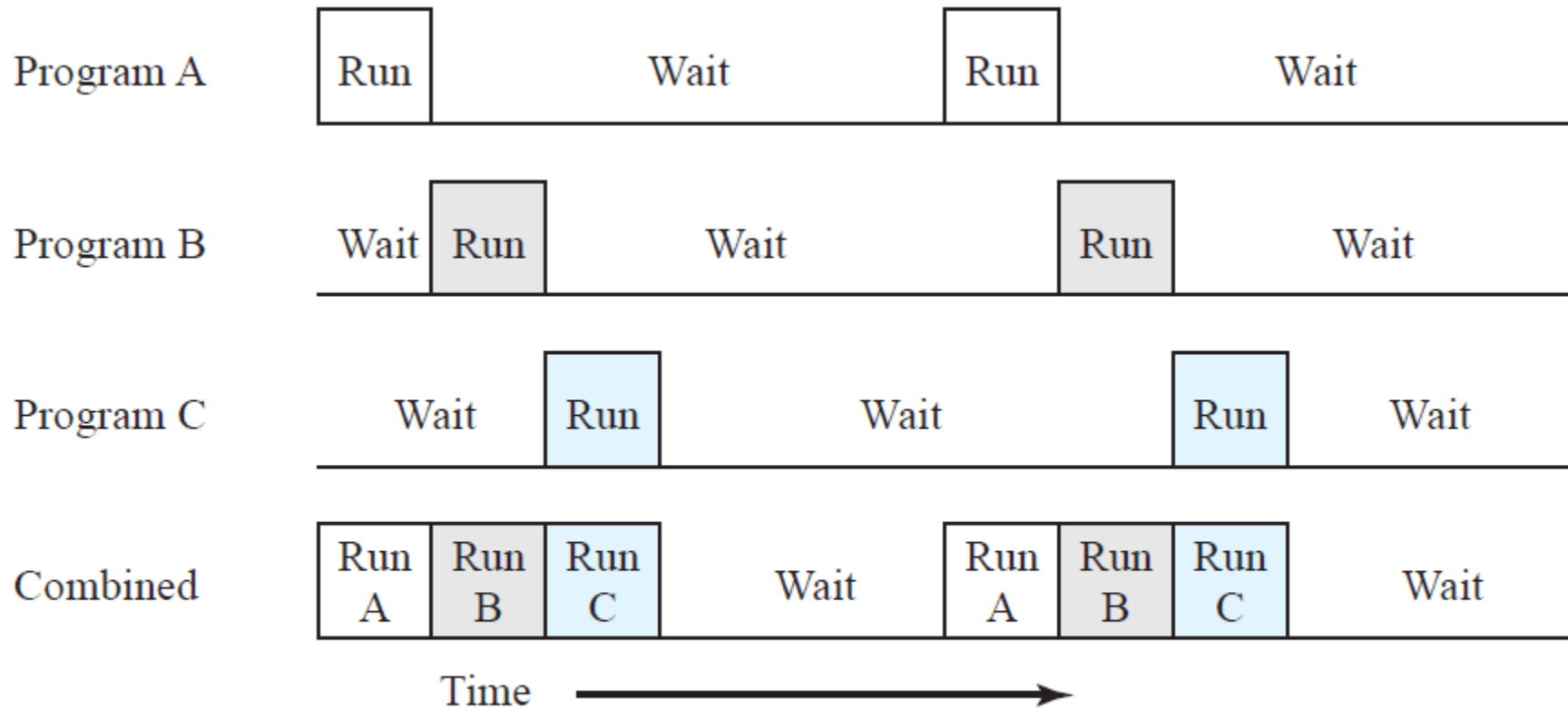
System Utilization Example

- In this example, the computer spends over 96% of its time waiting for I/O devices to finish transferring data to and from the file.

Multiprogrammed Batch Systems (contd...)



Multiprogrammed Batch Systems (contd...)



(c) Multiprogramming with three programs

Multiprogrammed Batch Systems (contd...)

- In multiprogrammed systems, several jobs are kept in main memory and they are executed concurrently, with the CPU switching rapidly between the jobs.
- Multiprogramming is the central theme of modern operating systems.
 - When several jobs are kept in main memory, **it requires some form of memory management.**
 - In addition, if several jobs are ready to run, the processor must decide which one to run, **this decision requires an algorithm for scheduling.**

Multiprogrammed Batch Systems (contd...)

- The most notable additional feature that is useful for multiprogramming is the hardware that supports I/O interrupts and DMA (direct memory access).
- With interrupt-driven I/O or DMA, the processor can issue an I/O command for one job and proceed with the execution of another job while the I/O is carried out by the device controller.
- When the I/O operation is complete, the processor is interrupted and control is passed to an interrupt-handling program in the OS.
- The OS will then pass control to another job.

Programmed I/O

- The problem with the Programmed I/O is that the processor has to wait a long time for the input/output module of concern to be ready for either reception or transmission of more data.
 - The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result the level of performance of entire system is degraded.
- An alternative approach for this is Interrupt-driven I/O.

Interrupt-driven I/O

- Interrupt driven I/O means CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.
 - The I/O module will then interrupt the processor to request service, when it is ready to exchange data with the processor.
 - The processor then executes the data transfer as before and then resumes its former processing.
- Interrupt-driven I/O still consumes a lot of time because every data has to pass with processor.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

Direct Memory Access (DMA)

- When large volumes of data are to be moved, a more efficient technique is required: **Direct Memory Access**.
- In DMA mechanism, **CPU grants I/O module authority to read from or write to memory without involvement**.
 - The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an I/O module.

Working of DMA

- When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending the following information:
 - Whether a read or write is requested.
 - The address of the I/O devices.
 - Starting location in memory to read from or write to.
 - The number of words to be read or written.
- The processor then continues with other work. It has delegated this I/O operation to the DMA module, and that module will take care of it.
 - The DMA module transfers the entire block of data, one word at time, directly to or from memory, without going through the processor.
 - When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus the processor is involved only at the beginning and at the end of the transfer.

Time-Sharing Systems

- With the use of multiprogramming, batch processing can be quite efficient.
- However, for many jobs, it is desirable to provide a mode in which the user interacts directly with the computer.
- Indeed, for some jobs, such as **transaction processing**, an interactive mode is essential.
- Today, the requirement for an interactive computing facility can be met by the use of a dedicated personal computer or workstation.
- The interactive computing facility was not available in the 1960s, when most computers were big and costly.
- Instead, time sharing was developed.

Time-Sharing Systems (contd...)

- When the concept of multiprogramming is used to handle **multiple interactive jobs**, it is referred to as time sharing.
 - Processor time is shared among multiple users.
 - Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short quantum of computation.
 - The main objective of time-sharing systems is **minimize response time**.
 - **Compatible Time-Sharing System (CTSS)** was one of the first time-sharing operating systems, developed at MIT in 1961 for the IBM 709.

Compatible Time-Sharing System (CTSS)

- The system ran on a computer with 32,000 words of main memory, with the monitor consuming 5000 of that.
- A system clock generated interrupts at a rate of approximately one every 0.2 seconds.
- At each clock interrupt, the OS regained control and could assign the processor to another user.
- This technique is known as time slicing.
- Thus, at regular time intervals, the current user would be preempted and another user loaded in.
- To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in.
- Subsequently, the old user program code and data were restored in main memory when that program was next given a turn.

Real-Time Embedded Systems

- Embedded computers are found everywhere - from car engines and manufacturing robots to microwave ovens.
- Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.
- These embedded systems vary considerably.
 - Some are general-purpose computers, running standard operating systems—such as UNIX—with special-purpose applications to implement the functionality.
 - Others are hardware devices with a special-purpose embedded operating system providing just the functionality desired.
 - Yet others are hardware devices with application-specific integrated circuits (ASICs) that perform their tasks without an operating system.

Real-Time Embedded Systems (contd...)

- Embedded systems almost always run real-time operating systems.
- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application.
- Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.

Real-Time Embedded Systems (contd...)

- A real-time system has well-defined, fixed time constraints.
 - Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building.
- A real-time system functions correctly only if it returns the correct result within its time constraints.
- Contrast this system with a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or a batch system, which may have no time constraints at all.

Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide the users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Distributed systems depend on networking for their functionality.
- Networks are characterized based on the distances between their nodes.
 - SAN (Small-Area Network)
 - LAN (Local-Area Network)
 - MAN (Metropolitan-Area Network)
 - WAN (Wide-Area Network)

Distributed Operating Systems (contd...)

- Various inter-connected computers communicate each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
 - These are referred as **loosely coupled systems** or distributed systems.
 - The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.
 - The different systems communicate closely enough to provide the illusion that only a single operating system controls the network.

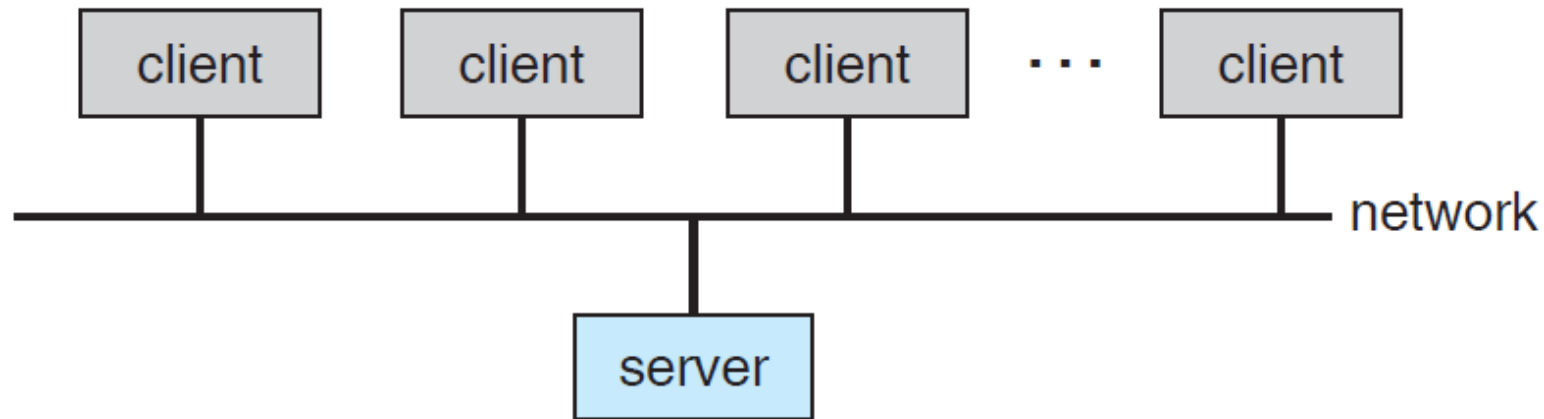


Computing Environments

- Client–Server Computing
- Peer–to–Peer Computing

Client–Server Computing

- It is a form of specialized distributed system.



- Many of today's systems act as **server systems** to satisfy requests generated by client systems.
- Server systems can be broadly categorized as:
 - Compute Servers, and
 - File Servers

Compute-Server System

- It provides an interface to which a client can send a request to perform an action (for example, read data, encrypt the plain text, etc.).
 - In response, the server executes the action and sends back results to the client.
 - A server running a database (program) that responds to client requests for accessing data is an example of such a system.
 - A server running a container (program) that responds to client requests for performing computation (business logic) is also an example of compute-server system.

File-Server System

- It provides a file-system interface where clients can create, update, read, and delete files.
- File servers may be categorized by the method of access:
 - **Internet File Servers**
 - An example of such a system is a Web server that delivers files to clients running Web browsers.
 - Internet file servers are frequently accessed by [FTP](#) or by [HTTP](#).
 - **LAN Servers**
 - Any computer can be configured to be a host and act as a file server.
 - A file server might be a dedicated network-attached storage (NAS) device that also serves as a remote hard disk drive for other computers.
 - LAN servers are accessed by [SMB \(Server Message Block\)](#) – used in [Windows and Mac](#) and [NFS \(Network File System\)](#) – used in [Unix-like Systems](#) protocol.

Peer-to-Peer (P2P) Computing

- It is another structure for a distributed system.
- In this model, clients and servers are not distinguished from one another.
- All nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.
- In P2P system, services can be provided by several nodes distributed throughout the network.
- To participate in a P2P system, a node must first join the network of peers.
- Once a node has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.

- Approach - 1

- When a node joins a network, it registers its service with a centralized lookup service on the network.
- Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service.
- The remainder of the communication takes place between the client and the service provider.

- Approach - 2

- A peer acting as a client first discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network.
- The node (or nodes) providing that service responds to the peer making the request.

System Call

- The system call is the means by which a process requests a specific kernel service.
 - System calls provide an interface to the services made available by an operating system.
 - System calls are generally available as routines written in C and C++.
 - Certain low-level tasks (for example, tasks where hardware must be accessed directly) may need to be written using assembly-language instructions.

How System Calls are Used? - Example

A simple program to read data from one file and copy them 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

In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define the two files.

Once the two file names are obtained, the program must open the input file and create the output file. Each of these operations requires another system call.

There are also possible error conditions for each operation. When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another sequence of system calls) and then terminate abnormally (another system call).

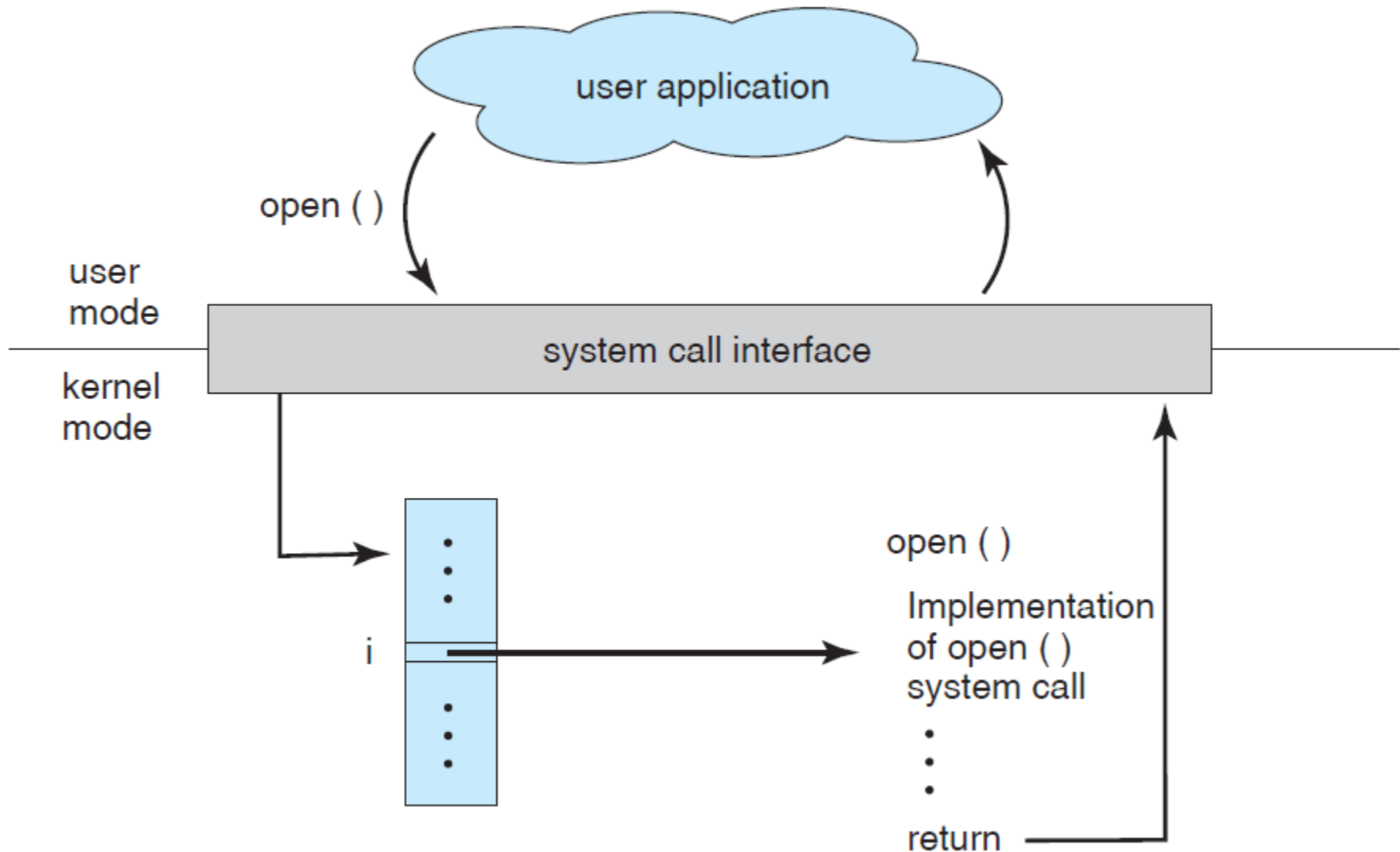
If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (another system call).

Frequently, systems execute thousands of system calls per second.

How System Calls are Used? (contd..)

- Most programmers never focus on detailed working levels of System Calls.
 - Typically, application developers design the programs according to an Application Programming Interface (API).
 - The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
 - Three of the most common APIs available to application programmers are the Win32 API for Windows systems, the POSIX (Portable Operating System Interface) API for POSIX-based systems (all versions of UNIX, Linux, and MacOS X), and the Java API for designing programs that run on the Java virtual machine.
- Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer.
- For example, the Win32 function `CreateProcess()` actually calls the `NTCreateProcess()` system call in the Windows kernel.

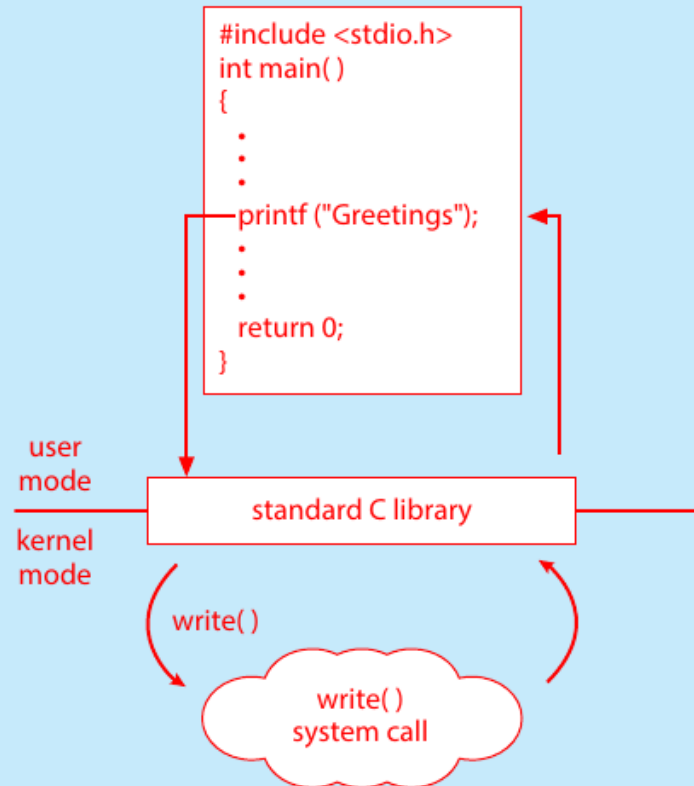
How System Calls are Used? (contd..)



How System Calls are Used? (contd..)

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



Why Programmers prefer API rather than System Calls

- Program Portability
 - An application programmer designing a program using an API can expect his program to compile and run on any system that supports the same API.
- Difficult to Work with System Calls
 - Actual system calls can often be more detailed and difficult to work with than the API available to an application programmer.
- System Call Interface in Programming Languages
 - The support system (a set of functions built into libraries included with a compiler) for most programming languages provides a system-call interface that serves as the link to system calls made available by the operating system.
 - The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.



Types of System Calls

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications
- Protection

Types of System Calls

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

- A process is a program in execution.
- A process is the unit of work in systems.
- Systems consist of a collection of processes:
 - **Operating-system processes** execute system code, and
 - **User processes** execute user code
 - All these processes may execute concurrently.
- A process need certain resources - such as CPU time, memory, files, and I/O devices - to accomplish its task.

Process (contd...)

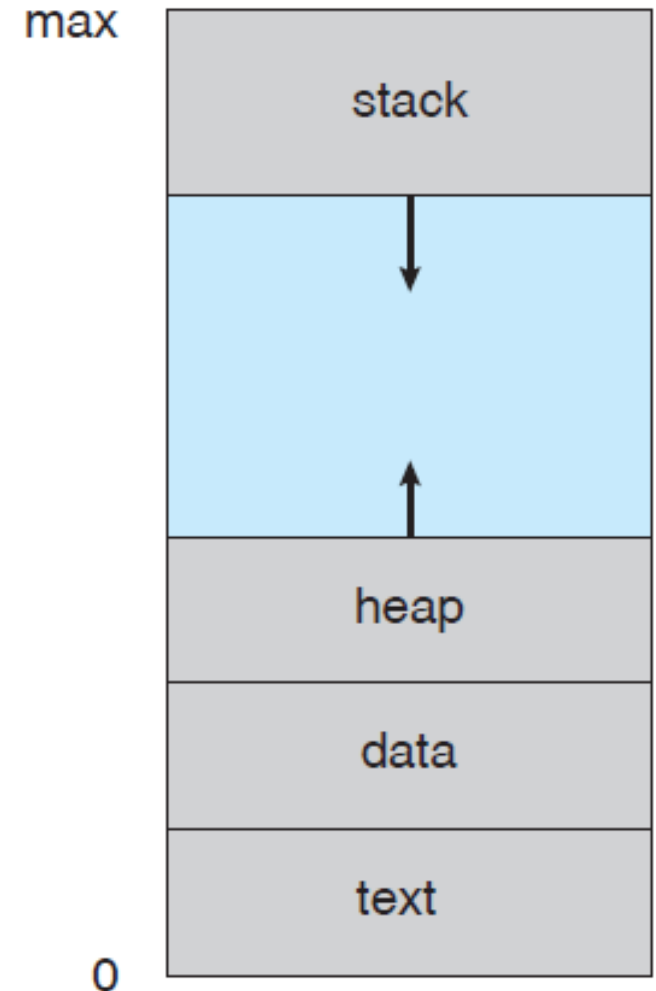
- Modern operating systems support processes that have multiple threads.
- The operating system is responsible for the following activities in connection with process and thread management:
 - The **creation** and **deletion** of both user and system processes;
 - The **scheduling** of processes; and
 - The provision of mechanisms for **synchronization**, **communication**, and **deadlock handling** for processes.

Process Concept

- What to call all the CPU activities?
 - A batch system executes **jobs**;
 - Time-shared system has **user programs**, or **tasks**.
 - The terms **job** and **process** are used almost interchangeably.
- Much of operating-system theory and terminology was developed during a time when the major activity of operating systems was **job processing**.

Process in Memory

- Informally, a process is a program in execution.
- However, a process is more than the program code. It includes:
 - Text Section (Program Code)
 - Stack Section (Temporary Data such as Function Parameters, Return Addresses, and Local Variables)
 - Data Section (Global Variables)
 - Heap (Memory that is Dynamically Allocated during Process Run Time)
- Process also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.



Process vs. Program

- A program by itself is not a process.
- A program is a **passive entity**, such as a file containing a list of instructions stored on disk (often called an executable file)
- A process is an **active entity**, with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.
- Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog.exe or a.out.)

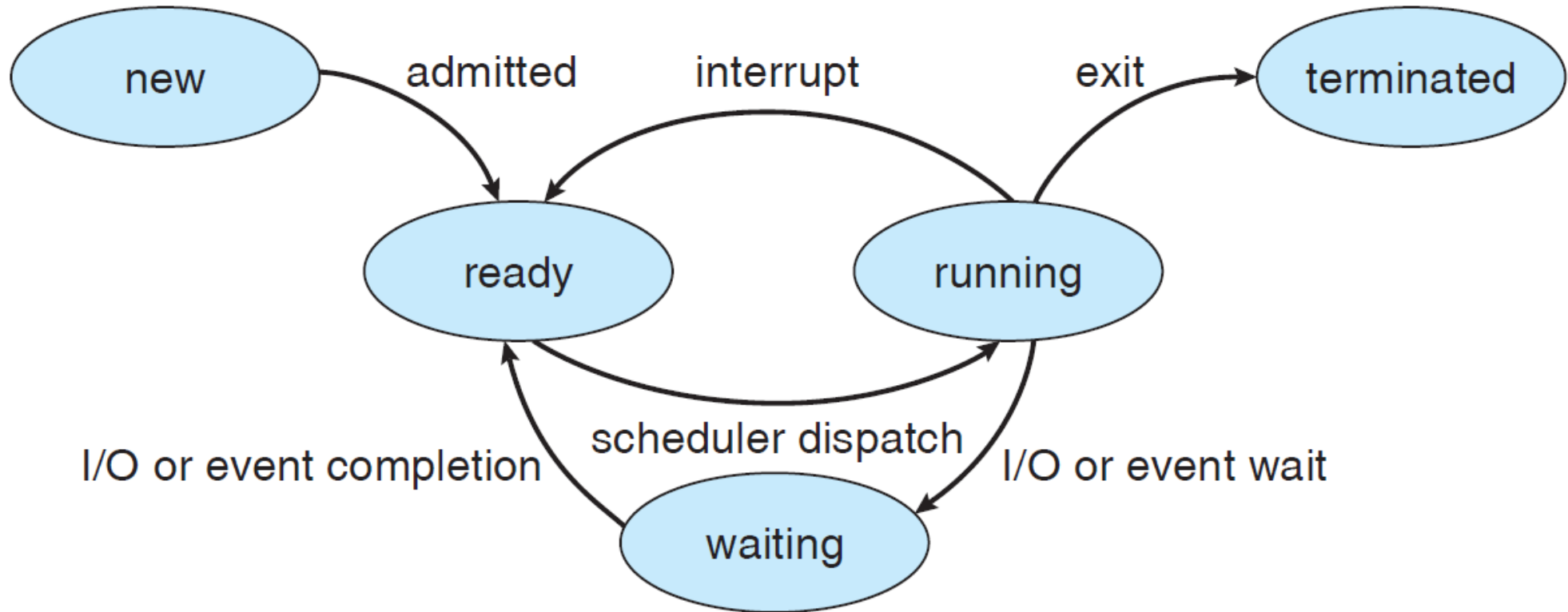
Process Concept (contd...)

- Although two processes may be associated with the same program, they are considered two separate execution sequences.
 - The same user may invoke many copies of the Web Browser program.
 - Each of these is a separate process.
 - Although the text sections are equivalent, the data, heap, and stack sections vary.

Process State

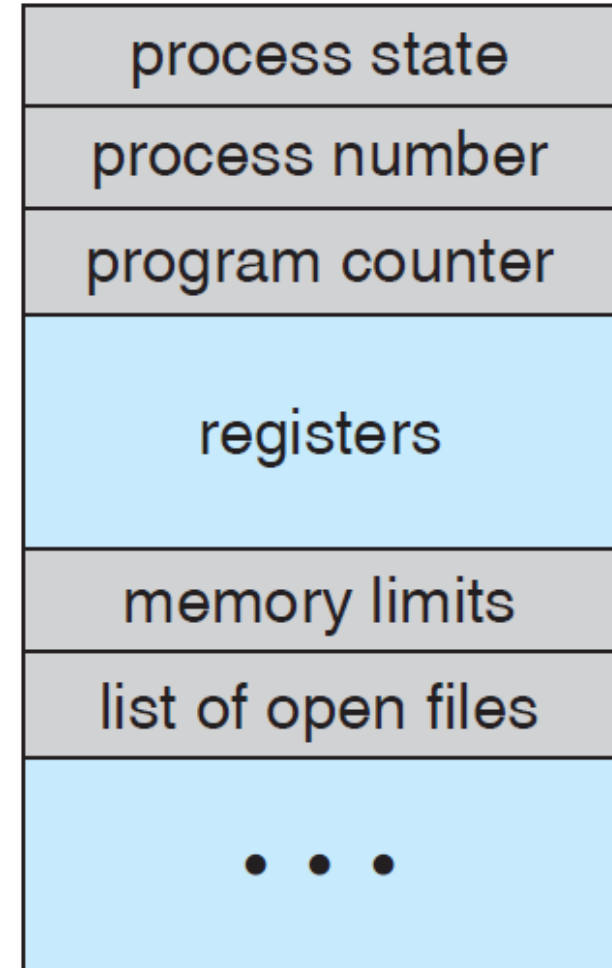
- As a process executes, it changes state.
- Each process may be in one of the following states:
 - **New** - The process is being created.
 - **Running** - Instructions are being executed.
 - **Waiting** - The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
 - **Ready** - The process is waiting to be assigned to a processor.
 - **Terminated** - The process has finished execution.

Process State (contd...)



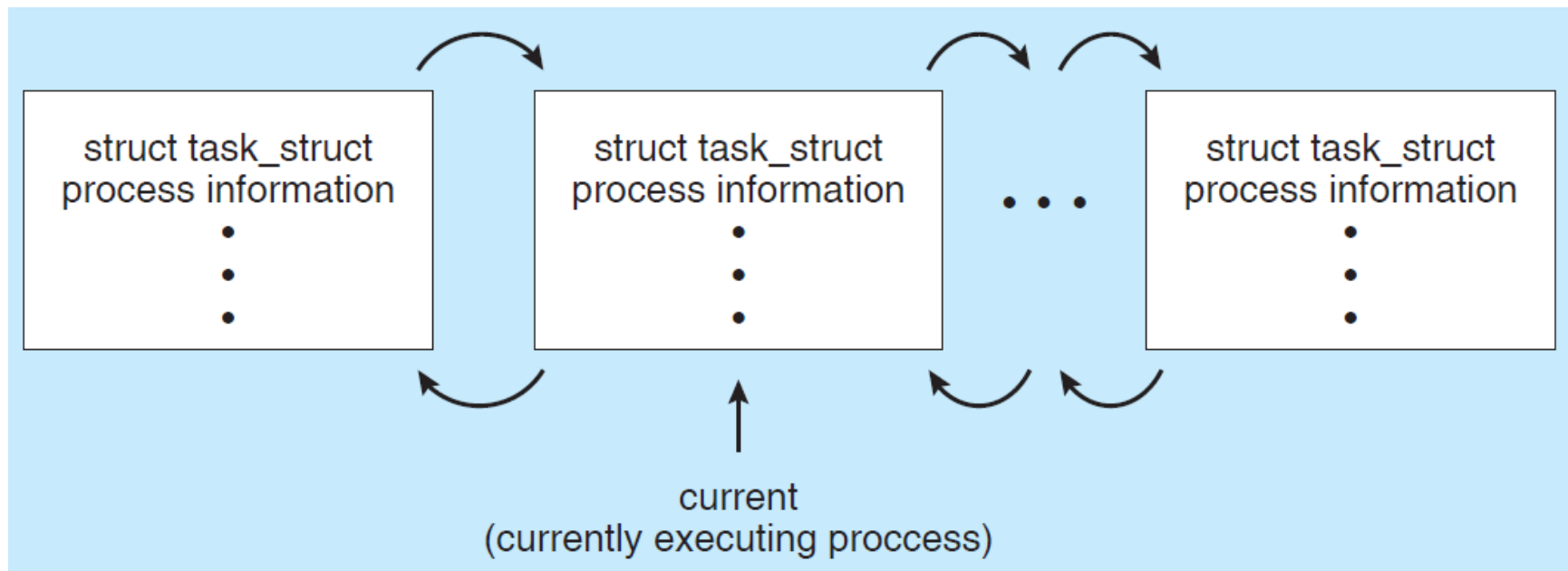
Process Control Block

- Each process is represented in the operating system by a Process Control Block (PCB) – A Data Structure.
- PCB contains many information, including:
 - Process State (The state may be new, ready, running, etc.)
 - Program Counter (The counter indicates the address of the next instruction to be executed for this process.)
 - CPU Registers (The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, etc.)
 - CPU-scheduling Information
 - Accounting Information (This information includes amount of CPU and real time used, job/process numbers, etc.)
 - I/O Status Information (This information includes the list of I/O devices allocated to the process, a list of open files, etc.)



Process Representation

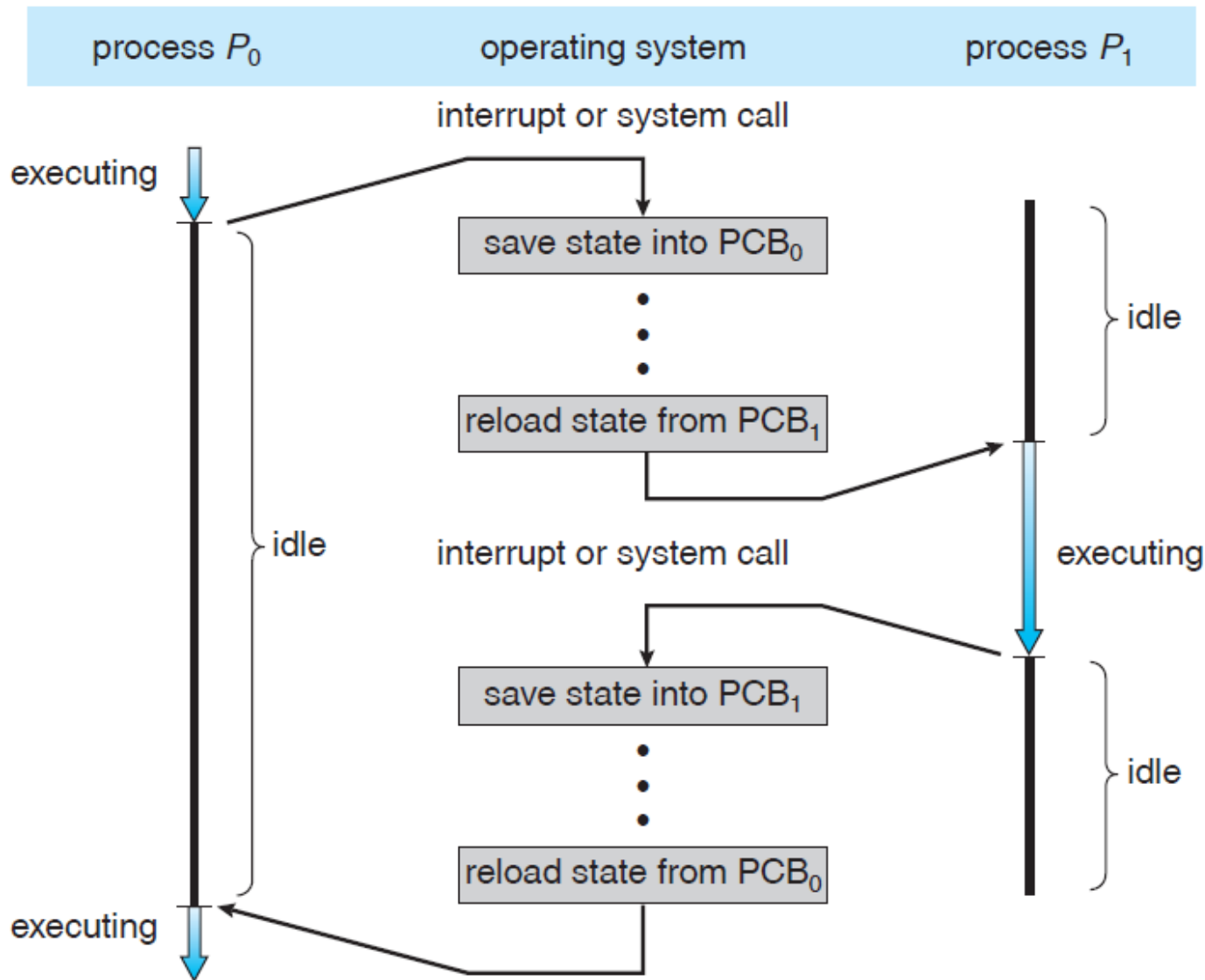
- The process control block in the **Linux** operating system is represented by the **C structure `task_struct`**.
- All active processes are represented using a **doubly linked list** of `task_struct`, and the kernel maintains a pointer - **current** - to the process currently executing on the system.



Process Scheduling

- The objective of **multiprogramming** is **to have some process running at all times**, to maximize CPU utilization.
- The objective of **time sharing** is to **switch the CPU among processes so frequently** that users can interact with each program while it is running.
- **To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for execution on the CPU.**
- For a single-processor system, there will never be more than one running process.
- If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

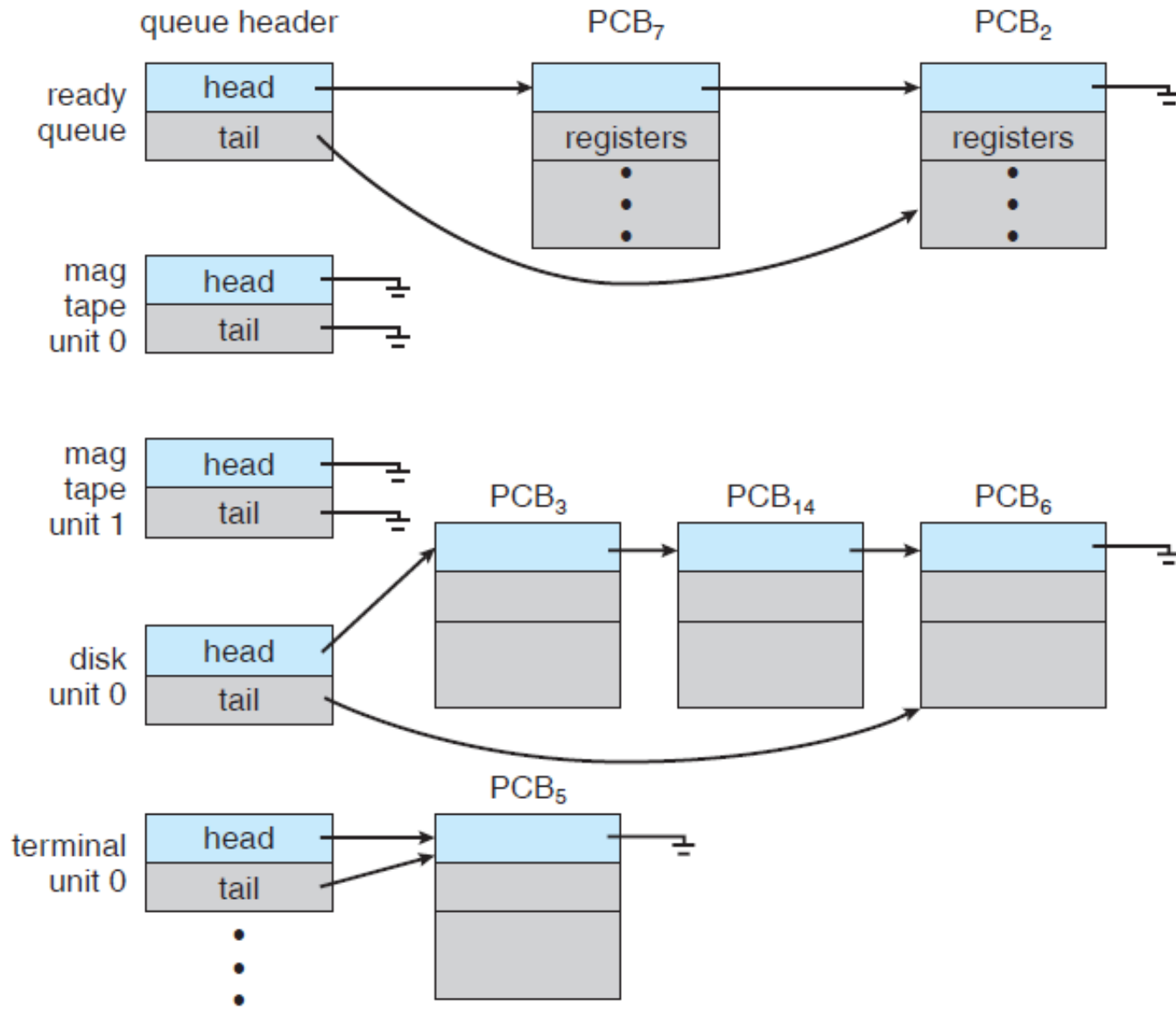
CPU Switching from Process to Process



Scheduling Queues

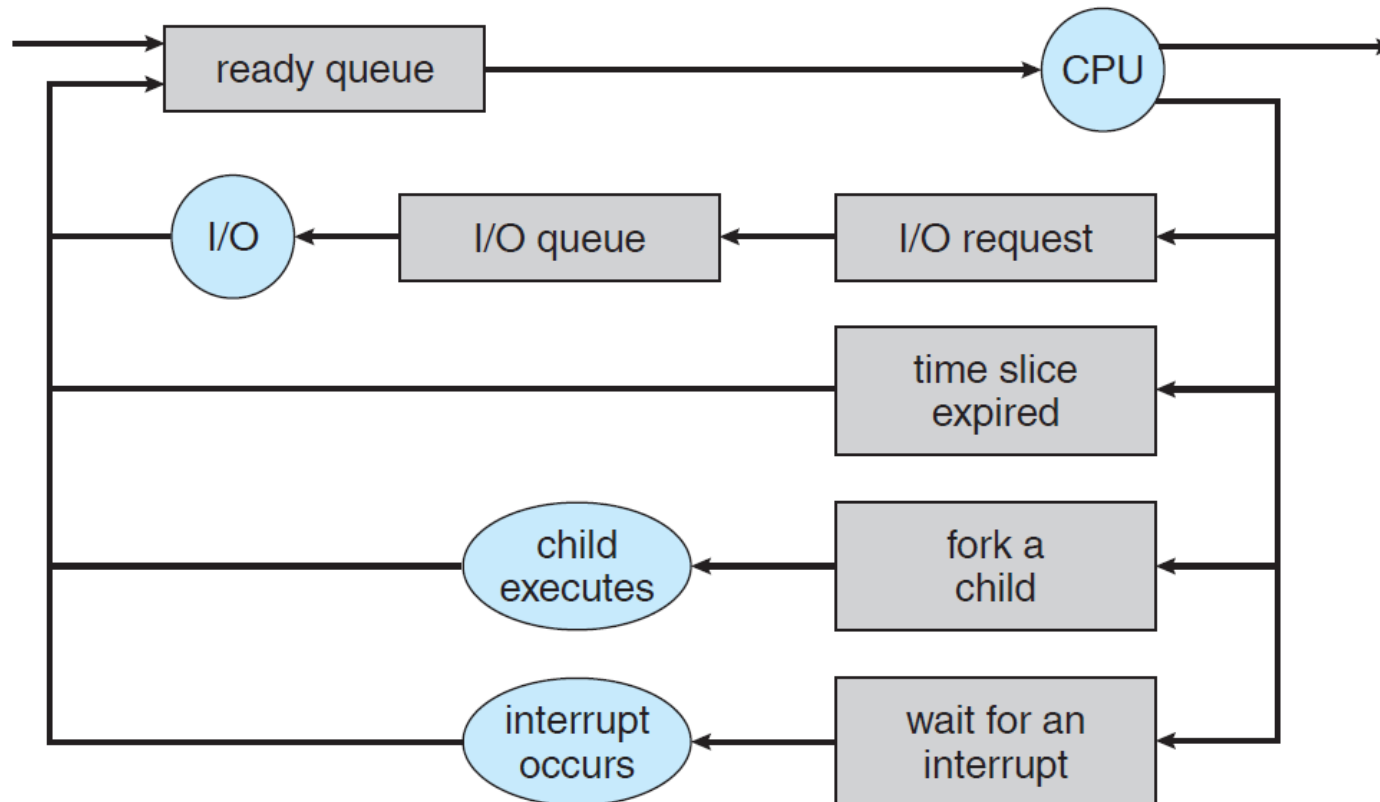
- As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
 - This queue is generally stored as a linked list.
 - A ready-queue header contains pointers to the first and final PCBs in the list.
 - Each PCB includes a pointer field that points to the next PCB in the ready queue.
- The process may have to wait for shared device, such as a disk. The list of processes waiting for a particular I/O device is called a **device queue**.
 - Each device has its own device queue.

Ready Queue and various I/O Device Queues



Process Scheduling: Queuing Diagram

- Two types of queues are present: the **ready queue** and a **set of device queues**.
- The circles represent the resources that serve the queues.
- The arrows indicate the flow of processes in the system.



Schedulers

- A process migrates among the various scheduling queues throughout its lifetime.
- The operating system selects processes from these queues with appropriate scheduler.
- Often, in a **batch system**, more processes are submitted than can be executed immediately.
 - These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution.
 - The **long-term scheduler**, or **job scheduler**, selects processes from this pool and loads them into memory for execution.
 - The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute and allocates the CPU to one of them.

Short-term Scheduler

- The short-term scheduler must select a new process for the CPU frequently.
- Often, the short-term scheduler executes at least once every 100 milliseconds.
- Because of the short time between executions, the short-term scheduler must be fast.
 - If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then $10/(100 + 10) = 9$ percent of the CPU is being used (wasted) simply for scheduling the work.
 - Short term scheduler controls **multitasking**.

Long-term Scheduler

- The long-term scheduler executes much less frequently.
 - Minutes may separate the creation of one new process and the next.
- The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).
 - If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
 - Thus, the long-term scheduler may need to be invoked only when a process leaves the system.
 - Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.

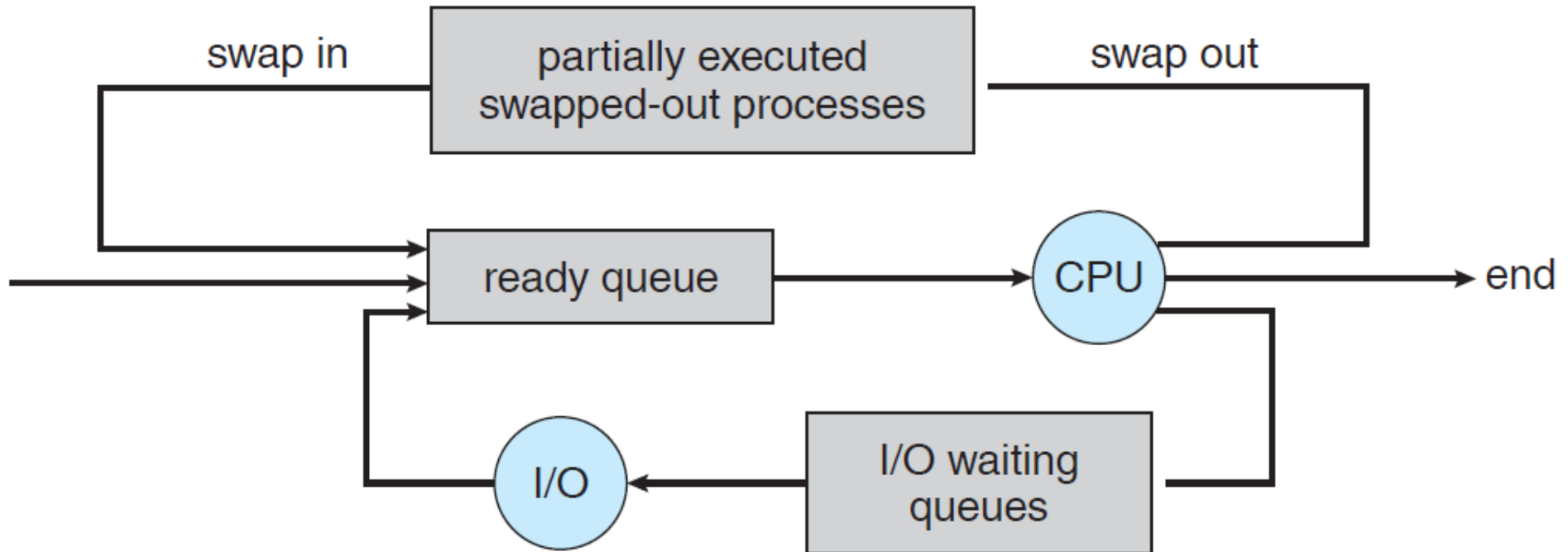
Long-term Scheduler (contd...)

- Most processes can be described as either **I/O bound** or **CPU bound**.
 - An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
 - A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.
 - If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.
 - If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
 - The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.

Medium-term Scheduler

- Some **time-sharing systems** may introduce an additional scheduler (**medium-term scheduler**).
- Sometimes it can be advantageous to **remove processes from memory**, and thus **reduce the degree of multiprogramming**.
- Later, the **process can be reintroduced into memory**, and its execution can be continued where it left off.
- **This scheme is called swapping.**
- The process is swapped out, and is later swapped in, by the medium-term scheduler.
- Swapping may be necessary to improve the **process mix** or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

Medium-term Scheduler



Addition of medium-term scheduling to the queuing diagram

Context Switch

- When an interrupt occurs, the system needs to **save the current context of the process running on the CPU** so that it can restore that context when its processing is done.
- **Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.**
- This task is known as a context switch.
- The context is represented in the PCB of the process.
- **Context-switch time is pure overhead.**
- Context-switch times are highly dependent on hardware support.

Operations on Processes: Creation

- A process may create several new processes during the course of execution.
 - The creating process is called a **parent process**.
 - The new processes are called the **children of that process**.
 - Each of these new processes may in turn create other processes, forming a **tree of processes**.
- Most operating systems identify processes according to a **unique process identifier** (or **pid**), which is typically an integer number.
- When a process creates a new process, two possibilities exist for execution:
 - The **parent continues to execute concurrently with its children**.
 - The **parent waits until some or all of its children have terminated**.

Process Creation (contd...)

- There are also two possibilities for the address space of the new process:
 - The child process is a duplicate of the parent process (it has the same program and data as the parent).
 - The child process has a new program loaded into it.
- In **UNIX**, using **fork() system call**, the new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process.
- In **WINDOWS**, using **spawn() system call**, a specified program is loaded into the address space of the child process at process creation.
 - **UNIX systems implement this as a second step, using exec() system call.**

Operations on Processes: Termination

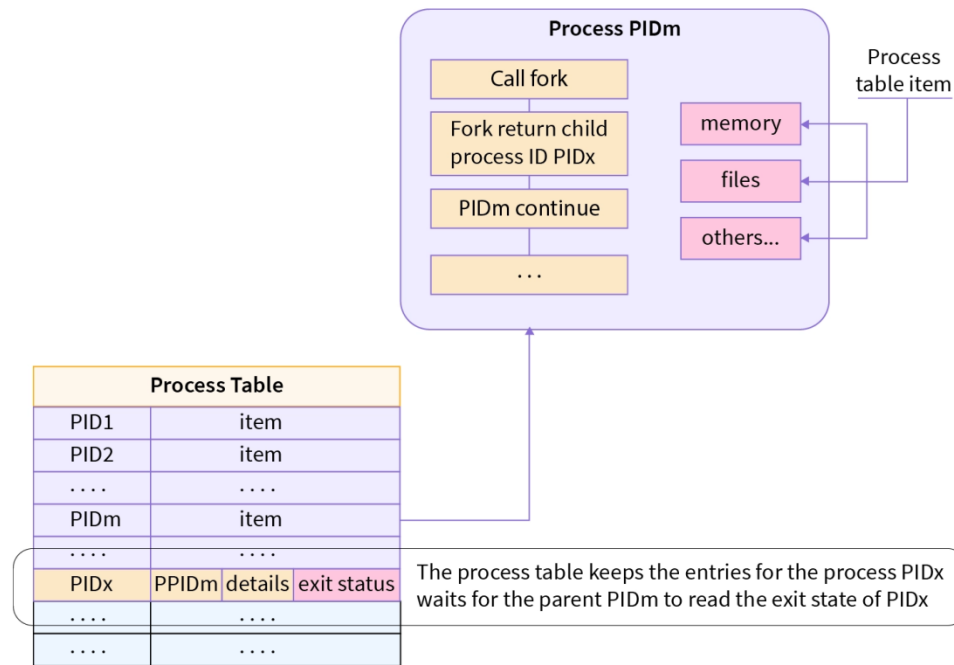
- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using appropriate system call.
- At that point, the process may return a status value (typically an integer) to its parent process.
- All the resources of the process - including physical and virtual memory, open files, and I/O buffers - are deallocated by the operating system.
- Termination can occur in other circumstances as well.
 - A process can cause the termination of another process via an appropriate system call.
 - Usually, such a system call can be invoked only by the parent of the process that is to be terminated.

Process Termination (contd...)

- A parent may terminate the execution of one of its children for a variety of reasons:
 - The child has exceeded its usage of some of the resources that it has been allocated.
 - The task assigned to the child is no longer required.
 - The operating system does not allow a child to continue if its parent terminates.
- Some systems do not allow a child to exist if its parent has terminated.
 - In such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated.
- In UNIX, if the parent terminates, however, all its children have assigned as their new parent the **init** process. Thus, the children still have a parent to collect their status and execution statistics.

Zombie Process in Operating System

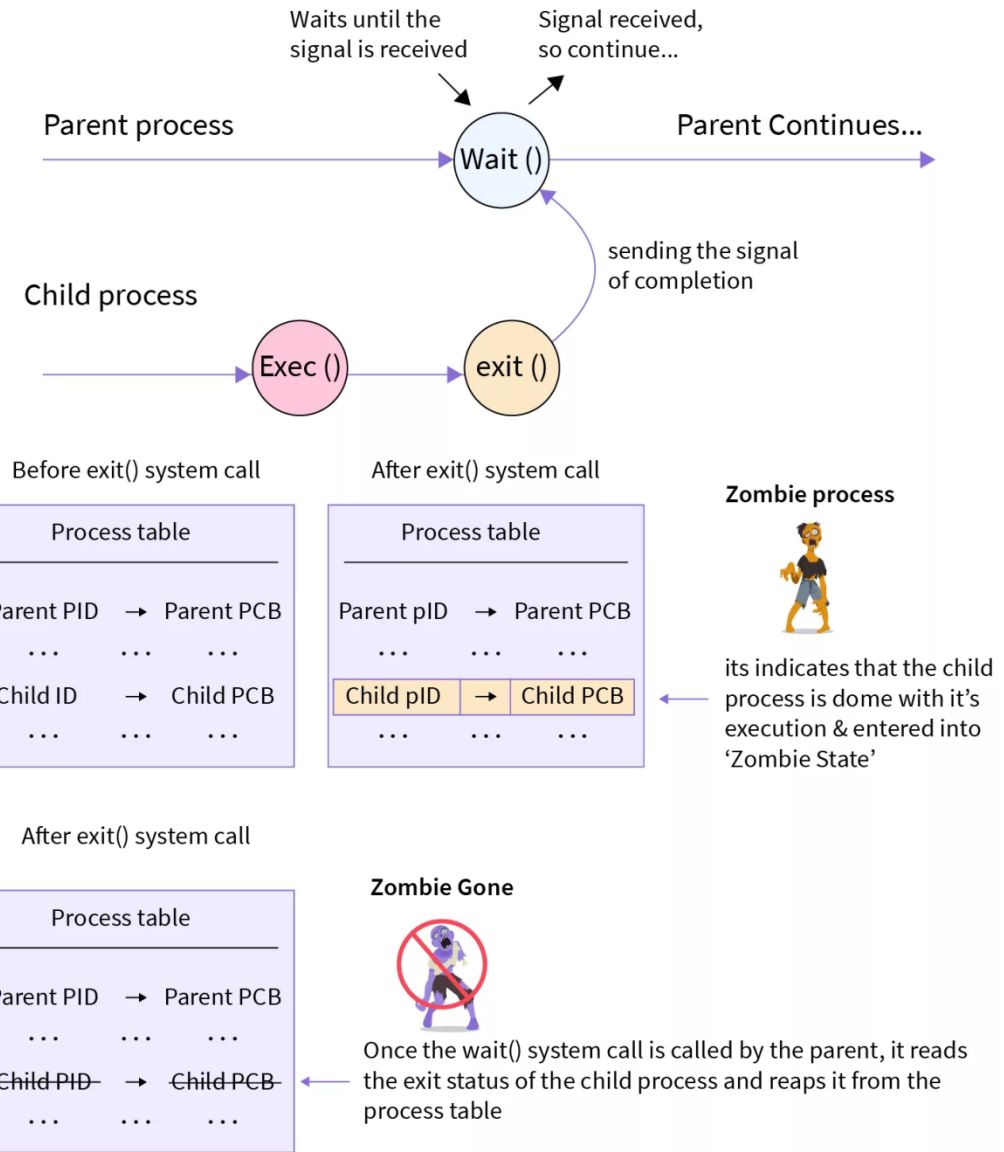
- Zombie process is also known as "dead" process.
 - Ideally when a process completes its execution, its entry from the process table should be removed but this does not happen in case of zombie process.



- **Analogy:** Zombie, mythological, is a dead person revived physically. Similarly, a zombie process in Operating System is a dead process (completed its execution) but is still revived (its entry is present in memory).

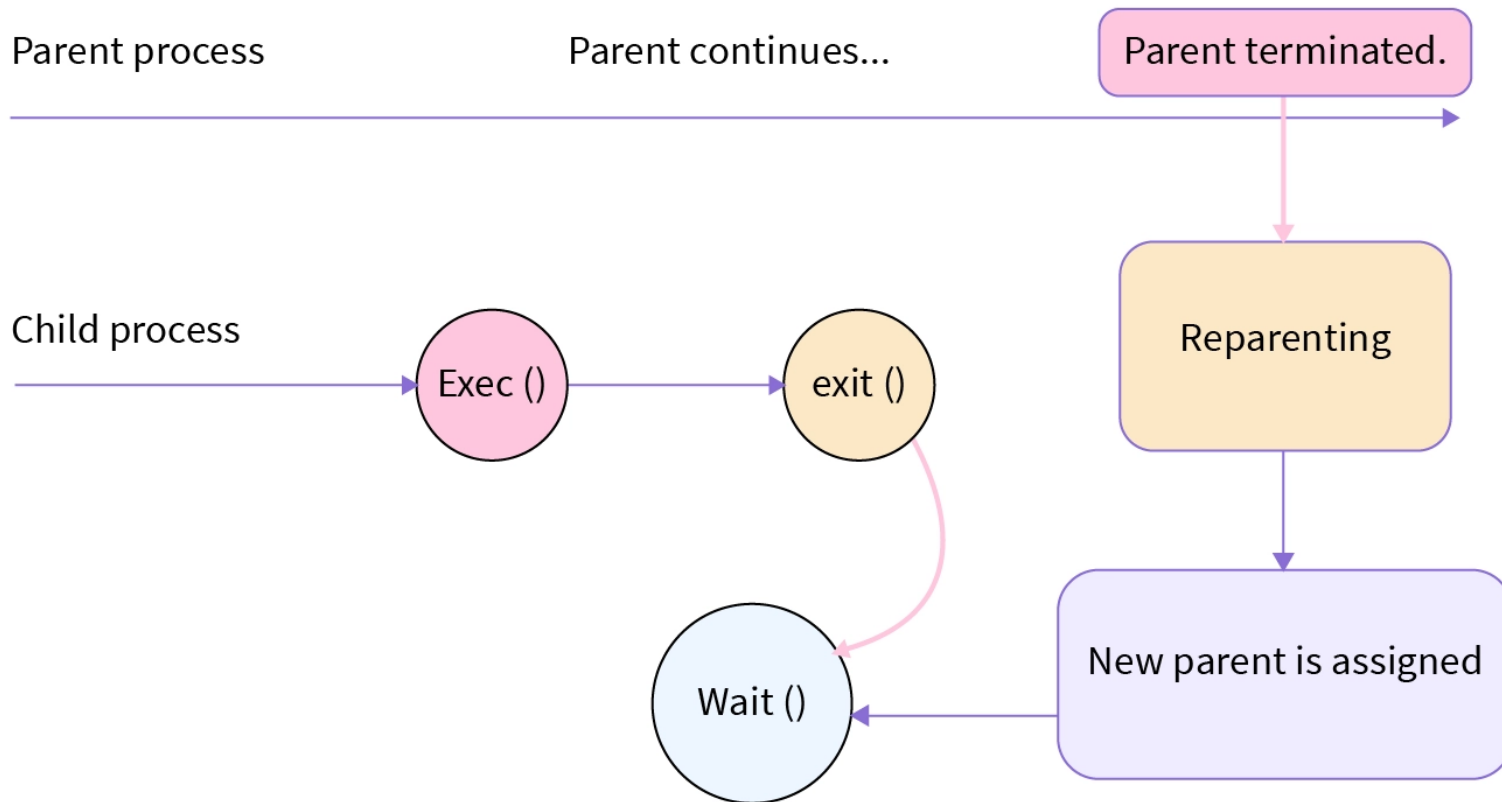
What happens with the Zombie Process

- **wait()** system call is used for removal of zombie processes.
 - **wait()** call ensures that the parent doesn't execute or sits idle till the child process is completed.



Orphan Process in Operating System

- A process which is executing (is alive) but **it's parent process has terminated** (dead) is called an **orphan process**.



Interprocess Communication

- A **process is independent** if it cannot affect or be affected by the other processes executing in the system.
 - Any process that does not share data with any other process is independent.
- A **process is cooperating** if it can affect or be affected by the other processes executing in the system.
 - Any process that shares data with other processes is a cooperating process.

Reasons for Cooperation

- Information Sharing

- Several users may be interested in the same piece of information (for instance, a shared file)
- We must provide an environment to allow concurrent access to such information.

- Computation Speedup

- We may break a task into subtasks, each of which will be executing in parallel with the others.
- Such a speedup can be achieved only with multiple processing elements.

- Modularity

- Dividing the system functions into separate processes or threads.

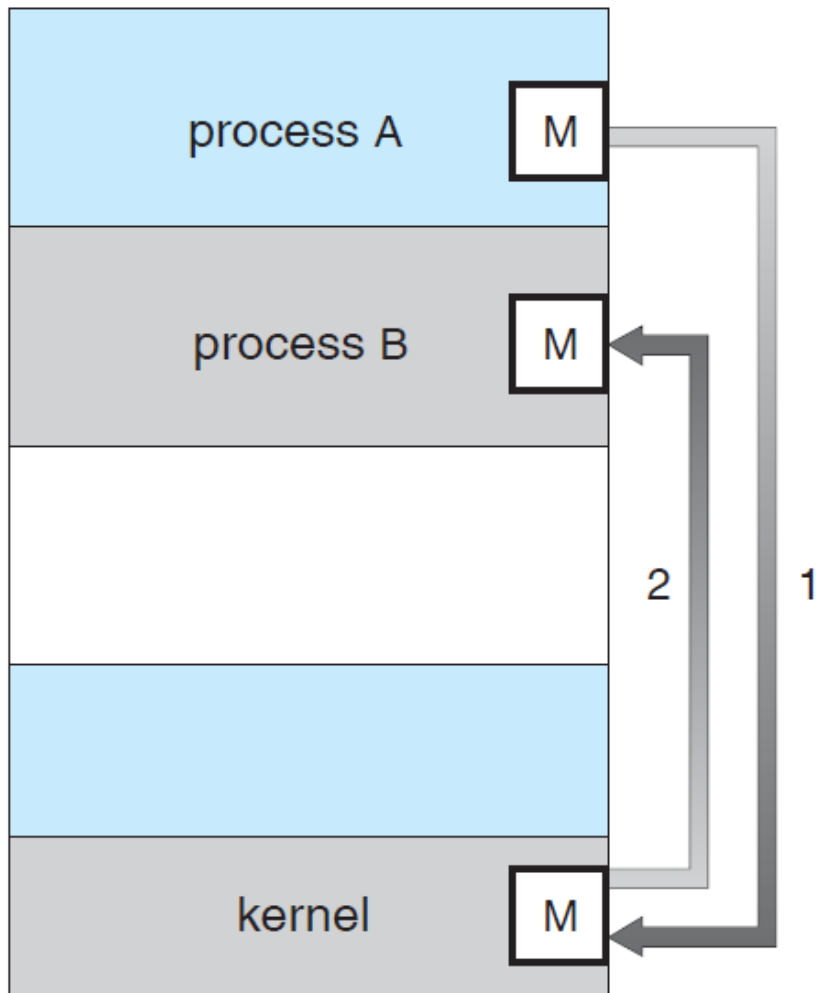
- Convenience

- An individual user may work on many tasks at the same time.

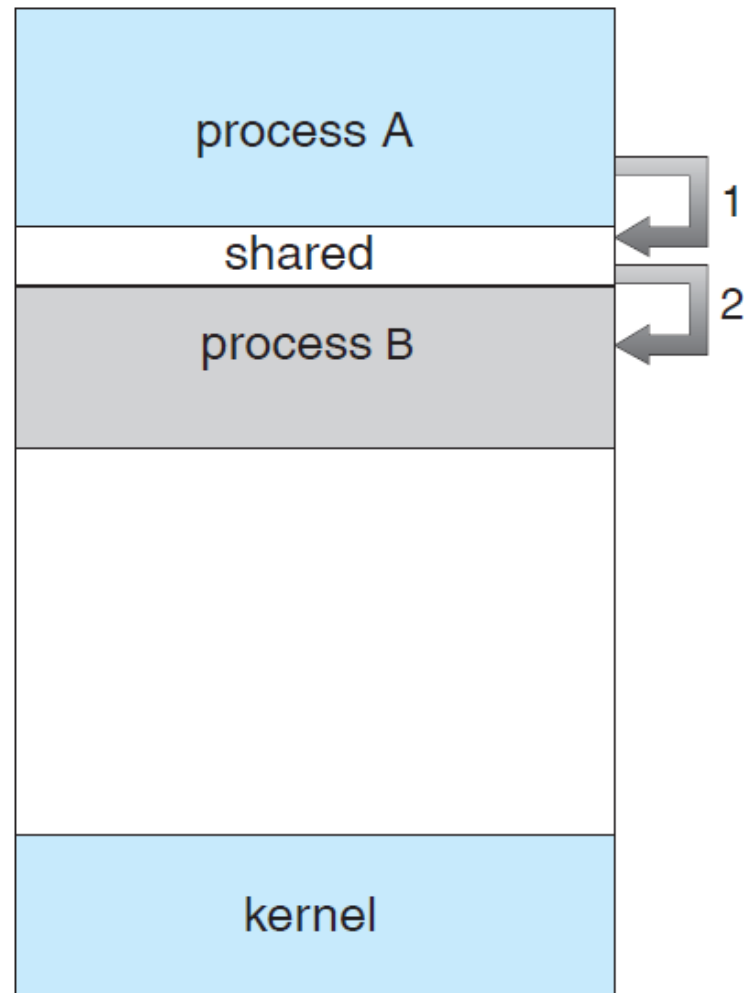
Interprocess Communication (contd...)

- Cooperating processes require an Interprocess Communication (IPC) mechanism.
 - IPC allows cooperating processes to exchange data and information.
- Models of Interprocess Communication:
 - Shared Memory
 - Message Passing

Models of Interprocess Communication



Message Passing



Shared Memory

Shared-Memory System

- Communicating processes establish a region of shared memory.
 - A shared-memory region resides in the address space of the process that creates the shared-memory region.
 - Other processes that wish to communicate using this shared-memory region must attach shared address space to their address space.
 - Normally, the operating system tries to prevent one process from accessing another process's memory.
 - Shared memory requires that two or more processes agree to remove this restriction.
 - The form of data and location are determined by cooperating processes and are not under the operating system's control.
 - The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Producer–Consumer Problem: A Paradigm for Cooperating Processes

- A producer process produces information that is consumed by a consumer process.
 - A Web server produces HTML, images, etc., client computer consume the information through the browser.
 - Compiler code makes assembly code consumed by an assembler.
- The use of shared memory is a solution of producer-consumer problem.
 - A buffer is maintained in the shared memory region, that is filled by a producer and emptied by the consumer.
 - A producer can produce one item while the consumer is consuming another item.
 - The producer and consumer are synchronized, so that the consumer could not try to consume an item that has not yet been produced.

Shared-Memory System (contd...)

- Two types of buffers can be used:
 - Unbounded Buffer
 - The consumer waits for a new item, however, **there is no restriction on the producer to produce items.**
 - Bounded Buffer
 - Fixed buffer size.
 - If the buffer is empty, the consumer must wait for a new item.
 - When the buffer is full, the producer waits until it can produce new items.
 - **A bounded buffer is implemented using a circular queue of an array.**

Message-Passing Systems

- It is useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.
- A message-passing facility provides at least two operations: `send(message)` and `receive(message)`.
- If processes **P** and **Q** want to communicate, they must send messages to and receive messages from each other.
- A communication link exists between the communicating processes.
 - Logically, the link can be implemented with several methods:
 - Direct or Indirect Communication
 - Synchronous or Asynchronous (**Blocking or Non-blocking**) Communication
 - Automatic or Explicit Buffering

Direct Communication

- Each process that wants to communicate must explicitly **name** the recipient or sender of the communication.
 - **send(P, message)** - Send a message to process P.
 - **receive(Q, message)** - Receive a message from process Q.
- **Properties of Communication Link:**
 - A link is established between every pair of processes that want to communicate.
 - The processes need to know only each other's **identity** to communicate.
 - A link is associated with exactly two processes.
 - Between each pair of processes, there exists exactly one link.
- **Changing the identifier of a process may necessitate examining all other process definitions.**

Indirect Communication

- The messages are sent to and received from **mailboxes** or **ports**.
 - A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
 - Each mailbox has a unique identification.
 - A process can communicate with some other process via a number of different mailboxes.
 - **Two processes can communicate only if the processes have a shared mailbox.**
 - `send(A, message)` - Send a message to mailbox A.
 - `receive(A, message)` - Receive a message from mailbox A.

- **Properties of Communication Link:**

- A link is established between a pair of processes only if both members of the pair have a shared mailbox (port).
 - A link may be associated with more than two processes.
 - Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.
- A mailbox may be owned either by a process or by the operating system.
 - Suppose that processes P1, P2, and P3 all share mailbox A. Process P1 sends a message to A, while both P2 and P3 execute a receive() from A. Which process will receive the message sent by P1?
 - The answer depends on the scheme that we choose: (a) Allow a link to be associated with at most two processes. (b) Allow at most one process at a time to execute a receive operation. (c) Allow the system to select arbitrarily which process will receive the message.

Synchronization

- Communication between processes takes place through calls to `send()` and `receive()` primitives.
- There are different design options for implementing each primitive.
- Message passing may be either **blocking** or **non-blocking** - also known as **synchronous** and **asynchronous**.
 - **Blocking send** - The sending process is blocked until the message is received by the receiving process or by the mailbox.
 - **Non-blocking send** - The sending process sends the message and resumes operation.
 - **Blocking receive** - The receiver blocks until a message is available.
 - **Non-blocking receive** - The receiver retrieves either a valid message or a null.

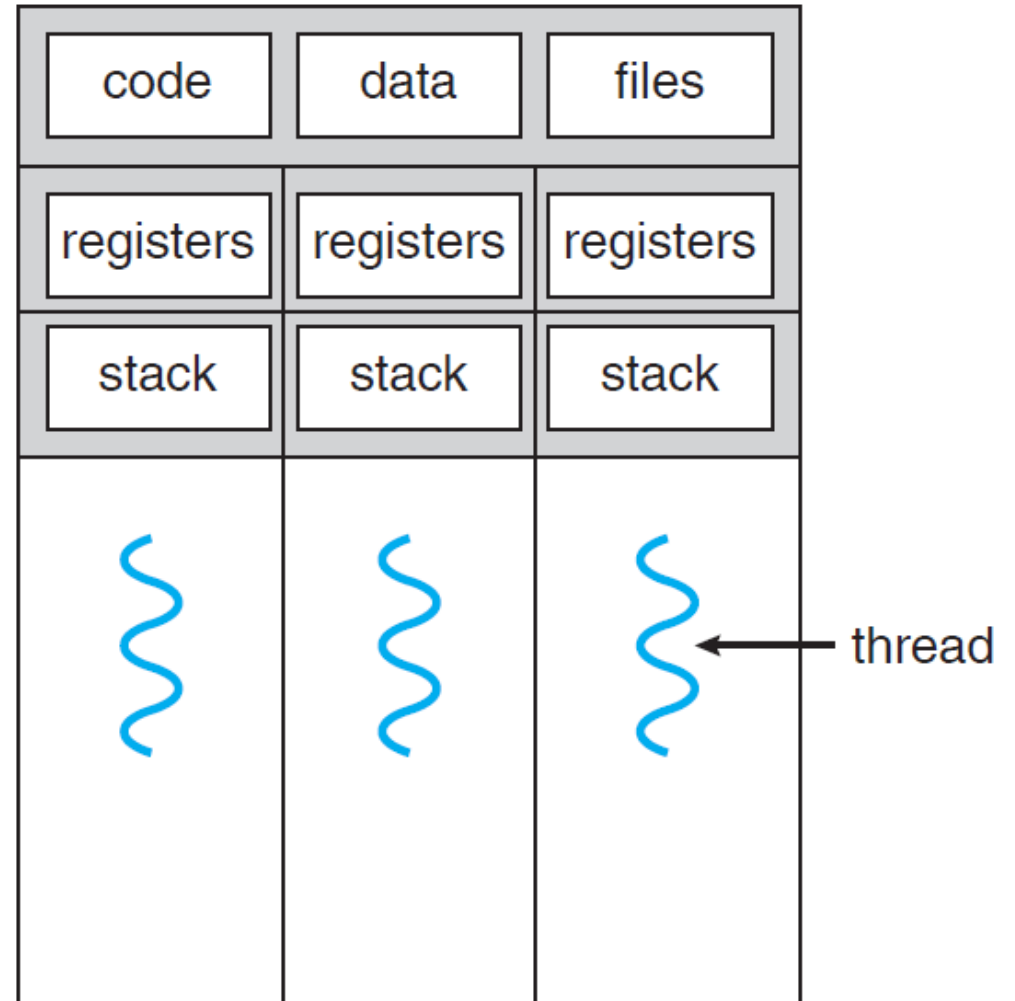
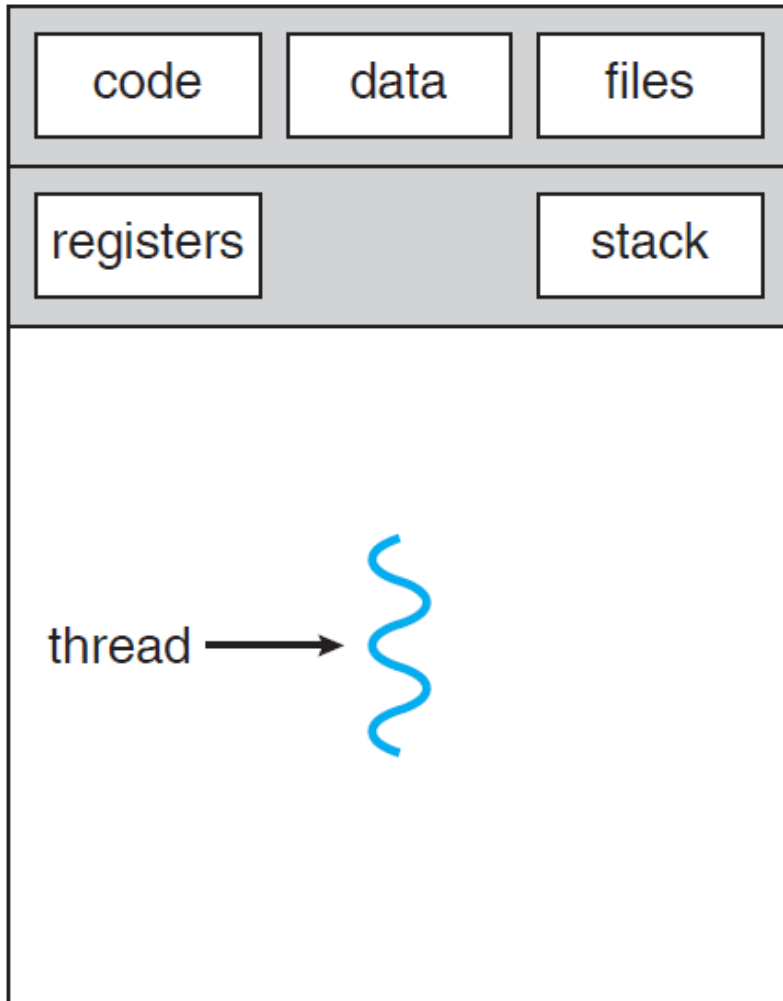
Buffering

- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.
- Such queues can be implemented in three ways:
 - **Zero Capacity** - This queue cannot keep any message waiting in it. For this, a sending process must be blocked until the receiving process receives the message. It is also known as no buffering.
 - **Bounded Capacity** - This queue has finite length n . Thus it can have n messages waiting in it. If the queue is not full, new message can be placed in the queue, and a sending process is not blocked. It is also known as automatic buffering.
 - **Unbounded Capacity** - The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

Threads: Motivation

- Many software packages that run on modern desktop PCs are multithreaded.
- An application typically is implemented as a separate process with several threads of control.
 - A Web browser might have one thread display images or text while another thread retrieves data from the network.
 - A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.
 - Web server runs as a single process and creates a separate thread for each client request.

Single-threaded and Multithreaded Process



Benefits of Multithreading

■ Responsiveness

- A multithreaded application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation.
- A multithreaded web browser could allow user interaction in one thread while an image was being loaded in another thread.

● Resource Sharing

- Threads share the memory and the resources of the process to which they belong by default.
- The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

● Economy

- Threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.

- Unix is considered as the mother of most of the operating systems.
- Earlier, the operating systems were designed with a particular machine in mind.
 - They were invariably written in a low-level language.
 - The systems were fast but restricted to the hardware they were designed for.
- The development of UNIX was started in 1969, by Thompson and Ritchie, at AT&T.
 - Initially, they designed a small system with limited utilities.
 - In 1973, they rewrote the entire system in C language.
- Unix is a family of multi-programing, multi-tasking and multi-user computer operating systems.

Berkeley Software Distribution (BSD)

- A U. S. Govt. decree (subsequently revoked) prevented AT&T from selling computer software.
- Therefore, AT&T distributed UNIX to academic and research institutions **at nominal cost**, **but without any support**.
- The University of California, Berkeley created a UNIX of its own.
 - **They called it BSD UNIX (Berkeley Software Distribution)**
 - Berkeley filled the gaps left behind by AT&T, and rewrite the whole OS.
 - They created standard editor of UNIX system (vi) and a popular shell (C shell).
- BSD offered UNIX for free to many companies.

Other Systems

- Sun used the BSD system as a foundation for developing their own brand of **UNIX** (SunOS, now known as Solaris).
- Other companies had their own brands, and **UNIX finally turned commercial**.
 - IBM -----> AIX
 - HP -----> HP-UX
- As each vendor modified and enhanced UNIX to create its own version, the **original UNIX lost its identity as a separate product**.
- AT&T sold its UNIX business to Novell, who later turned over the UNIX trademark to a standard body called X/OPEN, now merged with **The Open Group**.

Other Systems (contd...)

- UNIX is a registered trademark of **The Open Group**.
 - **The Open Group** refers to a family of computer operating systems and tools conforming to **The Open Group Base Specification**, Issue 7 (also known as POSIX.1-2008 or IEEE Std 1003.1 - 2008).
 - To use the Unix trademark, an operating system vendor must pay a licensing fee and annual trademark royalties to The Open Group.
 - Officially licensed Unix operating systems (and their vendors) include **Mac (Apple)**, **Solaris (Sun/Oracle)**, **AIX (IBM)**, **HP-UX (Hewlett-Packard)**, etc.

UNIX Philosophy

- Make each program do one thing well.
 - Reusable software tools: 1 tool = 1 function
- Expect the output of every program to become the input of another, yet unknown, program to combine simple tools to perform complex tasks
- Everything seen as a file.

UNIX-like Systems

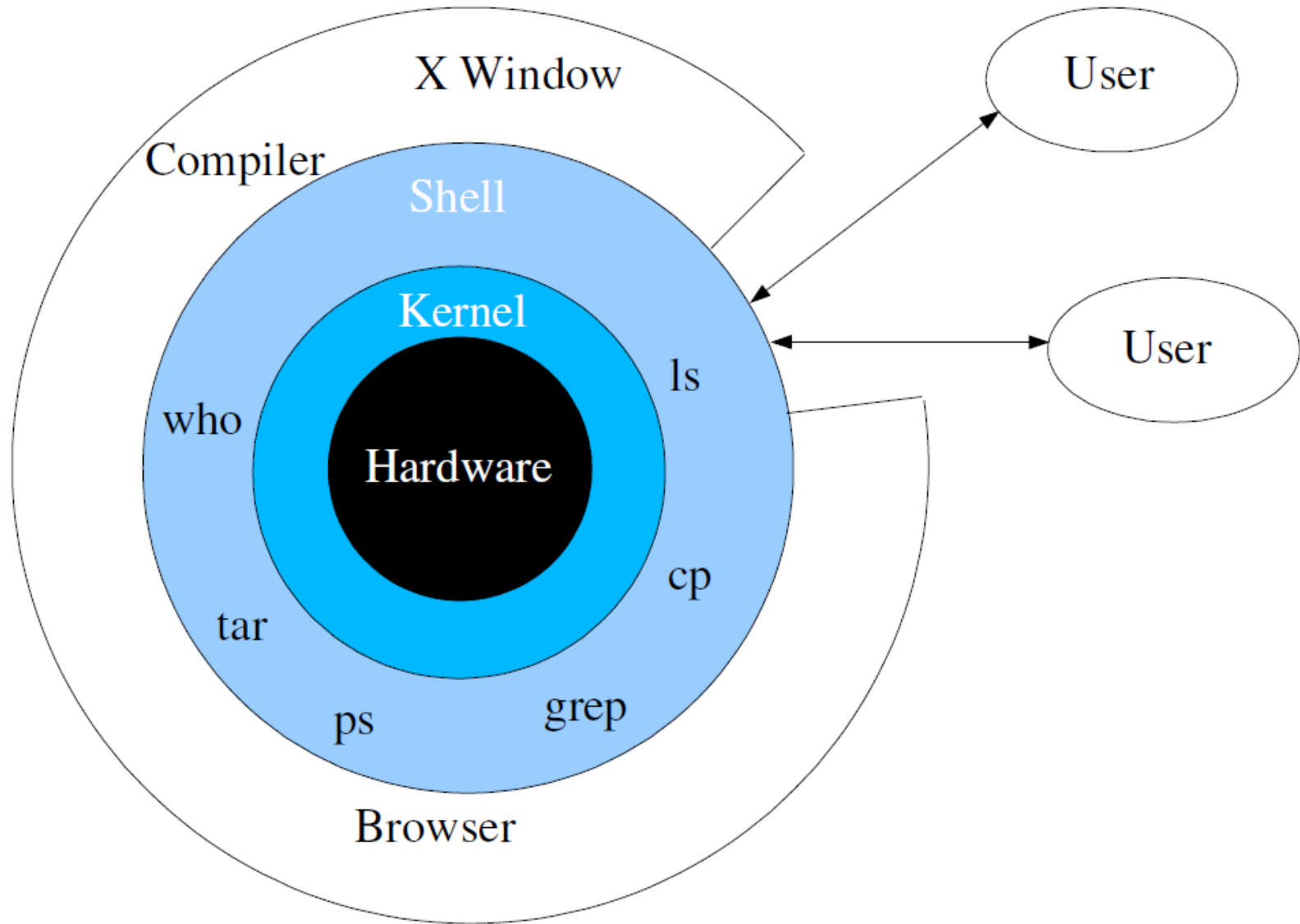
- The operating systems that behave like UNIX systems and provide similar utilities, but do not conform to UNIX specification or are not licensed by The Open Group, are commonly known as Unix-like systems.
- These include a wide variety of Linux distributions, for example:
 - Red Hat Enterprise Linux
 - Ubuntu
 - CentOS
 - Several Descendants of the Berkeley Software Distribution OS
 - FreeBSD
 - OpenBSD
 - NetBSD

- Linux and Unix are different but they do have a relationship with each other as Linux is derived from Unix.
- Linux is a Unix-like kernel.
- Linux is a Unix clone.
- Linus Torvalds is the father of Linux – **the free Unix**.
- Linux is just the kernel and not the complete OS.
 - The Linux kernel is generally packaged in Linux distributions which thereby makes it a complete OS.

Linux (contd...)

- Linux is distributed under the **GNU General Public License** which makes it mandatory for developers and sellers to make the source code public.
 - **GNU**, also known as **Free Software Foundation**, have written many important Linux tools .
 - Development of GNU/Linux began in 1984, when the Free Software Foundation began development of a free Unix-like operating system called GNU.
 - Today, development on Linux is carried out at several locations across the globe at the behest of the **Free Software Foundation**.
 - The most popular GNU/Linux distributions include **Red Hat**, **Caldera**, **Debian (Ubuntu)** and **Mandrake**.

Architecture of GNU/Linux

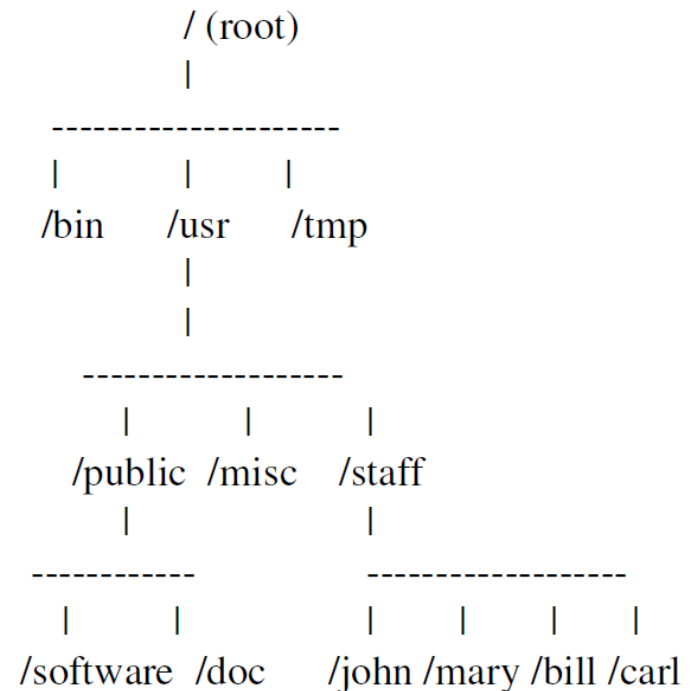
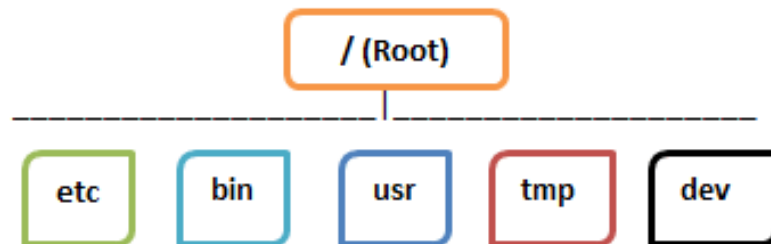


Features of GNU/Linux

- Portable
- Open Source
- Multiuser
- Multiprogramming
- Hierarchical File System
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

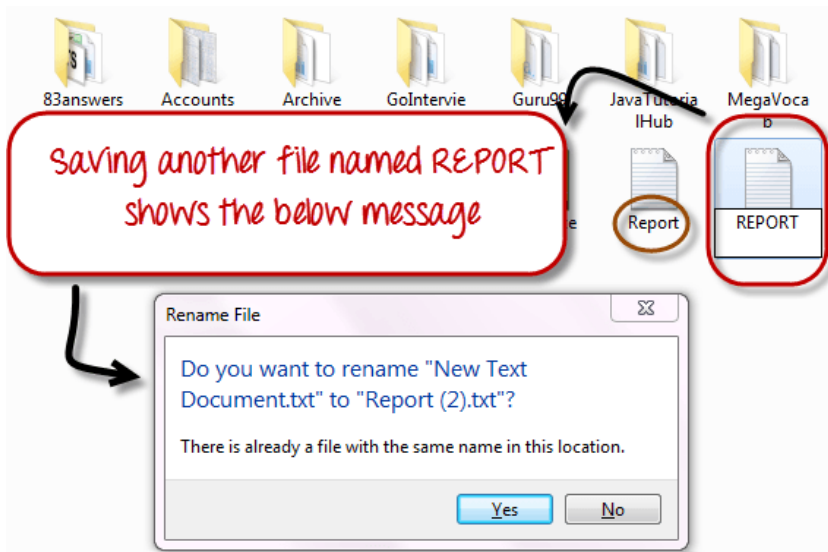
Linux File System

- In Linux, everything is seen as a file.
- Files are ordered in a tree structure starting with the root directory.
- This root directory can be considered as the start of the file system, and it further branches out various other subdirectories.
- The root is denoted with a forward slash '/'.

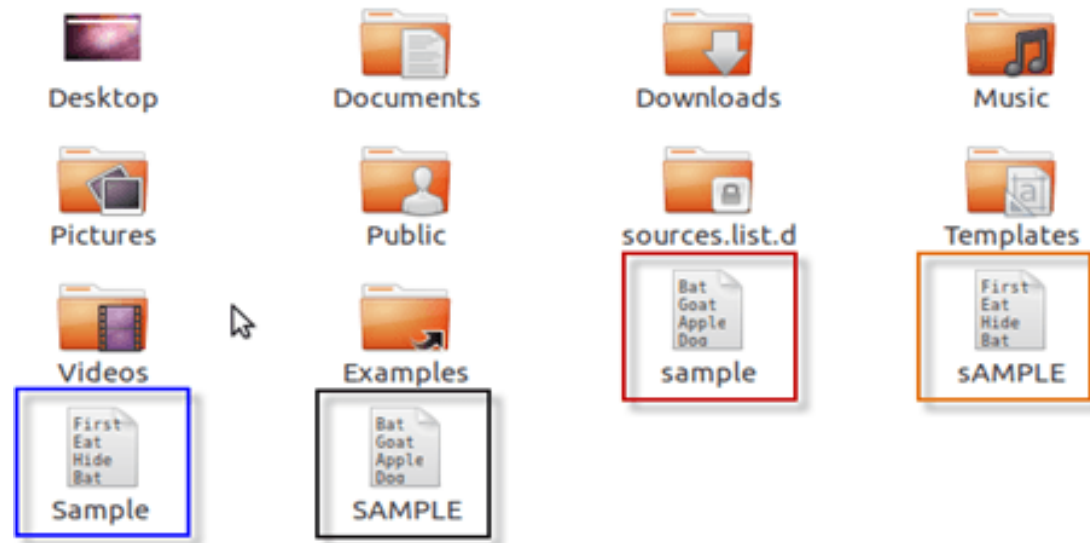


Linux File System (contd...)

- Linux is **cAsE sEnSiTiVe**
 - Each file in a given directory must be unique.



Windows



Linux

- In Linux, we can have 2 files with the same name in the same directory, provided they use different cases.

Types of Files

- In Linux and UNIX, everything is a file.
- If something is not a file, then it must be running as a process on the system.
- **General Files (Ordinary Files)** - They can contain image, video, program or simply text. They can be in ASCII or a Binary format.
- **Directory Files** - These files are a warehouse for other file types. We can have a directory file within a directory (sub-directory). We can take them as 'Folders' found in Windows operating system.
- **Device Files** - In MS Windows, devices like Printers, CD-ROM, and hard drives are represented as drive letters like G: H:. In Linux, they are represented as files. For example, if the hard drive has three partitions, they would be named and numbered as `/dev/sda1`, `/dev/sda2` and `/dev/sda3`.
- All device files reside in the directory `/dev/`

Linux File System (contd...)

- Files are referenced by names.
- **/home/sunil**
 - First / represents the root directory.
 - It is an absolute pathname, which is a sequence of directory names separated by slashes.
 - An absolute pathname shows a file's location with reference to the top, i.e., root.
- Shortcuts
 - .. (Parent Directory)
 - .(Current Directory)
 - ~ (Home Directory)

Windows Vs. Linux

Windows	Linux
Windows uses different data drives like C: D: E to stored files and folders.	Unix/Linux uses a tree like a hierarchical file system.
Windows has different drives like C: D: E	There are no drives in Linux
Hard drives, CD-ROMs, printers are considered as devices	Peripherals like hard drives, CD-ROMs, printers are also considered files in Linux/Unix
There are 4 types of user account types 1) Administrator, 2) Standard, 3) Child, 4) Guest	There are 3 types of user account types 1) Regular, 2) Root and 3) Service Account
Administrator user has all administrative privileges of computers.	Root user is the super user and has all administrative privileges.
In Windows, you cannot have 2 files with the same name in the same folder	Linux file naming convention is case sensitive. Thus, sample and SAMPLE are 2 different files in Linux/Unix operating system.
In windows, My Documents is default home directory.	For every user /home/username directory is created which is called his home directory.

Commands in Linux

- Linux commands are words that when keyed in cause some action to take place.
- Commands are actually files containing programs often written in C. These files are contained in directories.
- Seldom more than four characters in length like ls, who, ps.
- Lower case
- Case sensitive

Command Interpreter/Shell

- Shell is a program that acts as the interface between user and the Linux system.
- It allows us to enter commands for the operating system to execute.
- There are multiple shells in Linux:
 - Bash (standard shell in Linux)
 - Bourne shell (sh)
 - C Shell (csh)
 - Korn shell (ksh)
- When we key in a command at the command prompt, the shell searches for a file matching the command name in a list of directories.
- If it is found, then the program corresponding to the file is executed. If it is not found, a command not found error is given.

Types of Commands

- External Commands

- External commands are those which exist as separate files programs.
- Most commands in Linux are of this nature.
- Example: **ps**

- Internal Commands

- The commands whose implementation is written within the shell.
- They do not exist as a separate file.
- Example: **echo**

- With **type** command, we can determine whether a command is external or internal.

Structure of Commands

- Command can be one word or multiple words separated by white spaces.
- In multiple words command:
 - **First word**: the **actual name of the command**
 - **Other words**: **arguments** (can be options or expressions or file names)
 - A command can perform different task depending on the option specified.
 - The options are generally preceded by a by a single or double minus sign, called **Short** or **Long** option respectively.
 - **Commands can be combined together as per need.**