RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Information Technology, V-Semester

IT501 - Operating System

Course Objectives

- Learn concepts of operating systems
- Learn the mechanisms of OS to handle processes
- Study of various mechanisms involved in memory management techniques
- Gaining knowledge of deadlocks prevention and detection techniques
- Analyzing disk management functions and techniques

Unit I

Introduction to Operating Systems, Evaluation of OS, Types of operating Systems, system protection, Operating system services, Operating System structure, System Calls and System Boots, Operating System design and implementation, Spooling and Buffering.

Unit II

Basic concepts of CPU scheduling, Scheduling criteria, Scheduling algorithms, algorithm evaluation, multiple processor scheduling. Process concept, operations on processes, threads, inter process communication, precedence graphs, critical section problem, semaphores, classical problems of synchronization,

Unit III

Deadlock problem, deadlock characterization, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock, Methods for deadlock handling. Concepts of memory management, logical and physical address space, swapping, Fixed and Dynamic Partitions, Best-Fit, First-Fit and Worst Fit Allocation, paging, segmentation, and paging combined with segmentation.

Unit IV

Concepts of virtual memory, Cache Memory Organization, demand paging, page replacement algorithms, allocation of frames, thrashing, demand segmentation, Role of Operating System in Security, Security Breaches, System Protection, and Password Managment.

Unit V

Disk scheduling, file concepts, File manager, File organization, access methods, allocation methods, free space managements, directory systems, file protection, file organization & access mechanism, file sharing implement issue, File Management in Linux, introduction to distributed systems.

References:

1. Silberschatz ,"Operating system", Willey Pub

2. Tanenbaum " Modern Operating System" PHI Learning.
3. Dhamdhere, "System Programming and Operating System",TMH.
4. Stuart,"Operating System Principles, Design &Applications",Cengage Learning
5. Operating System : Principle and Design by Pabitra Pal Choudhury, PHI Learning

Suggested List of Experiments

1. Program to implement FCFS CPU scheduling algorithm.
2. Program to implement SJF CPU scheduling algorithm.
3. Program to implement Priority CPU Scheduling algorithm.
4. Program to implement Round Robin CPU scheduling algorithm.
5. Program to implement classical inter process communication problem(producer consumer).
6. Program to implement classical inter process communication problem(Reader Writers).
7. Program to implement classical inter process communication problem(Dining Philosophers).
8. Program to implement FIFO page replacement algorithm.
9. Program to implement LRU page replacement algorithm

Course Outcomes

Upon successful completion of this course the students will:

- Gain knowledge of history of operating systems
- Understand designissuesassociatedwithoperatingsystems
- Gain knowledge ofvariousprocessmanagementconceptsincludingscheduling,synchronization,deadlocks
- Understand conceptsofmemorymanagementincludingvirtualmemory
- Understand issuesrelatedtofilesysteminterfaceandimplementation,diskmanagement
- Befamiliarwithprotectionandsecuritymechanisms
- BefamiliarwithvarioustypesofoperatingsystemsincludingUnix

Class Notes

Introduction to operating System

- Operating System is a Resource Manager.
- It is a Control Program.

Definition: Is a collection of software enhancements, executed on the bare hardware, culminating in a high-level virtual machine that serves as an advanced programming environment

An Operating System (OS) is an interface between computer user and computer hardware. An operating system is software, which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.
Some popular Operating Systems include Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.
An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

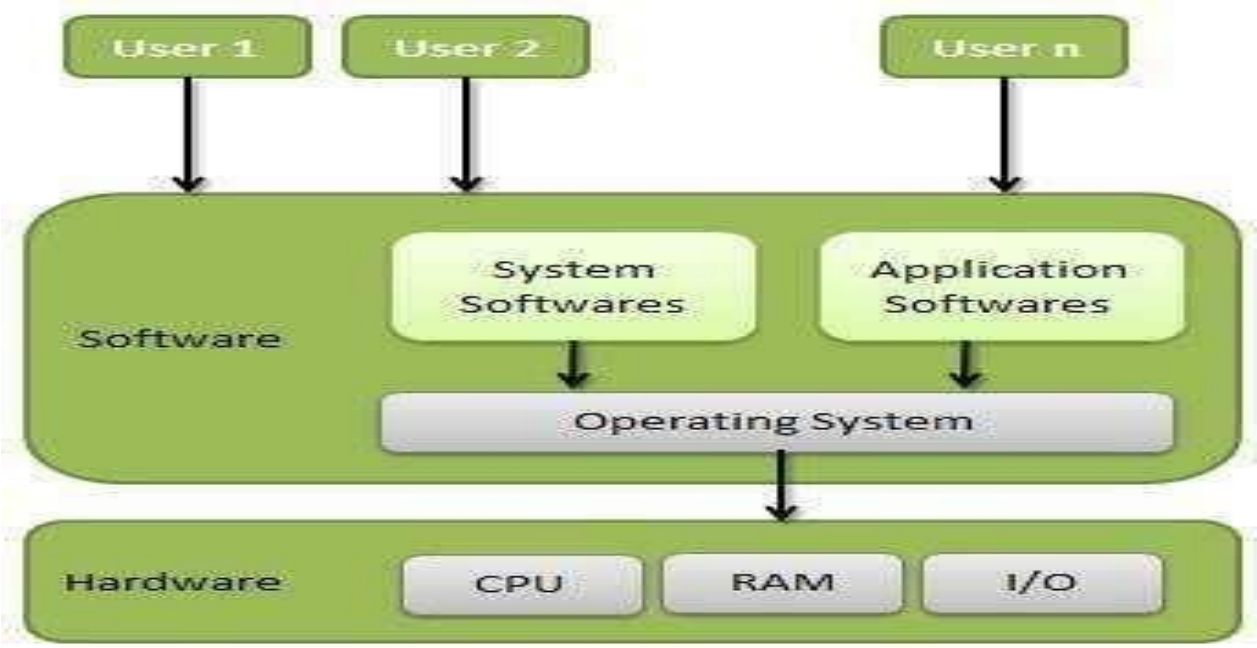


Fig 1.1 OS

Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be access directly by the CPU. For a program to be executed it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what parts are not in use
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so
- De-allocates the memory when a process no longer needs it or has been terminated

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process the program responsible for this task is known as traffic controller
- Allocates the processor (CPU) to a process
- De-allocates processor when a process is no longer required

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices program responsible for this task is known as the I/O controller
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way
- De-allocates devices

File Management

A file system is normally organized into directories for easy navigation and usage these directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc the collective facilities are often known as file system
- Decides who gets the resources
- Allocates the resources
- De-allocates the resources

Other Important Activities

Following are some of the important activities that an Operating System performs −

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids
- **Coordination between other software and users** − Coordination and assignment of compilers interpreters, assemblers and other software to the various users of the computer systems

Need of Operating System

- Computer hardware is developed to execute user programs and make solving user problems easier
- An operating system makes a computer more convenient to use.

It acts as an interface between user and computer hardware. Therefore, the end-users are not particularly concerned with the computer's architecture, and they view the computer system in terms of an application.

To programmers, it provides some basic utilities to assist him in creating programs, the management of files, and the control of I/O devices.

Operating System Objectives

- Convenience
- Efficiency
- Ability to evolve

Services Provided by Operating Systems

- Facilities for program creation
- Program execution
- Access to I/O and files
- System access
- Error detection and response
    - Internal and external hardware errors
        - Memory error
        - Device failure
    - Software errors
        - Arithmetic overflow
        - Access forbidden memory locations
    - Operating system cannot grant request of application
- Accounting
    - Collect statistics
    - Monitor performance
    - Used to anticipate future enhancements
    - Used for billing users

Computer System Components

A computer system can be dividing into four components

The Hardware: Provides basic computing resources (CPU, memory, I/O devices)

The Operating System: Controls and coordinates the use of the hardware among the various application programs for the various users.

The Application Programs: Define the ways in which the system resources are use to solve the computing problems of the users (compilers, database systems, video games, business programs)

The Users: Users (people, machines, other computers).These components can be view as a layer where each layer uses the services provided by the layer beneath it

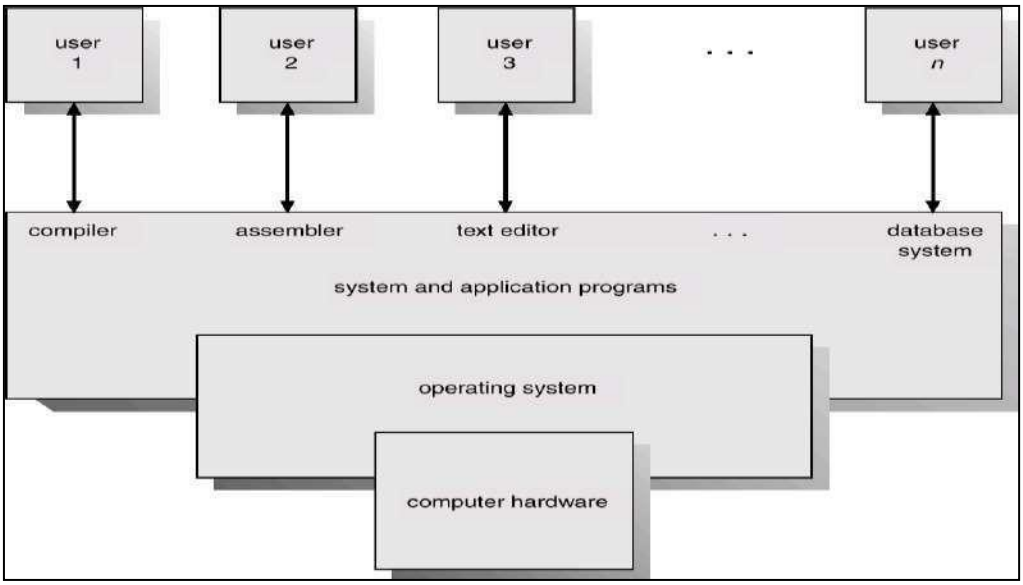


Fig 1.2 Components of OS

Evolution of operating systems

Serial Processing

- Users access the computer in series. From the late 1940's to mid-1950's, the programmer interacted directly with computer hardware i.e., no operating system. These machines were run with a console consisting of display lights, toggle switches, some form of input device and a printer.
- Programs in machine code are loaded with the input device like card reader. If an error occur the program was halted, and the error condition was indicated by lights. Programmers examine the registers and main memory to determine error. If the program is success, then output will appear on the printer.
- Main problem here is the setup time. That is single program needs to load source program into memory, saving the compiled (object) program and then loading and linking together.

Simple Batch Systems

To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers will leave their programs with the operator. The operator will sort programs into batches with similar requirements.

The problems with Batch Systems are:

- Lack of interaction between the user and job.
- CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.

For overcoming this problem use the Spooling Technique. Spool is a buffer that holds output for a device, such as printer, that cannot accept interleaved data streams. That is when the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the job is completed, the output is printed. Spooling technique can keep both the CPU and the I/O devices working at much higher rates.

Multiprogrammed Batch Systems

- Jobs must be run sequentially, on a first come, first-served basis. However, when several jobs are on a direct-access device like disk, job scheduling is possible. The main aspect of job scheduling is

multiprogramming. Single user cannot always keep the CPU or I/O devices busy. Thus, multiprogramming increases CPU utilization.

- In when one job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back.

Time-Sharing Systems

- Time-sharing systems are not available in 1960s. Time-sharing or multitasking is a logical extension of multiprogramming. That is processors time is shared among multiple users simultaneously is called time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is in Multiprogrammed batch systems its objective is maximize processor use, whereas in Time-Sharing Systems its objective is minimize response time.
- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is seconds at most.
- Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Personal-Computer Systems (PCs)

- A computer system is dedicated to a single user is called personal computer, appeared in the 1970s. Micro computers are considerably smaller and less expensive than mainframe computers. The goals of the operating system have changed with time; instead of maximizing CPU and peripheral utilization, the systems developed for maximizing user convenience and responsiveness. For e.g., MS-DOS, Microsoft Windows and Apple Macintosh.
- Hardware costs for microcomputers are sufficiently low. Decrease the cost of computer hardware (such as processors and other devices) will increase our needs to understand the concepts of operating system. Malicious programs destroy data on systems. These programs may be self-replicating and may spread rapidly via worm or virus mechanisms to disrupt entire companies or even worldwide networks.
- MULTICS operating system was developed from 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a computing utility. Many of the ideas in MULTICS were subsequently used at Bell Laboratories in the design of UNIX OS.

Parallel Systems

Most systems to date are single-processor systems; that is they have only one main CPU. Multiprocessor systems have more than one processor.
The advantages of parallel system are as follows:

- throughput (Number of jobs to finish in a time period)
- Save money by sharing peripherals, cabinets and power supplies
- Increase reliability
- Fault-tolerant (Failure of one processor will not halt the system).

Symmetric multiprocessing model

- Each processor runs an identical job (copy) of the operating system, and these copies communicate. Encore's version of UNIX operating system is a symmetric model.
- E.g., If two processors are connected by a bus. One is primary and the other is the backup. At fixed check points in the execution of the system, the state information of each job is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated, and is restarted from the most recent checkpoint. But it is expensive.

Asymmetric multiprocessing model

Each processor is assigned a specific task. A master processor controls the system. Sun's operating system SunOS version 4 is an asymmetric model. Personal computers contain a microprocessor in the keyboard to convert the keystrokes into codes to be sent to the CPU.

Distributed Systems

Distributed systems distribute computation among several processors. In contrast to tightly coupled systems (i.e., parallel systems), the processors do not share memory or a clock. Instead, each processor has its own local memory.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers and so on.

The advantages of distributed systems are as follows:

- Resource Sharing: With resource sharing facility user at one site may be able to use the resources available at another.
- Communication Speedup: Speedup the exchange of data with one another via electronic mail.
- Reliability: If one site fails in a distributed system, the remaining sites can potentially continue operating.

Real-time Systems

Real-time systems are used when there are rigid time requirements on the operation of a processor, or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail.

E.g., Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-appliance controllers.

There are two types of real-time systems:

- **Hard real-time systems**
  Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.

- **Soft real-time systems**
  Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems.

  E.g., Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers.

Types of OS

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time, Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows −

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, scientific experiments, medical image systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing, and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For

example, multimedia, virtual reality, Advanced Scientific Projects likes undersea exploration and planetary rovers, etc.

System Protection:

Goals of Protection

- To prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies,* which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

Principles of Protection

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Operating system structure:

An operating system might have many structures. According to the structure of the operating system; operating systems can be classified into many categories.

Some of the main structures used in operating systems are:

1. Monolithic architecture of operating system

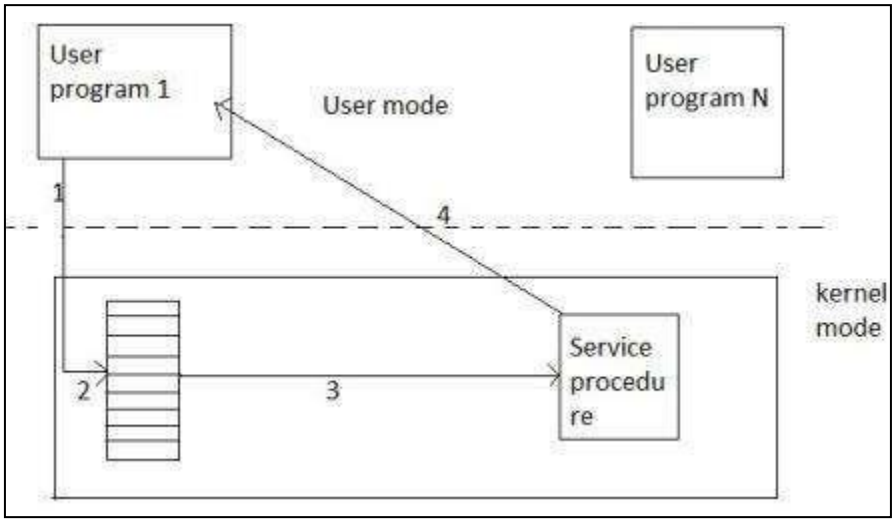


Fig 1.3: monolithic structure of operating system

It is the oldest architecture used for developing operating system. Operating system resides on kernel for anyone to execute. System call is involved i.e. Switching from user mode to kernel mode and transfer control to operating system shown as event 1. Many CPU has two modes, kernel mode, for the operating system in

which all instruction are allowed and user mode for user program in which I/O devices and certain other instruction are not allowed. Two operating system then examines the parameter of the call to determine which system call is to be carried out shown in event 2. Next, the operating system index's into a table that contains procedure that carries out system call. This operation is shown in events. Finally, it is called when the work has been completed and the system call is finished, control is given back to the user mode as shown in event 4.

2. Layered Architecture of operating system

The layered Architecture of operating system was developed in 60's in this approach; the operating system is broken up into number of layers. The bottom layer (layer 0) is the hardware layer and the highest layer (layer n) is the user interface layer as shown in the figure.

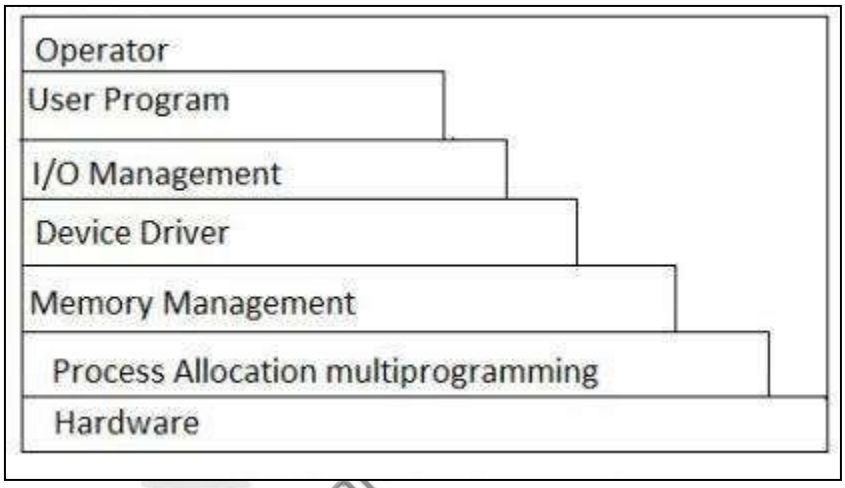


Fig 1.4: layered architecture

- The layered are selected such that each user functions and services of only lower level layer. The first layer can be debugged wit out any concern for the rest of the system. Its user basic hardware to implement this function once the first layer is debugged., it's correct functioning can be assumed while the second layer is debugged & soon. If an error is found during the debugged of particular layer, the layer must be on that layer, because the layer below it already debugged. Because of this design of the system is simplified when operating system is broken up into layer.
- Os/2 operating system is example of layered architecture of operating system another example is earlier version of Windows NT.
- The main disadvantage of this architecture is that it requires an appropriate definition of the various layers & a careful planning of the proper placement of the layer.

3. Virtual memory architecture of operating system

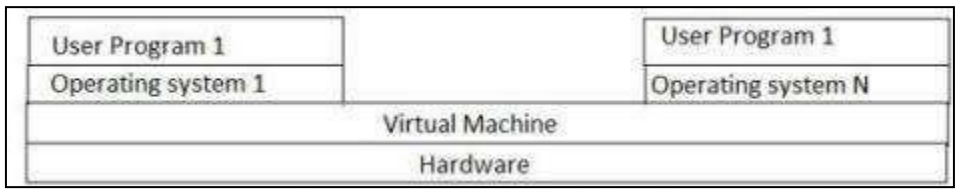


Fig 1.5: virtual memory architecture

Virtual machine is an illusion of a real machine. It is created by a real machine operating system, which make a single real machine appears to be several real machines. The architecture of virtual machine is shown above. The best example of virtual machine architecture is IBM 370 computer. In this system each user can choose a

different operating system. Virtual machine can run several operating systems at once, each of them on its virtual machine.

Its multiprogramming shares the resource of a single machine in different manner. The concepts of virtual machine are: -

a. Control program (cp): - cp creates the environment in which virtual machine can execute. It gives to each user facilities of real machine such as processor, storage I/0 devices.
b. conversation monitor system (cons): - cons is a system application having features of developing program. It contains editor, language translator, and various application packages.
c. Remote spooling communication system (RSCS): - provide virtual machine with the ability to transmit and receive file in distributed system.
d. IPCS (interactive problem control system):- it is used to fix the virtual machine software problems.

4. client/server architecture of operating system

A trend in modern operating system is to move maximum code into the higher level and remove as much as possible from operating system, minimizing the work of the kernel. The basic approach is to implement most of the operating system functions in user processes to request a service, such as request to read a particular file, user send a request to the server process, server checks the parameter and finds whether it is valid or not, after that server does the work and send back the answer to client server model works on request- response technique i.e. Client always send request to the side in order to perform the task, and on the other side, server gates complementing that request send back response. The figure below shows client server architecture.

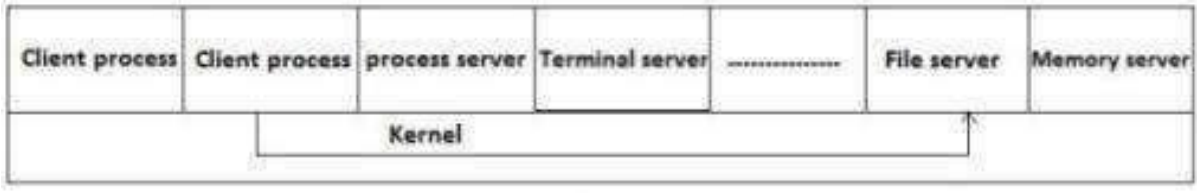


Fig 1.6: client server model

In this model, the main task of the kernel is to handle all the communication between the client and the server by splitting the operating system into number of ports, each of which only handle some specific task. I.e. file server, process server, terminal server and memory service.

Kernel

A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc). The main tasks of the kernel are:

Process management

- Device management
- Memory management
- Interrupt handling
- I/O communication
- File system

Types of Kernels

Kernels may be classified mainly in two categories

1. Monolithic
2. Micro Kernel

Monolithic Kernels

Earlier in this type of kernel architecture, all the basic system services like process and memory management, interrupt handling etc. were packaged into a single module in kernel space.
This type of architecture led to some serious drawbacks like
1) Size of kernel, which was huge.
2)Poor maintainability, which means bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours in a modern-day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and un-loaded. This modular approach allows easy extension of OS's capabilities. Linux follows the monolithic modular approach

Microkernels

This architecture majorly caters to the problem of ever-growing size of kernel code which we could not control in the monolithic approach. This architecture allows some basic services like device driver management, protocol stack, file system etc. to run in user space. This reduces the kernel code size and increases the security and stability of OS as we have the bare minimum code running in kernel. So, if suppose a basic service like network service crashes due to buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional.

System Calls Basics

To understand system calls, first one needs to understand the difference between **kernel mode** and **user mode** of a CPU. Every modern operating system supports these two modes.

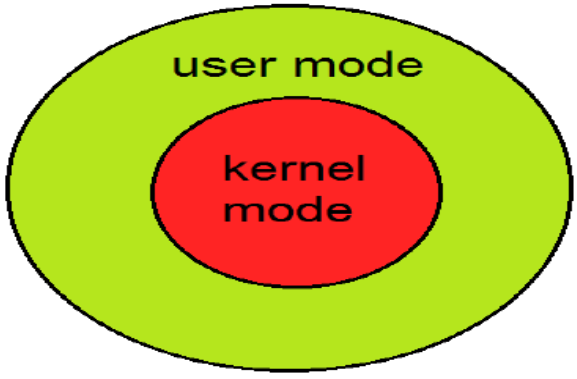


Fig 1.7 Modes supported by the operating system

Kernel Mode

- When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.

User Mode

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode.

System Call

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.
When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.

Then the kernel provides the resource which the program requested. After that, another context switching happens which results in change of mode from kernel mode back to user mode.
Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

System Boot

- Booting the system is done by loading the kernel into main memory and starting its execution.
- The CPU is given a reset event, and the instruction register is loaded with a predefined memory location, where execution starts.
    - The initial bootstrap program is found in the BIOS read-only memory.
    - This program can run diagnostics, initialize all components of the system, loads and starts the Operating System loader. (Called **boot strapping**)
    - The loader program loads and starts the operating system.
    - When the Operating system starts, it sets up needed data structures in memory, sets several registers in the CPU, and then creates and starts the first user level program. From this point, the operating system only runs in response to interrupts.

The following diagram demonstrates the steps involved in a system boot process:

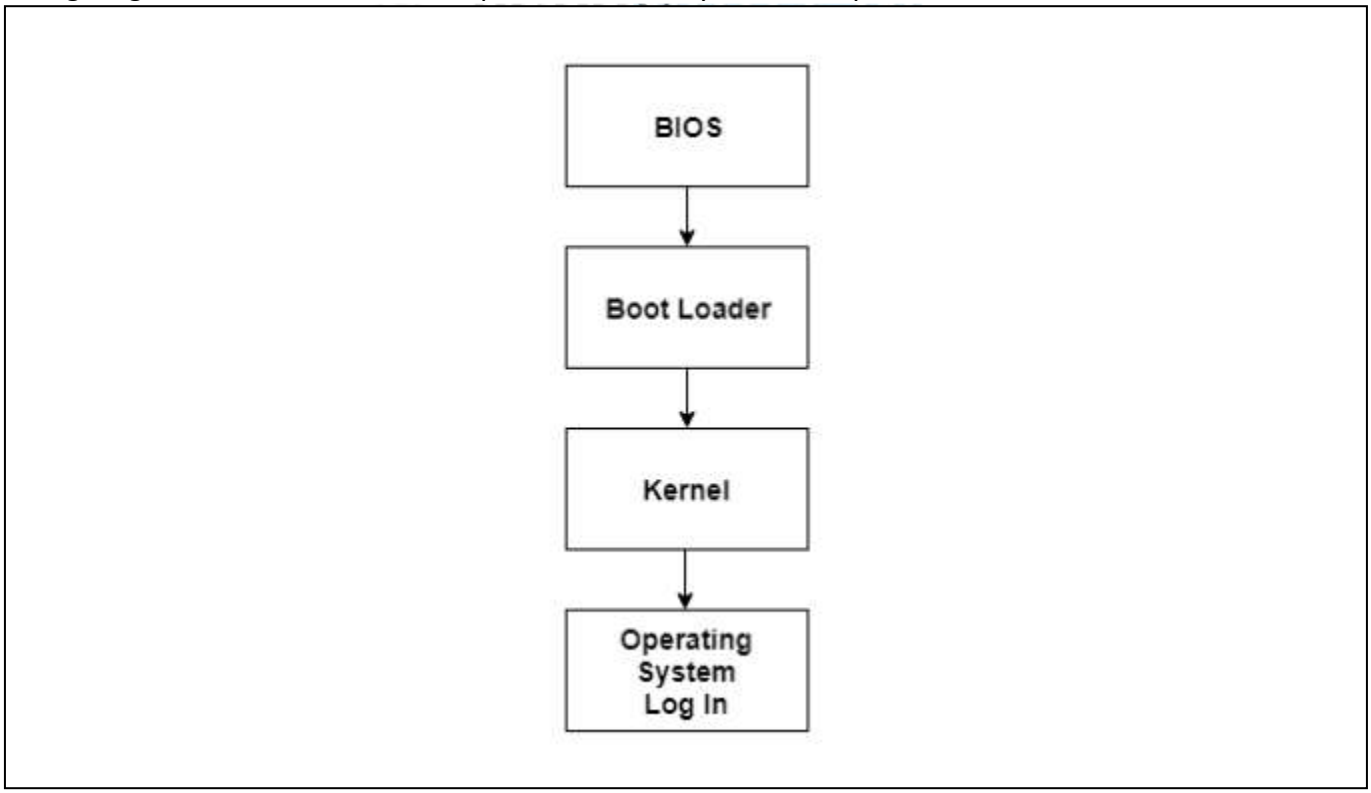


Fig 1.8: steps involved in a system boot process

Operating System Design:

Design Goals

The design changes depending on the type of the operating system i.e. if it is batch system, time shared system, single user system, multiuser system, distributed system etc.

There are basically two types of goals while designing an operating system. These are:

User Goals

The operating system should be convenient, easy to use, reliable, safe and fast according to the users. However, these specifications are not very useful as there is no set method to achieve these goals.

System Goals

The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

Operating System Implementation

The operating system needs to be implemented after it is designed. Earlier they were written in assembly language, but now higher-level languages are used. The first system not written in assembly language was the Master Control Program (MCP) for Burroughs Computers.

Advantages of Higher-Level Language

There are multiple advantages to implementing an operating system using a higher-level language such as: the code is written faster; it is compact and easier to debug and understand. Also, the operating system can be easily moved from one hardware to another if it is written in a high-level language.

Disadvantages of Higher-Level Language

Using high level language for implementing an operating system leads to a loss in speed and increase in storage requirements. However, in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager. Also, the bottleneck routines in the system can be replaced by assembly language equivalents if required.

Spooling –

Spooling stands for Simultaneous peripheral operation online. A spool is a like buffer as it holds the jobs for a device till the device is ready to accept the job. It considers disk as a huge buffer which can store as many jobs for the device till the output devices are ready to accept them.

Buffering –

Main memory has an area called buffer that is used to store or hold the data temporarily that is being transmitted either between two devices or between a device or an application. Buffering is an act of storing data temporarily in the buffer. It helps in matching the speed of the data stream between the sender and receiver. If speed of the sender's transmission is slower than receiver, then a buffer is created in main memory of the receiver, and it accumulates the bytes received from the sender and vice versa.

The basic difference between Spooling and Buffering is that Spooling overlaps the input/output of one job with the execution of another job while the buffering overlaps input/output of one job with the execution of the same job.

Differences between Spooling and Buffering –

- The key difference between spooling and buffering is that Spooling can handle the input/output of one job along with the computation of another job at the same time while buffering handles input/output of one job along with its computation.
- Spooling is a stand for Simultaneous Peripheral Operation online. Whereas buffering is not an acronym.
- Spooling is more efficient than buffering, as spooling can overlap processing two jobs at a time.
- Buffering use limited area in main memory while Spooling uses the disk as a huge buffer.

Department of Information Technology
OS (IT-501)

Unit 2

Process management: Process management is the process by which operating systems manage processes, threads, enable processes to share information, protect process resources and allocate system resources to processes that request them in a safe manner. This can be a daunting task to the operating system developer and can be very complex in design.

Basic Concepts of CPU Scheduling

- Maximum CPU utilization obtained with multiprogramming
- In a uniprocessor system, only one process may run at a time; any other processes must wait until the CPU is free and can be rescheduled

CPU–I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait.
- Processes alternate between these two states
- Process execution begins with a CPU burst , followed by an I/O burst , then another CPU burst ... etc
- The last CPU burst will end with a system request to terminate execution rather than with another I/O burst.
- The duration of these CPU burst have been measured.
- An I/O - bound program would typically have many short CPU bursts, A CPU- bound program might have a few very long CPU bursts.
- This can help to select an appropriate CPU - scheduling algorithm.

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc. thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Dispatcher

Another component involved in the CPU scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency. Dispatch Latency can be explained using the below figure:

Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for example, completion of I/O).
4. When a process terminates.

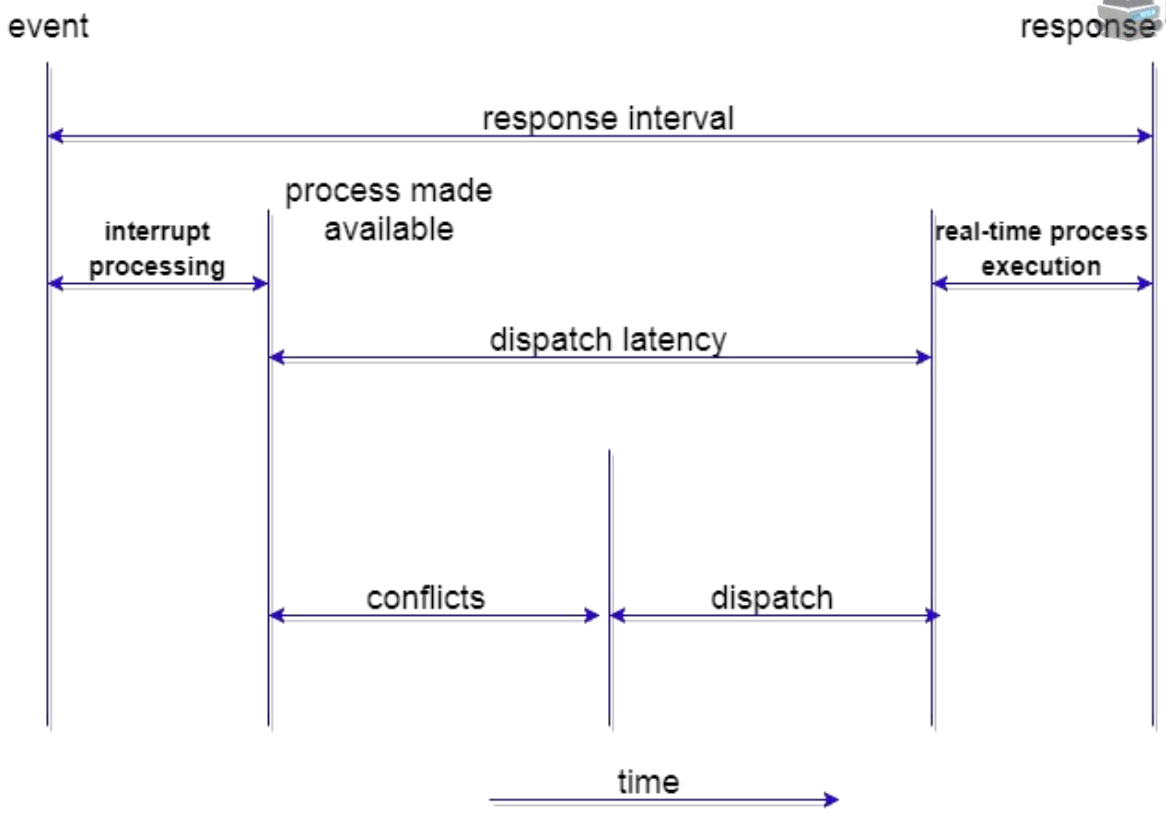


Figure 2.1 CPU Scheduling

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is non-preemptive; otherwise the scheduling scheme is preemptive.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware (for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Scheduling Criteria

There are many different criteria's to check when considering the "best" scheduling algorithm :

- CPU utilization
  To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- Throughput
  It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- Turnaround time
  It is the amount of time taken to execute a particular process, i.e. the interval from time of submission of the process to the time of completion of the process (Wall clock time).

- Waiting time
  The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- Load average
  It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- Response time
  Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

Scheduling algorithm:

A Process Scheduler schedules different process to be assigned to the CPU based on particular scheduling algorithms. There are six popular process-scheduling algorithms, which we are going to discuss in this chapter-

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithm are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

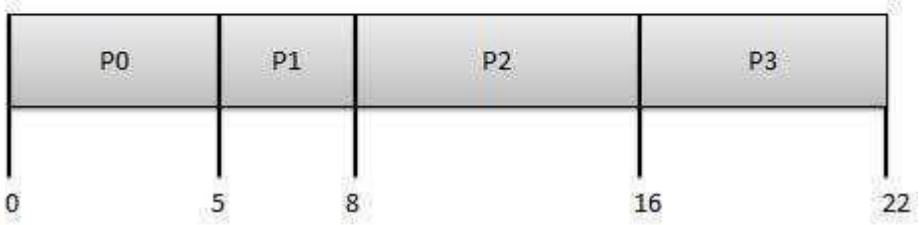


Figure 2.2 FCFS

Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job First(SJF)

- This is also known as shortest job first, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known
- The processer should know in advance how much time process will take

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

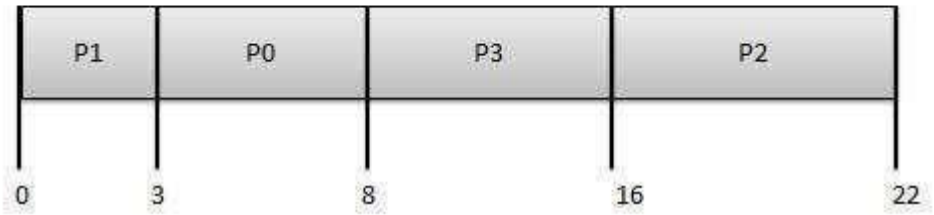


Figure 2.3 SJF

Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority process with highest priority is to be executed first and so on
- Processes with same priority are executed on first come first served basis
- Priority can be decided based on memory requirements, time requirements or any other resource requirement

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

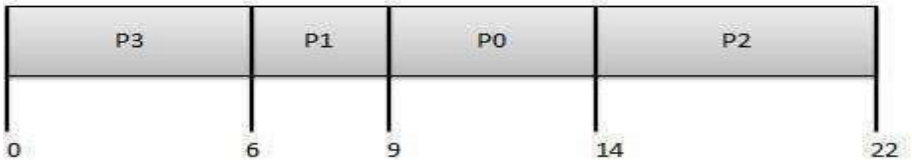


Figure 2.4 Priority Based Scheduling

Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion
- Impossible to implement in interactive systems where required CPU time is not known
- It is often used in batch environments where short jobs need to give preference

Round Robin Scheduling

- Round Robin is the preemptive process-scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes

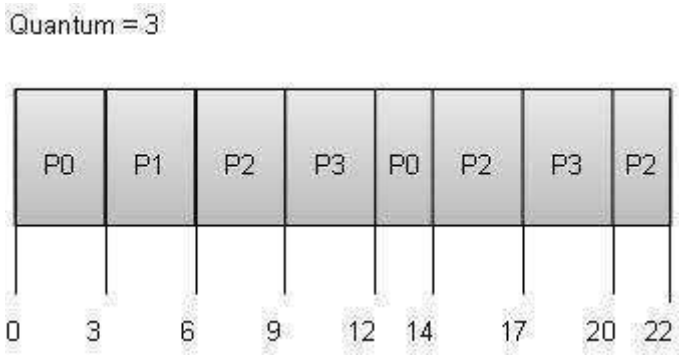


Figure 2.5 Round Robin Scheduling

Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are dependent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics

- Multiple queues are maintained for processes with common characteristics
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can schedule in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.

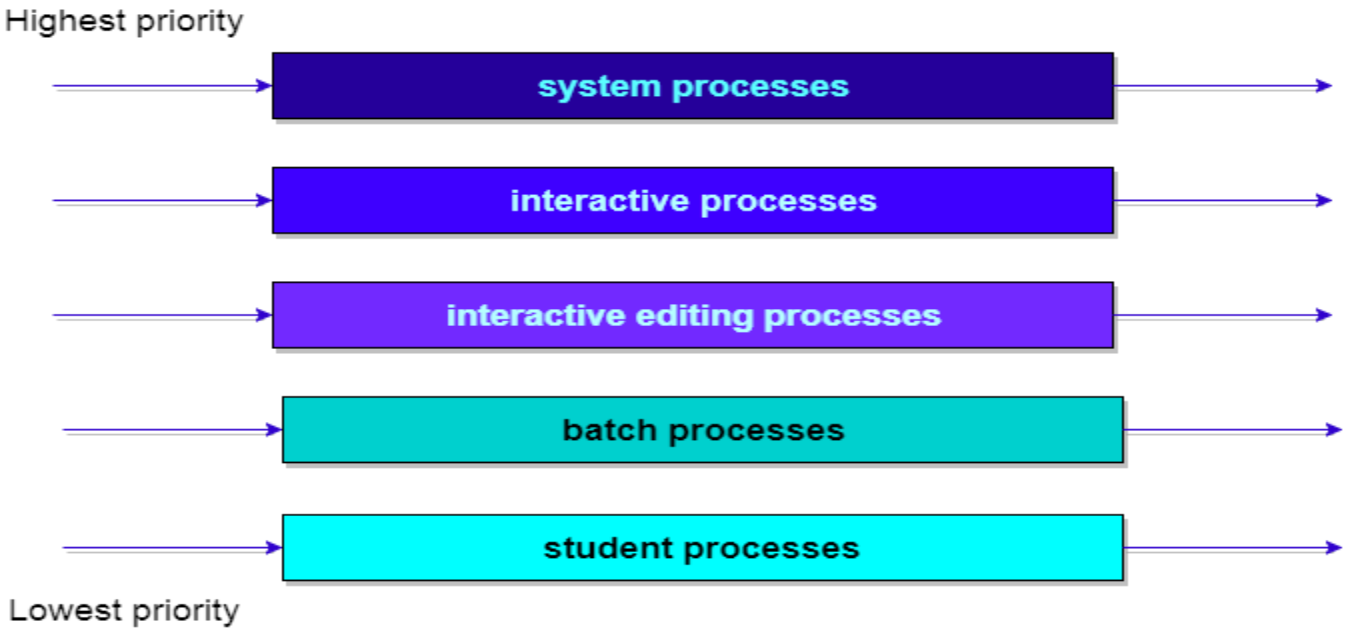


Figure 2.6 Multiple-Level Queues Scheduling

Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

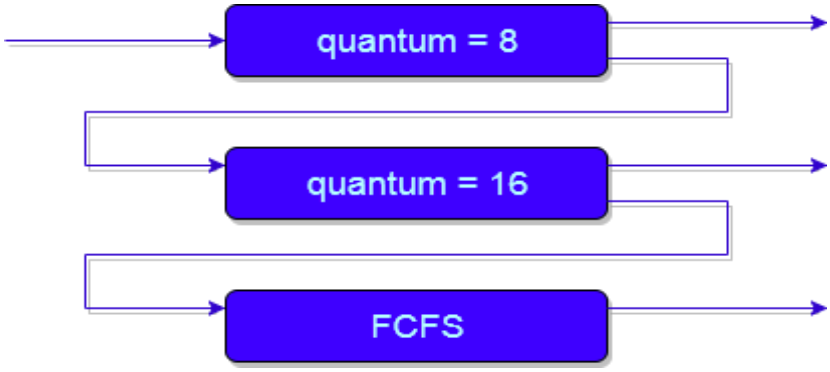


Figure 2.7 Multilevel Feedback Queue Scheduling

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

Evaluation of Process Scheduling Algorithms

Deterministic Modeling: This evaluation method takes a predetermined workload and evaluates each algorithm using that workload.

Queuing Models

Another method of evaluating scheduling algorithms is to use queuing theory. Using data from real processes we can arrive at a probability distribution for the length of a burst time and the I/O times for a process. We can now generate these times with a certain distribution.

- We can also generate arrival times for processes (arrival time distribution).
- If we define a queue for the CPU and a queue for each I/O device we can test the various scheduling algorithms using queuing theory.
- Knowing the arrival rates and the service rates we can calculate various figures such as average queue length, average wait time, CPU utilization etc.

Simulations

Rather than using queuing models we simulate a computer. A Variable, representing a clock is incremented. At each increment the state of the simulation is updated.

- Statistics are gathered at each clock tick so that the system performance can be analysed.
- The data to drive the simulation can be generated in the same way as the queuing model, although this leads to similar problems.
- Alternatively, we can use trace data. This is data collected from real processes on real machines and is fed into the simulation. This can often provide good results and good comparisons over a range of scheduling algorithms.
- However, simulations can take a long time to run, can take a long time to implement and the trace data may be difficult to collect and require large amounts of storage.

Implementation

The best way to compare algorithms is to implement them on real machines. This will give the best results but does have a number of disadvantages.

- It is expensive as the algorithm has to be written and then implemented on real hardware.
- If typical workloads are to be monitored, the scheduling algorithm must be used in a live situation. Users may not be happy with an environment that is constantly changing.
- If we find a scheduling algorithm that performs well there is no guarantee that this state will continue if the workload or environment changes.

Multiple-Processor Scheduling

•CPU scheduling more complex when multiple CPUs are available-Most current general purpose processors are multiprocessors (i.e. multicore processors)

-No single 'best' solution to multiple-processor scheduling

• A multicore processor typically has two or more homogeneous processor cores.

-Because the cores are all the same, any available processor can be allocated to any process in the system

Approaches to Multiple-Processor Scheduling

- Asymmetric multiprocessing

-All scheduling decisions, I/O processing, and other system activities handled by a single processor-Only one processor accesses the system data structures, alleviating the need for data sharing

- Symmetric multiprocessing (SMP)

-Each processor is self-scheduling-All processes may be in a common ready queue, or each processor may have its own private queue of ready processes-Currently, most common approach to multiple-processor scheduling

Process Concept

Process: A process is a program in execution. The execution of a process must progress in a sequential fashion a process defines as an entity, which represents the basic unit of work implemented in the system.
To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process, which performs all the tasks mentioned in the program.
When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –

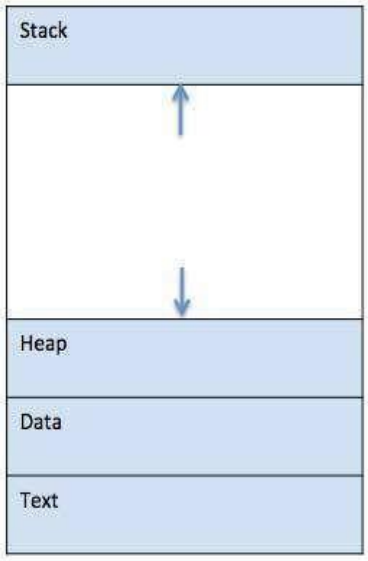


Figure 2.8 layout of a process inside main memory

Process State

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created
2	**Ready** The process is waiting to assign to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory

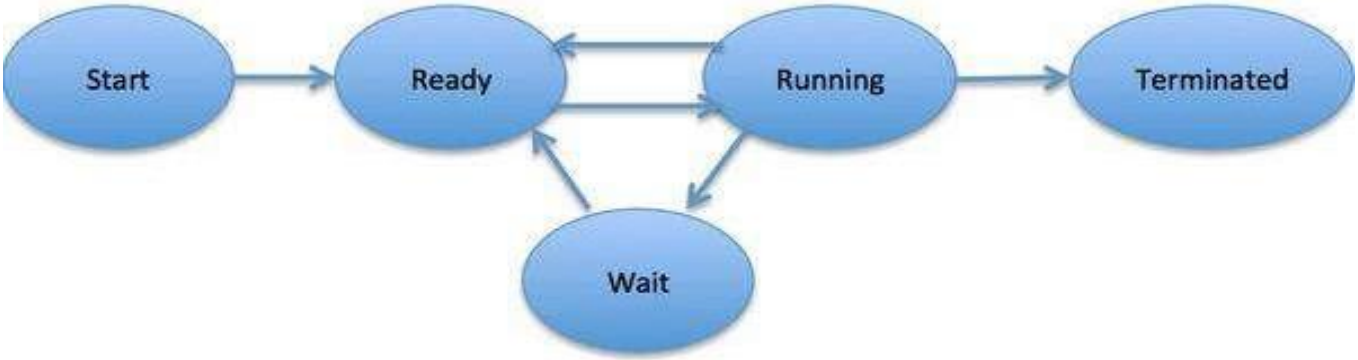


Figure 2.9 Process State Diagram

Process Control Block(PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. An integer process ID (PID) identifies the PCB. A PCB keeps all the information needed to keep track of a process as listed below in the table −

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system
4	**Pointer** A pointer to parent process
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process
8	**Memory management information** This includes the information of page table, memory limits, and Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −

Figure 2.10 PCB

The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates

Schedulers

Schedulers are special system software, which handles process scheduling in various ways. Their main task is to select jobs submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

A long-term scheduler determines which programs admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. 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.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long-term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers are.

Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspend if it makes an I/O request. Suspended processes cannot make any progress towards completion in this condition, to remove the process from memory and make space for

other process; the suspended process moved to the secondary storage. This process called swapping, and the process said swapped out or rolled out. Swapping may be necessary to improve the process mix

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long-term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can resume from the same point later. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

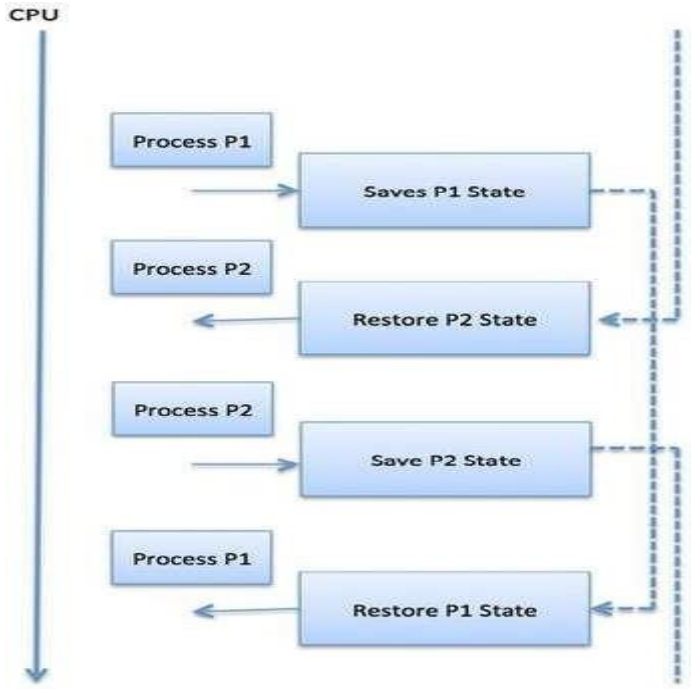


Figure 2.11 Context switches

Context switches are computationally intensive since register and memory state, some hardware systems employ two or more sets of processor registers. When the process switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Operations on Processes:

There are many operations that can be performed on processes. Some of these are process creation, process preemption, process blocking, and process termination. These are given in detail as follows:

Process Creation

Processes need to be created in the system for different operations. This can be done by the following events:

- User request for process creation
- System initialization
- Execution of a process creation system call by a running process
- Batch job initialization

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files, and environment strings. However, they have distinct address spaces.

A diagram that demonstrates process creation using fork() is as follows:

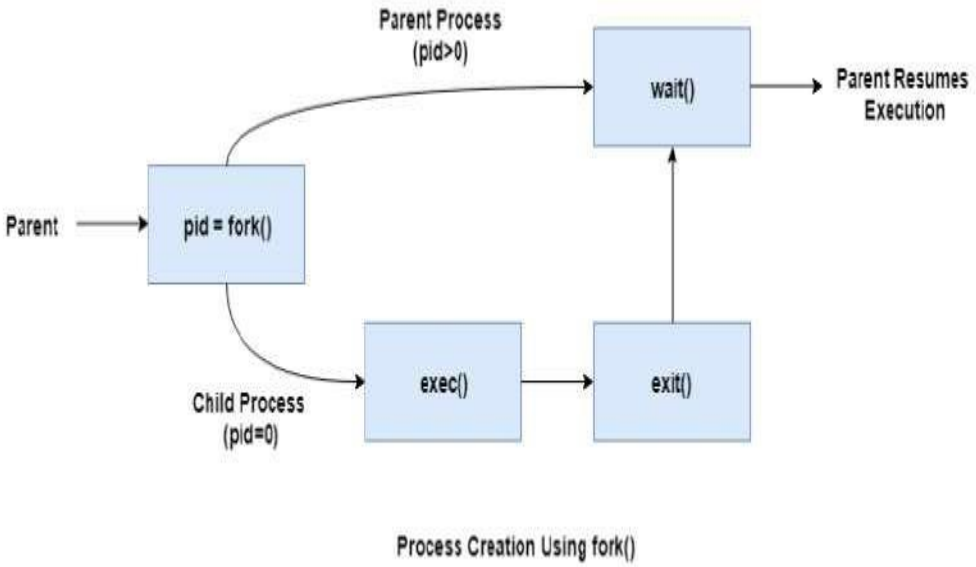


Figure 2.12 Process Creation

Process Preemption

An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by the short-term scheduler. Preemption makes sure that all processes get some CPU time for execution.

A diagram that demonstrates process preemption is as follows:

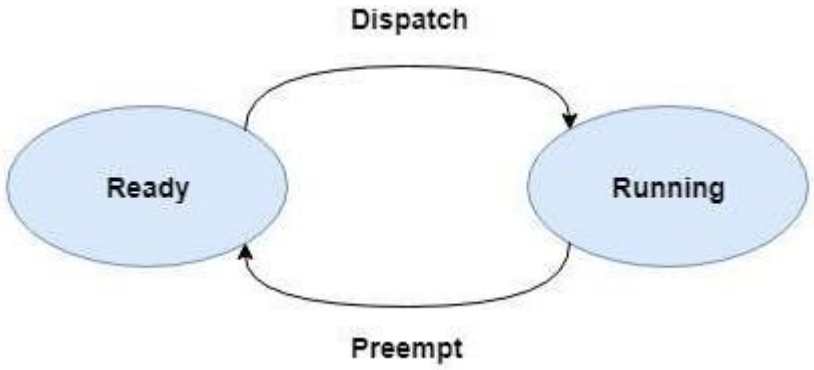


Figure 2.13 Process Preemption

Process Blocking

The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and don't require the processor. After the event is complete, the process again goes to the ready state.

A diagram that demonstrates process blocking is as follows:

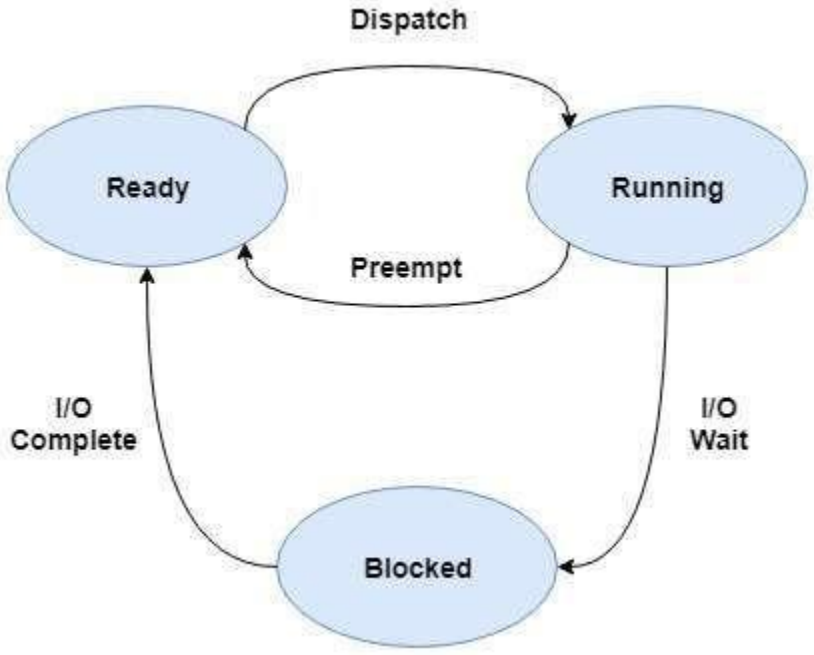


Figure 2.14 Process Blocking

Process Termination

After the process has completed the execution of its last instruction, it is terminated. The resources held by a process are released after it is terminated.

A child process can be terminated by its parent process if its task is no longer relevant. The child process sends its status information to the parent process before it terminates. Also, when a parent process is terminated, its child processes are terminated as well as the child processes cannot run if the parent processes are terminated.

Thread

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack, which contains the execution history.

A thread shares with its peer threads little information like code segment, data segment and open files. When one thread alters a code segment memory items all other threads see that

A thread is also called a lightweight process Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads has been successfully used in implementing network servers and web server they also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

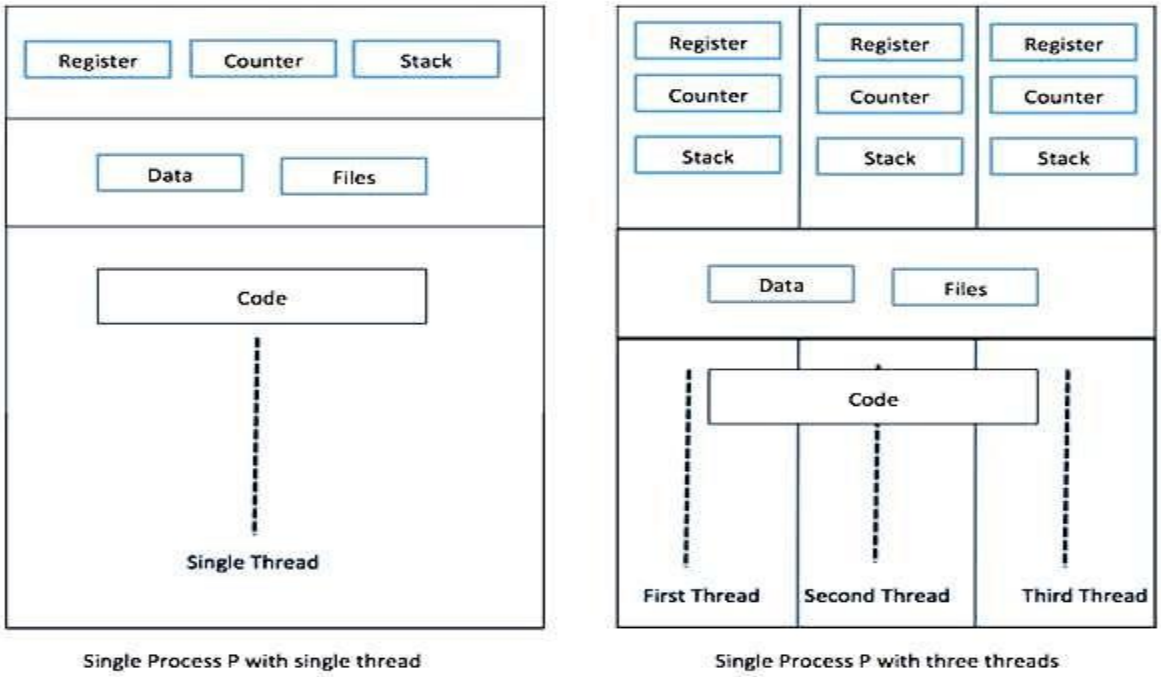


Figure 2.15 Threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.

3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes, each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads is implemented in following two ways-

User Level Threads − User managed threads.

Kernel Level Threads − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

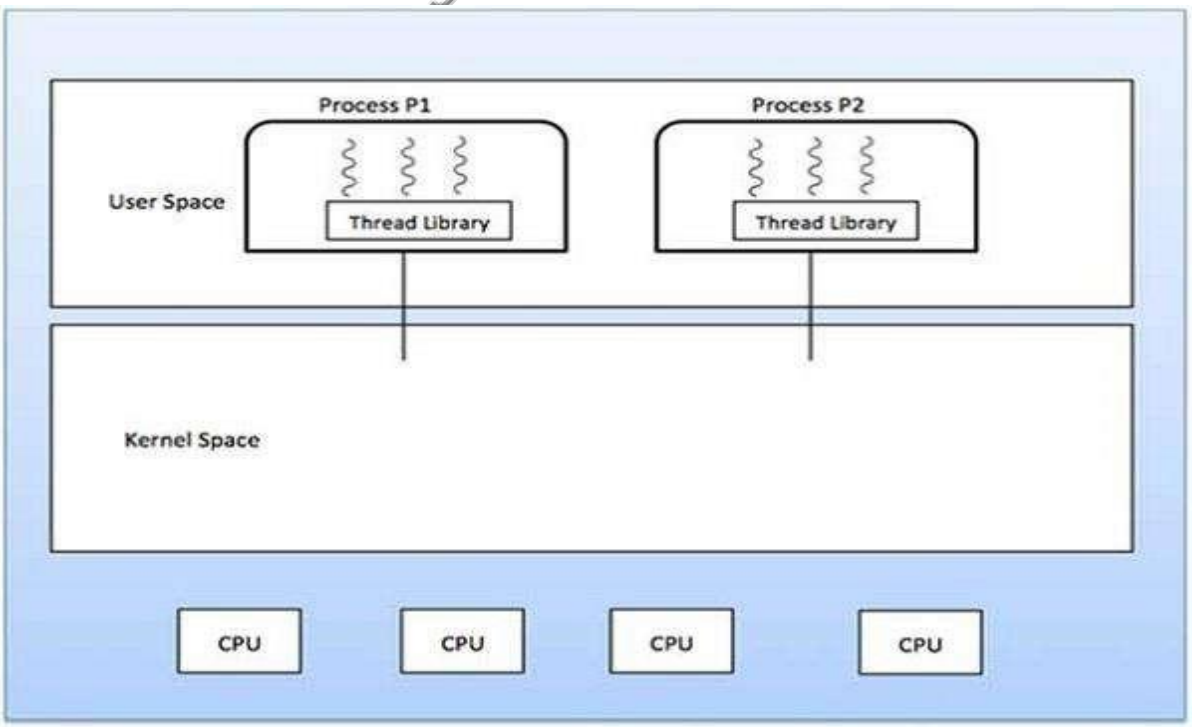


Figure 2.16 User level threads

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.
- Disadvantages
- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, the Kernel does thread management. There is no thread management code in the application area. Kernel threads supported directly by the operating system

Any application can be programmed to be multithreaded All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual's threads within the process. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many-to-many relationship
- Many to one relationship
- One to one relationship

Many-to-Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

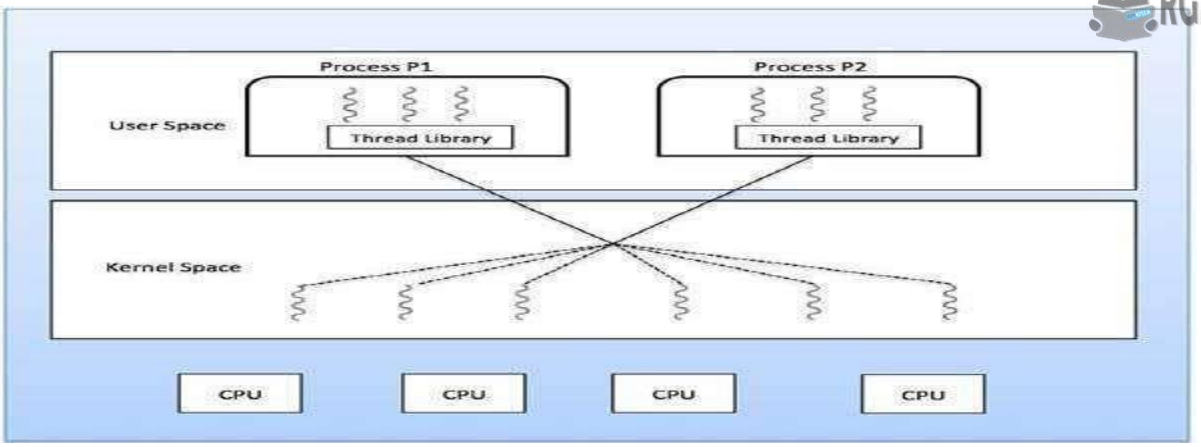


Figure 2.17 Many-to-Many Model

Many-to-One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management done in user space by the thread library. When thread makes a blocking system call, only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes

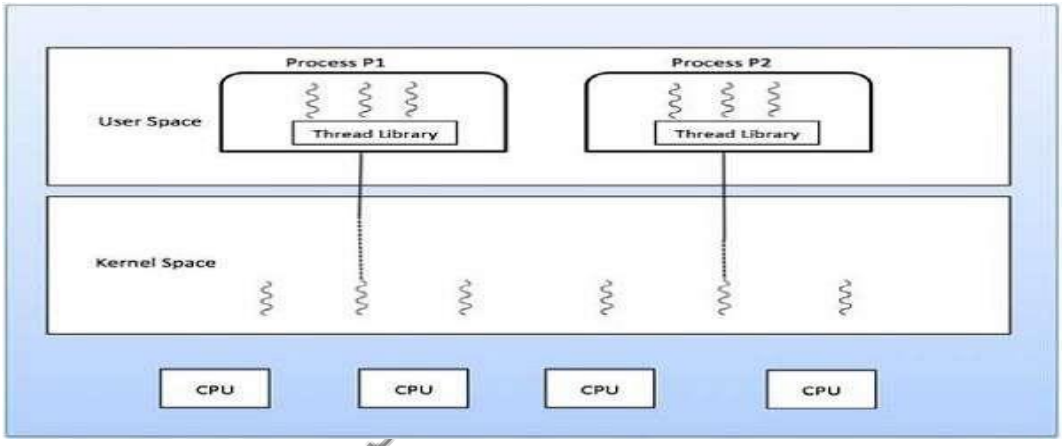


Figure 2.18 Many-to-One Model

One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, Windows NT and windows 2000 use one to one relationship model.

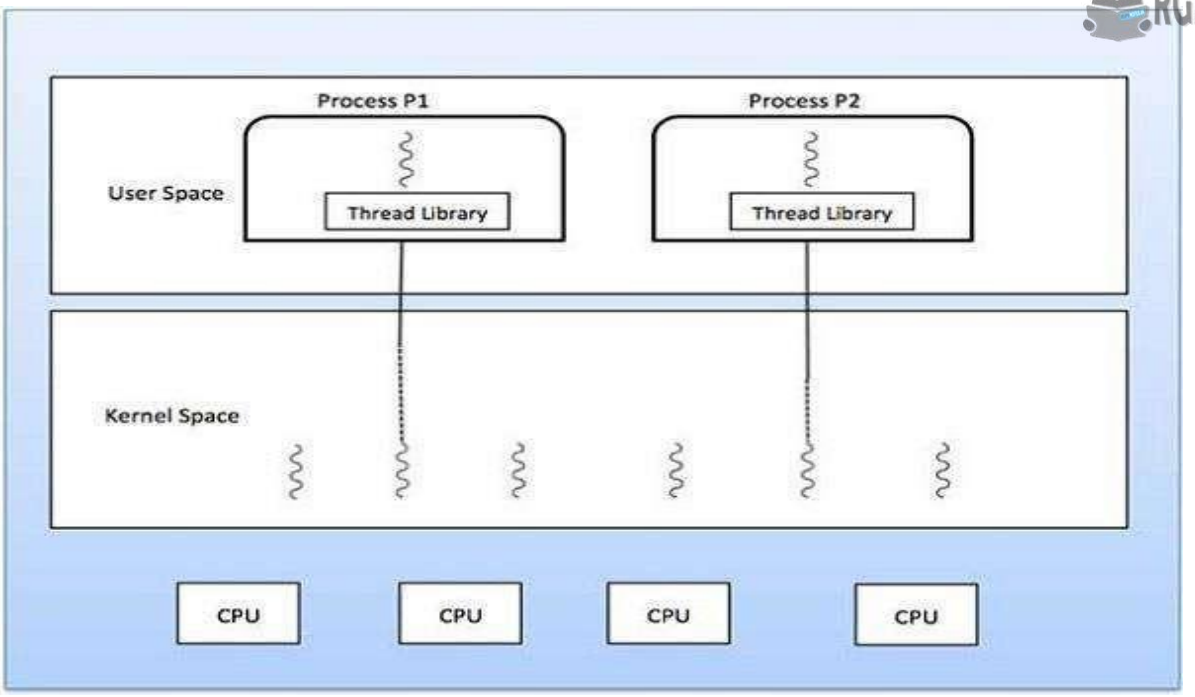


Figure 2.19 One to One Model

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process**: Execution of one process does not affect the execution of other processes.
- **Cooperative Process**: Execution of one process affects the execution of other processes.

Cooperating Processes

- Once we have multiple processes or threads, it is likely that two or more of them will want to communicate with each other
- Process cooperation (i.e., interprocess communication) deals with three main issues
  - Passing information between processes/threads
  - Making sure that processes/threads do not interfere with each other
  - Ensuring proper sequencing of dependent operations
- These issues apply to both processes and threads
  - Initially we concentrate on shared memory mechanisms

Cooperating Process Definition

- An independent process cannot affect or be affected by the execution of another process.
- A cooperating process can affect or be affected by the execution of another process

Advantages of process cooperation

- Information sharing
- Computation speed-up
- Modularity
- Convenience

Issues for Cooperating Processes

- Race conditions
  - A race condition is a situation where the semantics of an operation on shared memory are affected by the arbitrary timing sequence of collaborating processes
- Critical regions
  - A critical region is a portion of a process that accesses shared memory
- Mutual exclusion
  - Mutual exclusion is a mechanism to enforce that only one process at a time is allowed into a critical region

Cooperating Processes Approach

- Any approach to process cooperation requires that
  - No two processes may be simultaneously inside their critical regions
  - No assumptions may be made about speeds or the number of CPUs
  - No process running outside of its critical region may block other processes
  - No process should have to wait forever to enter its critical region

Inter Process Communication

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via shared memory method and via message passing.

An operating system can implement both method of communication. First, we will discuss the shared memory method of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.

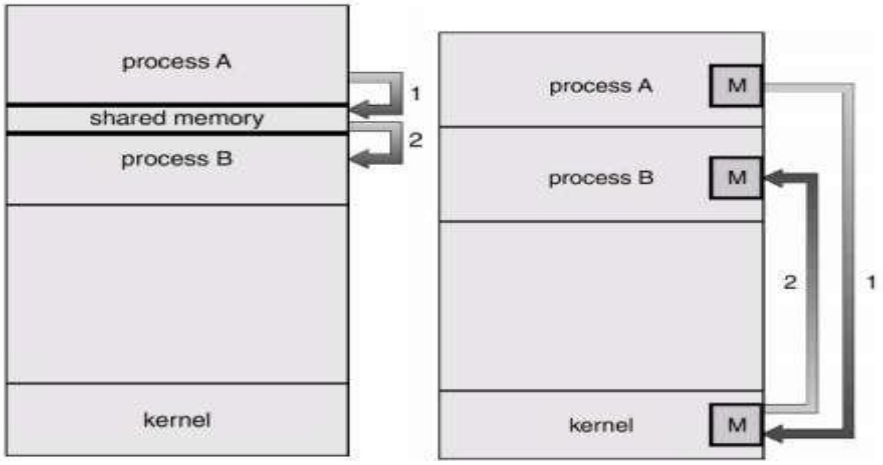


Figure 2.20 Message Passing

i) Shared Memory Method

Ex: Producer-Consumer problem

There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes shares a common space or memory location known as buffer where the item produced by Producer is stored and from where the Consumer consumes the item if needed. There are two version of this problem: first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on size of buffer, the second one is known as bounded buffer problem in which producer can produce up to a certain amount of item and after that it starts waiting for consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, and then producer will start producing items. If the total produced item is equal to the size of buffer, producer will wait to get it consumed by the Consumer. Similarly, the consumer first checks for the availability of the item and if no item is available, Consumer will wait for producer to produce it. If there are items available, consumer will consume it.

ii) Messaging Passing Method

Now, we will start our discussion for the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follow:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
- We need at least two primitives:
  Send (message, destination) or **send** (message)
  Receive (message, host) or **receive** (message)

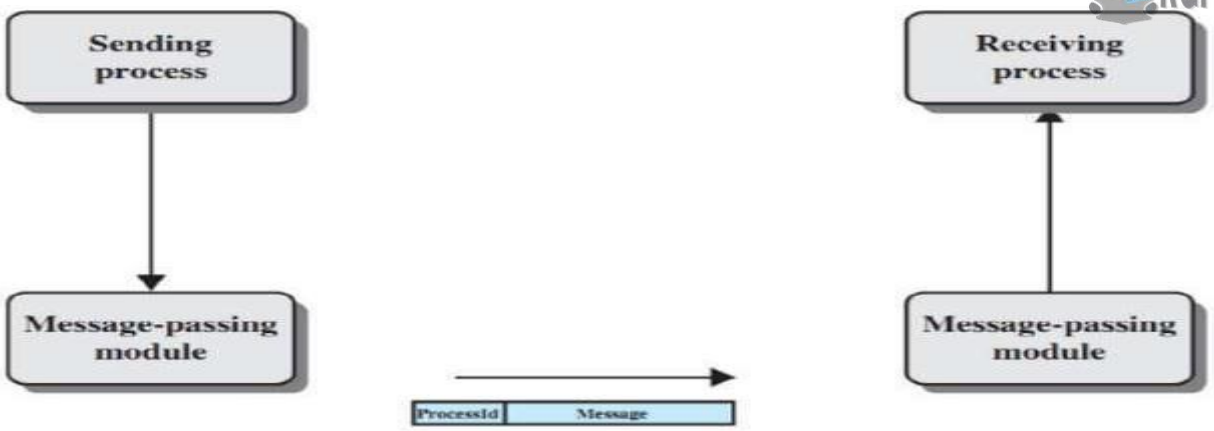


Figure 2.21 Message Passing method

The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for OS designer but complicated for programmer and if it is of variable size then it is easy for programmer but complicated for the OS designer. A standard message can have two parts: **header and body.** The **header part** is used for storing Message type, destination id, source id, and message length and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Synchronous and Asynchronous Message Passing:

A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. IPC is possible between the processes on same computer as well as on the processes running on different computer i.e. in networked/distributed system. In both cases, the process may or may not be blocked while sending a message or attempting to receive a message so Message passing may be blocking or non-blocking. Blocking is considered **synchronous** and **blocking send** means the sender will be blocked until the message is received by receiver. Similarly, **blocking receive** has the receiver block until a message is available. Non-blocking is considered **asynchronous** and Non-blocking send has the sender sends the message and continue. Similarly, Non-blocking receive has the receiver receive a valid message or null. After a careful analysis, we can come to a conclusion that, for a sender it is more natural to be non-blocking after message passing as there may be a need to send the message to different processes But the sender expect acknowledgement from receiver in case the send fails. Similarly, it is more natural for a receiver to be blocking after issuing the receive as the information from the received message may be used for further execution but at the same time, if the message send keep on failing, receiver will have to wait for indefinitely. That is why we also consider the other possibility of message passing. There are basically three most preferred combinations:

- Blocking send and blocking receive
- Non-blocking send and Non-blocking receive
- Non-blocking send and Blocking receive (Mostly used)

Process Synchronization

Process Synchronization means sharing system resources by processes in a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions.

Precedence Graph

A precedence graph is a directed graph acyclic graph where edge represents execution order and node represents individual statements of the program code.

For example: consider the following statements;
(S1): a = x + y;
(S2): b = a + z;
(S3): c = x - y;
(S4): w = b + c;

Precedence graph:

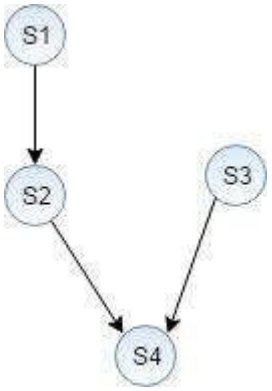
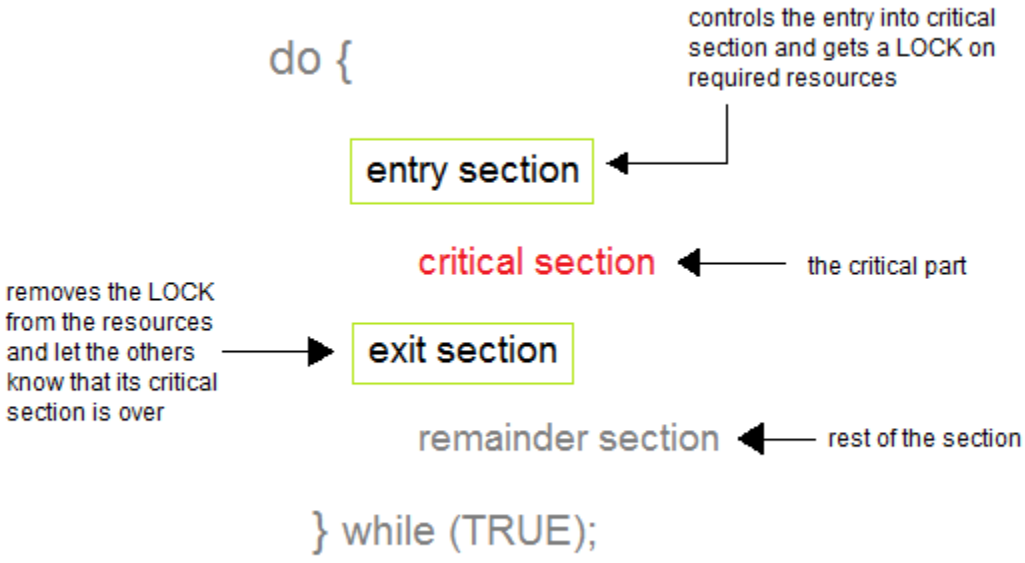


Figure 2.22 Example of Precedence graph:

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress
   If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.
3. Bounded Waiting
   After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

Semaphores

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called semaphore. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

- Wait: decrement the value of its argument S as soon as it would become non-negative.
- Signal: increment the value of its argument, S as an individual operation.

Properties of Semaphores

1. Simple
2. Works with many processes
3. Can have many different critical sections with different semaphores
4. Each critical section has unique access semaphores
5. Can permit multiple processes into the critical section at once, if desirable.

Types of Semaphores

Semaphores are mainly of two types:

1. Binary Semaphore
   It is a special form of semaphore used for implementing mutual exclusion; hence it is often called Mutex. A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.
2. Counting Semaphores
   These are used to implement bounded concurrency.

Limitations of Semaphores

1. Priority Inversion is a big limitation of semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called Deadlock. We will be studying deadlocks in details in coming lessons.

Classical Problem of Synchronization

Following are some of the classical problem faced while process synchronization in systems where cooperating processes are present.

Bounded Buffer (or Producer and Consumer) Problem

We have two processes named as Producer and Consumer. There is finite size of buffer or circular queue with two pointers 'in' and 'out'.

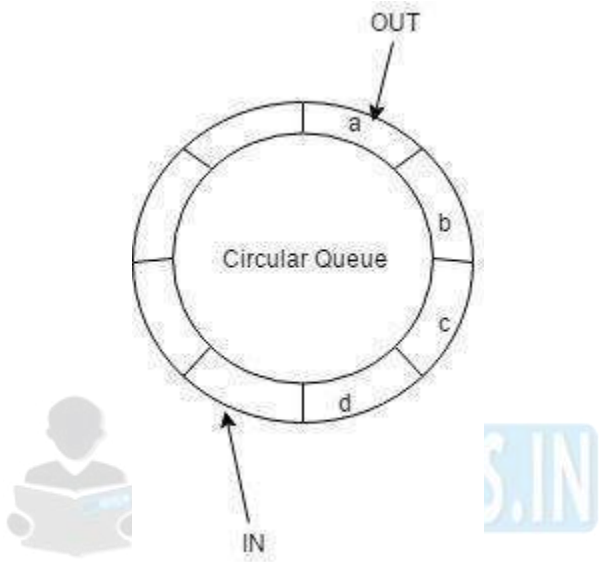


Figure 2.13 Bounded Buffers (or Producer and Consumer) Problem

Producer process produces items and fills into buffer using 'in' pointer and consumer process consumes process from the buffer using 'out' pointer. We must maintain count for empty and full buffers. Also, there should be synchronization between producing items and consuming items otherwise processes go to sleep an item is available to consume or a slot is available to put item.

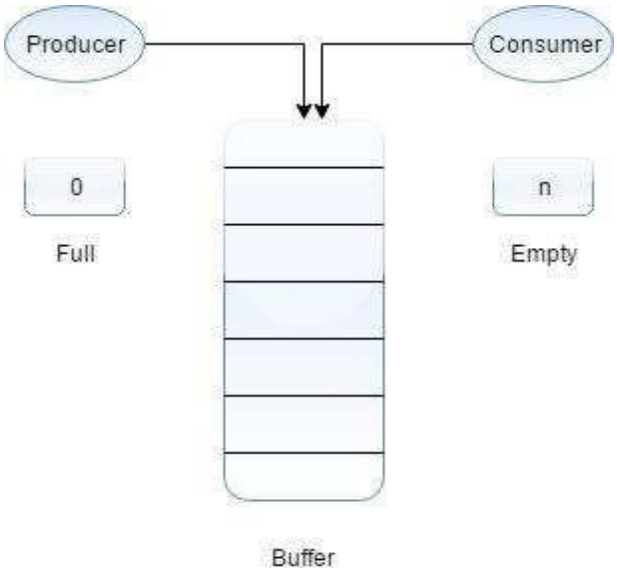


Figure 2.14 Bounded Buffers (or Producer and Consumer) Problem

Producer consumer problem can be solved using semaphores or monitors. Here we see solution using semaphores. There are three semaphores: full, empty, and mutex. 'Full' counts number of full slots, 'empty' counts number of empty slots, and 'mutex' enforces mutual exclusion condition.

Shared Data:
Semaphore mutex = 1 ; /* for mutual exclusion */
Semaphore full = 0 ; /* counts number of full slots, initially none */
Semaphore empty = n ; /* counts number of empty slots, initially all */

Producer Process:

```
Producer ()
{
While (true)
{
Produce_item (item) ; /* produce new item */
P (empty) ; /* decrease number of empty slots */
P (mutex) ; /* enter into critical section */
Add_item (item) ; /* item added in the buffer */
V (mutex) ; /* end the critical section */
V (full) ; /* increase number of full slots */
}
}
```

Consumer Process:

```
Consumer ()
{
while()
{
P (full) ; /* decrease number of full slots */
P (mutex) ; /* enter into critical section */
Remove_item () ; /* item removed from the buffer */
V (mutex) ; /* end the critical section */
V (empty) ; /* increase the number of empty slots */
}
}
```

Mutual exclusion is enforced by 'mutex' semaphore. Synchronization is enforced using 'full' and 'empty' semaphores that avoid race around condition and deadlock in the system.
If we do not use these three semaphores in the producer and consumer problem then there will be race around, deadlock situation and loss of data.

Readers Writers Problem
If we have shared memory or resources, then readers - writers can create conflict due to these combinations: writer - writer and writer- reader access critical section simultaneously that create synchronization problem, loss of data etc. We use semaphores to avoid these problems in reader writer problem.

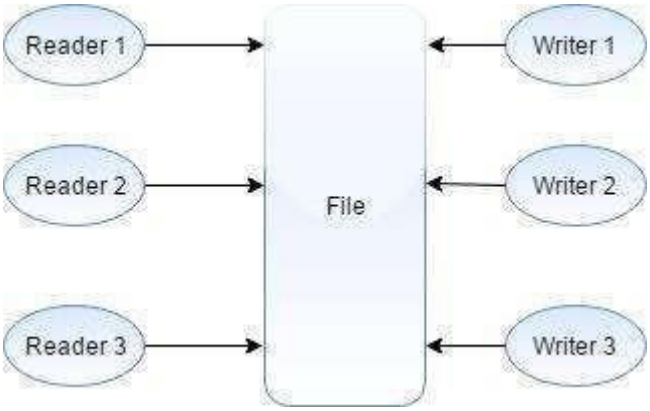


Figure 2.15 Readers Writers Problem

We follow some concepts using semaphore to solve readers - writer's problem: reader and writer cannot enter into critical section at same time. Multiple readers can access critical section simultaneously but multiple writer cannot access.

```
Shared Data:
Semaphore mutex = 1 ;
Semaphore wsem = 1 ;
intreadcount = 0 ;
```

```
Reader Process:
Reader ()
{
While (true)
{
Wait (mutex) ;
Readcount ++ ;
If (readcount = = 1)
Wait (wsem) ;
Signal (mutex) ;
// Critical Section () ;
Wait (mutex) ;
Readcount - - ;
If (readcount = = 0)
Signal (wsem) ;
Signal (mutex) ;
}
}
```

```
Writer Process:
Writer ()
{
While (true)
{
Wait (wsem) ;
// Critical Section () ;
Signal (wsem) ;
}
}
```

Semaphore 'mutex' ensures mutual exclusion property. Readcount and wsem ensures that there is no conflict in the critical section problem. These avoids race around, loss of data, and synchronization problem.

Dining Philosophers Problem

There are n numbers of philosophers meeting around a table, eating spaghetti and talking about philosophy. There are only n forks are available and each philosopher needs 2 forks to eat. Only one fork is available between each philosopher. Now we have design algorithm that ensures maximum number of philosophers can eat at once and none starves as long as each philosopher eventually stop eating.

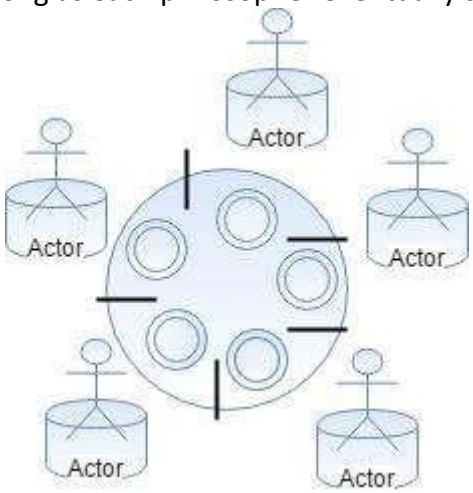


Figure 2.16 Dining Philosophers Problem

```
N = 5 ;  /* total number of philosophers */
Right (i) = (i + 1) mod n ;
Left (i) = ((i = = n) ? 0 : (i + 1))
Philosopher_state[] = {thinking, hungry, eating}
```

```
Semaphore mutex = 1 ;
Semaphore S[n] ;  /*one per philosopher, all 0 initially*/
Philosopher (int process)
{
while(true)
{
Think () ;
Get_forks (process) ;
Eat () ;
Put_forks (process) ;
}
}
Test (int i)
{
If (state[i] = hungry)&&(state[left(i)] != eating)&&(state[right(i) != eating])
{
State[i] = eating ;
V(S[i]) ;
}
}
Get_forks(int i)
{
P(mutex)  ;
State[i] = hungry ;
Test [i] ;
V (mutex) ;
P(S[i])
}
Put_forks(int i)
{
P(mutex) ;
State [i] = thinking ;
Test (left(i)) ;
Test (right(i)) ;
V (mutex) ;
}
```

Unit 3

A process in operating systems uses different resources and uses resources in following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

Deadlock:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.

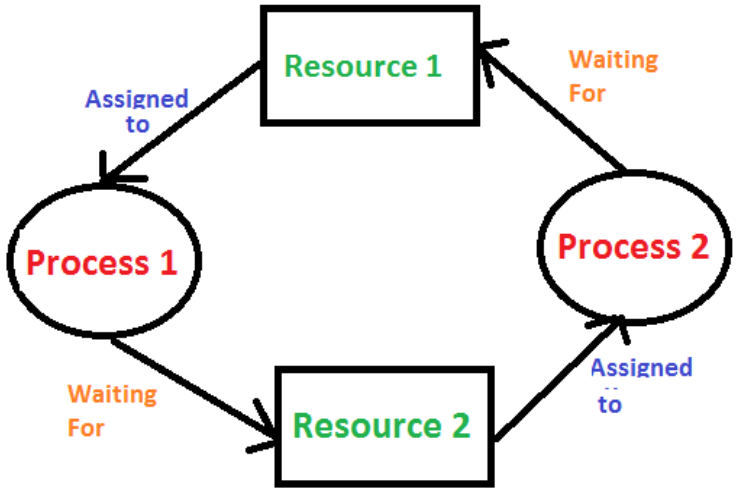


Figure 3.1 Deadlock Example

Deadlock characterization (Necessary Conditions)

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)

- **Mutual Exclusion:** One or more than one resource is non-sharable (Only one process can use at a time)
- **Hold and Wait:** A process is holding at least one resource and waiting for resources.
- **No Pre-emption:** A resource cannot be taken from a process unless the processes release the resource
- **Circular Wait:** A set of processes are waiting for each other in circular form.

Resource Allocation Graph (RAG).

We know that any graph contains vertices and edges. So, RAG also contains vertices and edges. In RAG vertices are two types –

1. Process vertex – Every process will be represented as a process vertex. Generally, the process will be represented with a circle.

2. Resource vertex – Every resource will be represented as a resource vertex. It is also two types –

- **Single instance type resource –** It represents as a box, inside the box, there will be one dot. So, the number of dots indicates how many instances are present of each resource type.
- **Multi-resource instance type resource –** It also represents as a box, inside the box, there will be many dots present.

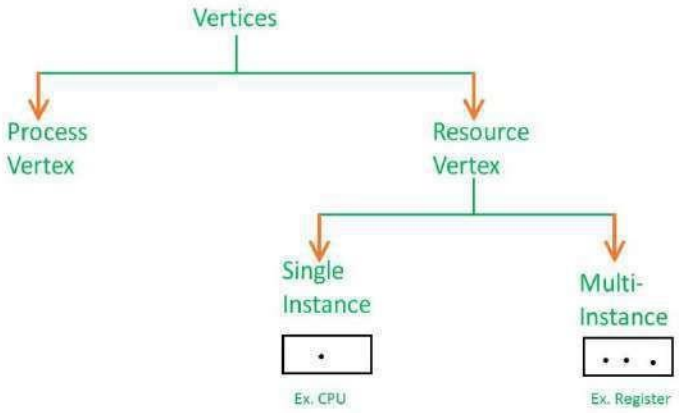


Figure 3.2 Types of Resource

Now coming to the edges of RAG, there are two types of edges in RAG –

1. **Assign Edge –** If you already assign a resource to a process then it is called Assign edge.
2. **Request Edge –** It means in future the process might want some resource to complete the execution that is called request edge.

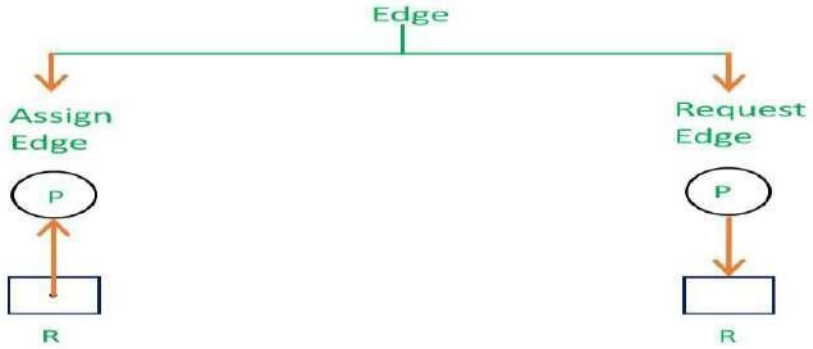


Figure 3.3 Types of Edges in RAG

So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

Example 1 (Single instances RAG) –

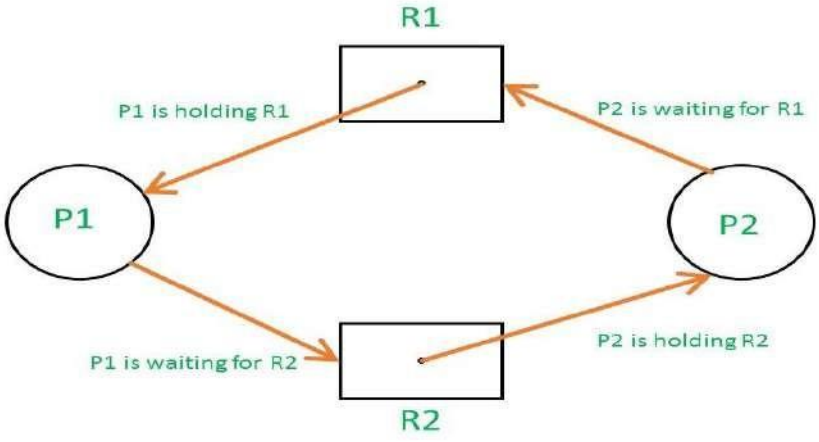


Figure 3.4 Single instance resource types with deadlock

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.

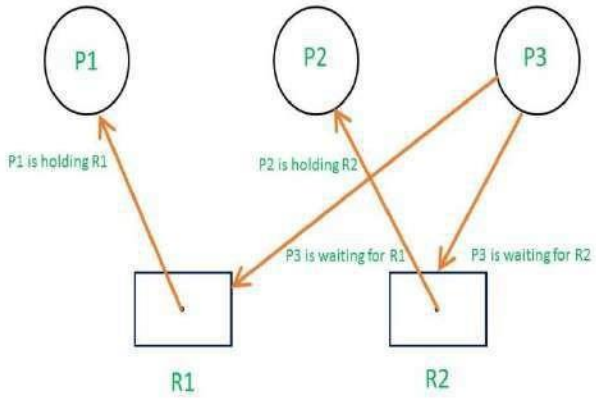


Figure 3.5 Single instance resource types without deadlock

Here's another example that shows Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this example, there is no deadlock because there is no circular dependency.
So cycle in single-instance resource type is the sufficient condition for deadlock.

Example 2 (Multi-instances RAG) –

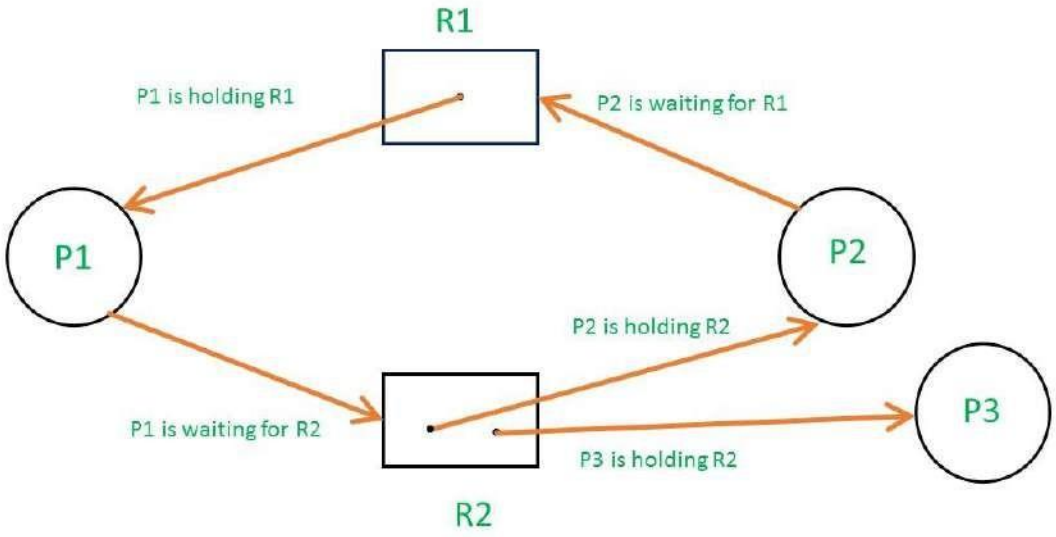


Figure 3.6 Multi instance without deadlock

Methods for handling deadlock

There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.
2) Deadlock detection and recovery: Let deadlock occur, then do pre-emption to handle it once occurred.
3) Ignore the problem all together: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

Deadlock Prevention

Deadlock prevention algorithms ensure that at least one of the necessary conditions (Mutual exclusion, hold and wait, no pre-emption and circular wait) does not hold true. However, most prevention algorithms have poor resource utilization, and hence result in reduced throughputs.

- **Mutual Exclusion**

Not always possible to prevent deadlock by preventing mutual exclusion (making all resources shareable) as certain resources are cannot be shared safely.

- **Hold and Wait:** We will see two approaches, but both have their disadvantages -

A resource can get all required resources before it starts execution. This will avoid deadlock but will result in reduced throughputs as resources are held by processes even when they are not needed. They could have been used by other processes during this time.
Second approach is to request for a resource only when it is not holing any other resource. This may result in a starvation as all required resources might not be available freely always.

- **No pre-emption**

We will see two approaches here. If a process request for a resource which is held by another waiting resource, then the resource may be pre-empted from the other waiting resource. In the second approach, if a process request for a resource which are not readily available, all other resources that it holds are pre-empted.
The challenge here is that the resources can be pre-empted only if we can save the current state can be saved and processes could be restarted later from the saved state.

- **Circular wait**

To avoid circular wait, resources may be ordered, and we can ensure that each process can request resources only in an increasing order of these numbers. The algorithm may itself increase complexity and may also lead to poor resource utilization.

Deadlock Avoidance

This requires that the system has some information available up front. Each process declares the maximum number of resources of each type which it may need. Dynamically examine the resource allocation state to ensure that there can never be a circular-wait condition.
The system's resource-allocation state is defined by the number of available and allocated resources, and the maximum possible demands of the processes. When a process requests an available resource, the system must decide if immediate allocation leaves the system in a *safe state*.
The system is in a safe state if there exists a safe sequence of all processes:
Sequence < P1, P2, Pn> is safe for the current allocation state if, for each P*i*, the resources which P*i* can still request can be satisfied by

- The currently available resources plus
- The resources held by all of the P*j*'s, where *j*<*i*.

If the system is in a safe state, there can be no deadlock. If the system is in an unsafe state, there is the *possibility* of deadlock.

Deadlock Avoidance Algorithms

Two deadlock avoidance algorithms:

- Resource-allocation graph algorithm
- Banker's algorithm

Resource-allocation graph algorithm

- Only applicable when we only have 1 instance of each resource type
- Claim edge (dotted edge), like a *future* request edge
- When a process requests a resource, the claim edge is converted to a request edge
- When a process releases a resource, the assignment edge is converted to a claim edge
- Cycle detection: $O(n^2)$

Banker's Algorithm

- A classic deadlock avoidance algorithm
- More general than resource-allocation graph algorithm (handles multiple instances of each resource type), but
- Is less efficient

Resource-allocations graphs for deadlock avoidance

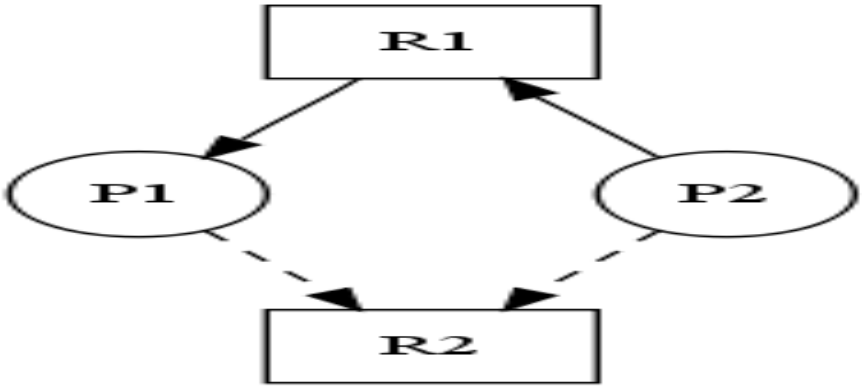


Figure 3.7 Resource-allocations graphs for deadlock avoidance

Banker's Algorithm in Detail

We call Banker's algorithm when a request for R is made. Let n be the number of processes in the system, and m be the number of resource types.

Define:

- Available[m]: the number of units of R currently unallocated (e.g., available [3] = 2)
- Max[n][m]: describes the maximum demands of each process (e.g., max [3][1] = 2)
- Allocation[n][m]: describes the current allocation status (e.g., allocation [5][1] = 3)
- Need[n][m]: describes the *remaining* possible need (i.e., need[i][j] = max[i][j] - allocation[i][j])

Resource-request algorithm:

Define:

- Request[n][m]: describes the current outstanding requests of all processes (e.g., request [2][1] = 3)

1. If request[i][j] <= need[i][j], go to step 2; otherwise, raise an error condition.
2. If request[i][j] > available[j], then the process must wait.
3. Otherwise, *pretend* to allocate the requested resources to p*i* :
   Available[j] = available[j] - request[i][j]
   Allocation[i][j] = allocation[i][j] + request[i][j]
   Need[i][j] = need[i][j] - request[i][j]
   Once the resources are allocated, check to see if the system state is safe. If unsafe, the process must

wait, and the old resource-allocated state is restored.

Safety algorithm (to check for a safe state):

1. Let work be an integer array of length m, initialized to available. Let finish be a Boolean array of length n, initialized to false.
2. Find an i such that both:
   - finish[i] == false
   - need[i] <= work

   If no such i exists, go to step 4
3. work = work + allocation[i]; finish [i] = true; Go to step 2
4. If finish[i] == true for all i, then the system is in a safe state, otherwise unsafe.

Run-time complexity: $O(m \times n^2)$.

Example: consider a system with 5 processes (P0 ... P4) and 3 resources types (A (10) B (5) C (7))

Resource-allocation state at time t0:

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	3	3	2
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Table 3.1

Above given table is in safe state and sequence for execution is:

< P1, P3, P4, P2, P0 >

Now suppose, P1 requests an additional instance of A and 2 more instances of type C

Request [1] = (1, 0, 2)

1. Check if request [1] <= need[i] (yes)
2. Check if request [1] <= available[i] (yes)
3. Do pretend updates to the state

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	3	3	2
P1	3	0	2	3	2	2	0	2	0			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Table 3.2

Above given table is in safe state and sequence for execution is:

<P1, P3, P4, P0, P2>

Hence, we immediately grant the request.

Deadlock Detection

- Requires an algorithm which examines the state of the system to determine whether a deadlock has occurred
- Requires overhead
  - Run-time cost of maintaining necessary information and executing the detection algorithm
  - Potential losses inherent in recovering from deadlock

Single instance of each resource type

- Wait-graph
- P*i* → p*j* = p*i*>r*q* and r*q* → p*j*
- Detect cycle: ***o*(*n*²)**
- Overhead: maintain the graph + invoke algorithm

Resource-allocations graphs for deadlock detection

Resource-allocation graph:

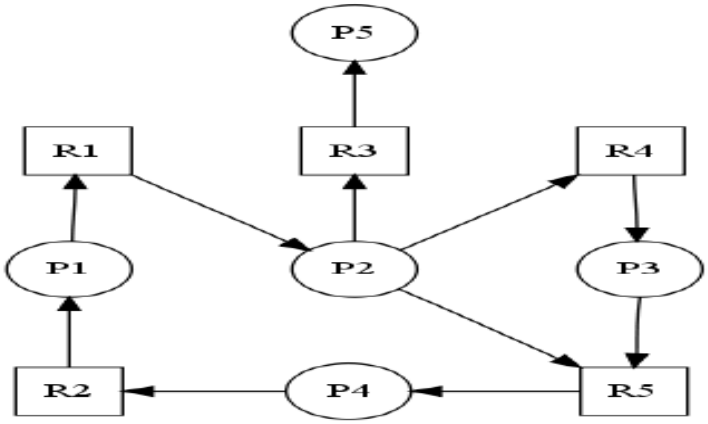


Figure 3.8 Resource-allocations graphs for deadlock detection

Corresponding wait-for graph:

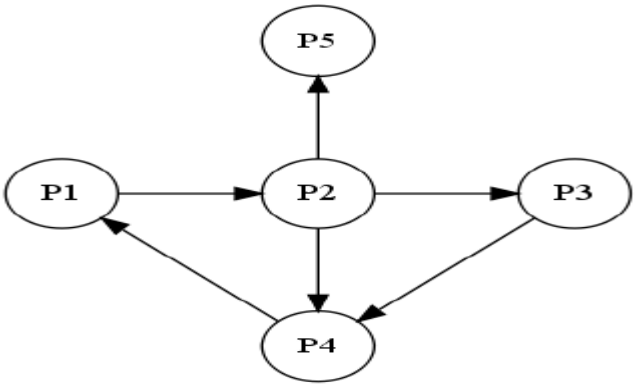


Figure 3.9 wait-for graph for deadlock detection

Multiple instances of a resource type: use an algorithm similar to Banker's, which simply investigates *every* possible allocation sequence for the processes which remain to be completed.

Define:

- Available[m]
- Allocation[n][m]
- Request[n][m]

with their usual semantics.

Algorithm:

1. Let work be an integer array of length m, initialized to available. Let finish be a Boolean array of length n. For all i,

If allocation[i]! = 0, then finish[i] = false;
Otherwise finish[i] = true.

2. Find an i such that both
   - Finish[i] == false // P*i* is currently *not* involved in a deadlock
   - Request[i] <= work

   If no such i exists, go to step 4
3. Reclaim the resources of process P*i*work = work + allocation[i]; finish[i] = true; Go to step 2
4. If finish [i] == false for some i, Then the system is in a deadlocked state. Moreover, if finish[i] == false, then process P*i* is deadlocked.

Run-time complexity: $O(m \times n^2)$.

Example: consider a system with 5 processes (P0 ... P4) and 3 resources types (A (7) B (2) C (6))

Resource-allocation state at time t0:

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Table 3.3

Above given table is in safe state and sequence for execution is:

< P0, P2, P3, P1, P4 >

Now suppose, P2 requests an additional instance of C:

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Table 3.4

We can reclaim the resources held by P0, the number of available resources is insufficient to fulfil the requests of the other processes.

Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

When should we invoke the detection algorithm? Depends on:

- How *often* is a deadlock likely to occur
- How *many* processes will be affected by deadlock when it happens

If deadlocks occur frequently, then the algorithm should be invoked frequently.

Deadlocks only occur when some process makes a request which cannot be granted (if this request is completes a chain of waiting processes).

- **Extreme:** invoke the algorithm every time a request is denied

- **Alternative**: invoke the algorithm at less frequent time intervals:
    - Once per hour
    - Whenever CPU utilization < 40%
    - Disadvantage: cannot determine exactly which process 'caused' the deadlock

Deadlock Recovery

How to deal with deadlock:

- Inform operator and let them decide how to deal with it manually
- Let the system recover from the deadlock automatically:
    - Abort or more of the deadlocked processes to break the circular wait
    - Pre-empt some resources from one or more of the processes to break the circular wait

1. Process termination

Aborting a process is not easy; involves clean-up (e.g., file, printer).

- Abort all deadlocked processes (disadvantage: wasteful)
- Abort one process at a time until the circular wait is eliminated
    - Disadvantage: lot of overhead; must re-run algorithm after each kill
    - How to determine which process to terminate? Minimize cost
        - Priority of the process
        - How long has it executed? How much more time does it need?
        - How many and what type of resources has the process used?
        - How many more resources will the process need to complete?
        - How many processes will need to be terminated?
        - Is the process interactive or batch?

2. Resource pre-emption

Incrementally pre-empt and re-allocate resources until the circular wait is broken.

- Selecting a victim (see above)
- Rollback: what should be done with process which lost the resource? Clearly it cannot continue; must rollback to a safe state (???) => total rollback
- Starvation: pick victim only small (finite) number of times; use number of rollbacks in decision

Method for handling deadlock:

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection

Introduction to Memory management:

Memory management is the functionality of an operating system, which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each memory location, regardless of it allocated to some process or it is free. It checks how much memory is to allocate to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory

allocation to the program. There are three types of addresses used in a program before and after memory allocated −

S.N.	Memory Addresses & Description
1	Symbolic addresses The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	Relative addresses At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	Physical addresses The loader generates these addresses at the time when a program is loaded into main memory.

Address binding

Memory consists of large array of words or arrays, each of which has address associated with it. Now the work of CPU is to fetch instructions from the memory-based program counter. Now further these instructions may cause loading or storing to specific memory address.

Address binding is the process of mapping from one address space to another address space. Logical address is address generated by CPU during execution whereas Physical Address refers to location in memory unit (the one that is loaded into memory). Note that user deals with only logical address (Virtual address). The logical address undergoes translation by the MMU or address translation unit. The output of this process is the appropriate physical address or the location of code/data in RAM.

An address binding can be done in three different ways:

Compile Time – It work is to generate logical address (also known as virtual address). If you know that during compile time where process will reside in memory, then absolute address is generated.
Load time – It will generate physical address. If at the compile time it is not known where process will reside then relocatable address will be generated. In this if address changes then we need to reload the user code.
Execution time- It helps in differing between physical and logical address. This is used if process can be moved from one memory to another during execution (dynamic linking-Linking that is done during load or run time).

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.
The set of all logical addresses generated by a program referred to as a logical address space the set of all-physical addresses corresponding to these logical addresses referred to as physical address space

- The value in the base register added to every address generated by a user process, which treated as offset at the time it sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Logical and Physical Address space: Logical Address is address generated by CPU while a program is running. Logical address is also called virtual address as it does not exist physically. The set of all logical address space is called logical address space. Logical address is mapped to corresponding physical address using MMU (Memory Management Unit).
Compile time and load time address binding produces same logical and physical address, while run time address binding produces different.

Physical Address is the actual address location in memory unit. This is computed by MMU. User can access physical address in the memory unit using the corresponding logical address.

Static vs. Dynamic Loading

The choice between Static or Dynamic Loading is to make at the time of computer program developed. If you have to load your program statically, then at the time of compilation, the complete programs compiled, linked without leaving any external program or module dependency the linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a dynamically loaded program, then your compiler will compile the program and for all the modules, which you want to include dynamically, only references provided and rest of the work done at the time of execution

At the time of loading, with static loading, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using dynamic loading, dynamic routines of the library are stored on a disk in re-locatable form and are loaded into memory only when they are needed by the program

Static vs. Dynamic Linking

As explained above, when static linking is to use, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in UNIX are good examples of dynamic libraries.

MMU

before discussing memory allocation further, we must discuss the issue of memory mapping and protection. When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Because every address generated by the CPU is checked against these registers, we can protect both the operating system and the other users' programs and data from being modified by this running process.

The relocation-register scheme provides an effective way to allow the operating-system size to change dynamically. This flexibility is desirable in many situations. For example, the operating system contains code and buffer space for device drivers. If a device driver [or other operating-system service] is not commonly used, we do not want to keep the code and data in memory, as we might be able to use that space for other purposes. Such code is sometimes called transient operating-system code; it comes and goes as needed. Thus, using this code changes the size of the operating system during program execution.

To protect the operating system code and data by the user processes as well as protect user processes from one another using relocation register and limit register.

This is depicted in the figure below:

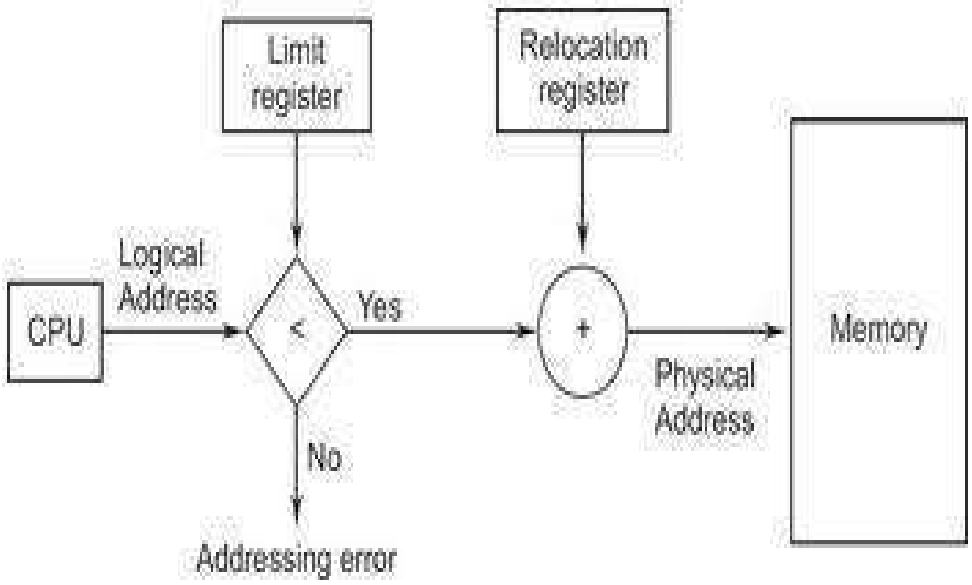


Figure 3.10 MMU

Contiguous Memory Allocation

In Contiguous Memory allocation strategies, the Operating System allocates memory to a process that is always in a sequence for faster retrieval and less overhead, but this strategy supports static memory allocation only. This strategy is inconvenient in the sense that there are more instances of internal memory fragmentation than compared to Contiguous Memory Allocation strategies. The Operating System can also allocate memory dynamically to a process if the memory is not in sequence; i.e. they are placed in non–contiguous memory segments. Memory is allotted to a process as it is required. When a process no longer needs to be in memory, it is released from the memory to produce a free region of memory or a memory hole. These memory holes and the allocated memory to the other processes remain scattered in memory. The Operating System can compact this memory at a later point in time to ensure that the allocated memory is in a sequence and the memory holes are not scattered. This strategy has support for dynamic memory allocation and facilitates the usage of Virtual Memory. In dynamic memory allocation there are no instances of internal fragmentation.

Memory management Techniques

In operating system, following are four common memory management techniques.

Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

Partitioned allocation: Memory is divided in different blocks

Paged memory management: Memory is divided in fixed sized units called page frames, used in virtual memory environment.

Segmented memory management: Memory is divided in different segments (a segment is logical grouping of process' data or code) In this management, allocated memory doesn't have to be contiguous.

Memory Allocation:

Single Partition Allocation

In this scheme Operating system is residing in low memory and user processes are executing in higher memory.

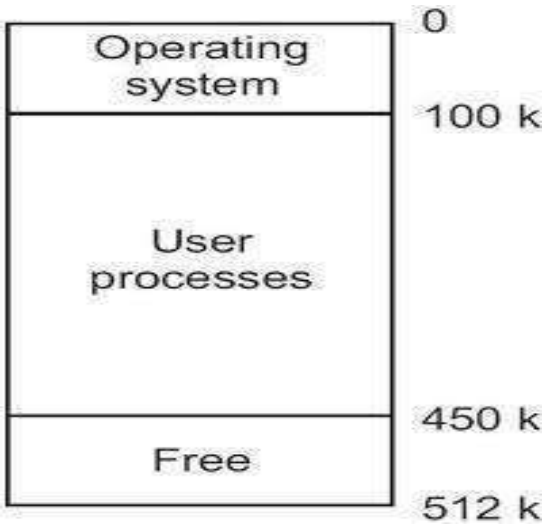


Figure 3.11 Single Partition Allocations

Advantages

- It is simple.
- It is easy to understand and use.

Disadvantages

- It leads to poor utilization of processor and memory.
- User's job is limited to the size of available memory.

Multi-partition Allocation (fixed sized partitions)

One of the simplest methods for allocating memory is to divide memory into several fixed sized partitions. There are two variations of this.

- **Fixed Equal-size Partitions**
  It divides the main memory into equal number of fixed sized partitions, operating system occupies some fixed portion and remaining portion of main memory is available for user processes.
  Advantages
  - Any process whose size is less than or equal to the partition size can be loaded into any available partition.
  - It supports multiprogramming.

  Disadvantages
  - If a program is too big to fit into a partition use overlay technique.
  - Memory use is inefficient, i.e., block of data loaded into memory may be smaller than the partition. It is known as internal fragmentation.
- **Fixed Variable Size Partitions**
  By using fixed variable size partitions, we can overcome the disadvantages present in fixed equal size partitioning. This is shown in the figure below:

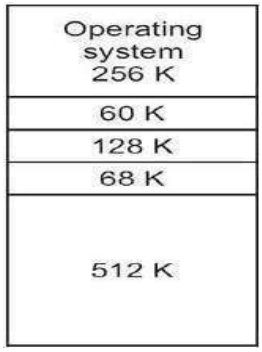


Figure 3.12 Fixed Variable Size Partitions

With unequal-size partitions, there are two ways to assign processes to partitions.

Use multiple queues: - For each and every process one queue is present, as shown in the figure below. Assign each process to the smallest partition within which it will fit, by using the scheduling queues, i.e., when a new process is to arrive it will put in the queue it is able to fit without wasting the memory space, irrespective of other blocks queues.

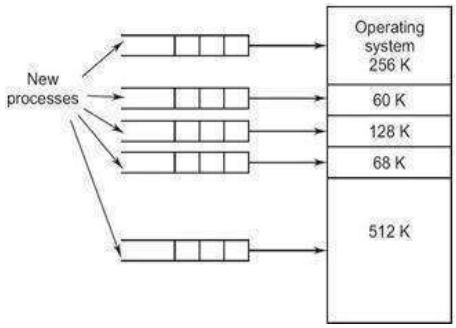


Figure 3.13 Fixed Variable Size Partitions

Advantages

- Minimize wastage of memory.

Disadvantages

- This scheme is optimum from the system point of view. Because larger partitions remains unused.

Use single queue: - In this method only one ready queue is present for scheduling the jobs for all the blocks irrespective of size. If any block is free even though it is larger than the process, it will simply join instead of waiting for the suitable block size. It is depicted in the figure below:

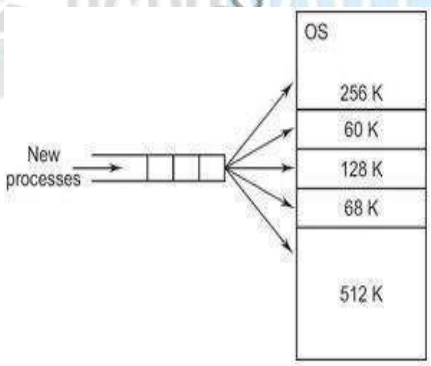


Figure 3.14 Use single queues

Advantages

- It is simple and minimum processing overhead.

Disadvantages

- The number of partitions specified at the time of system generation limits the number of active processes.
- Small jobs do not use partition space efficiently.

Dynamic Partitioning

Even though when we overcome some of the difficulties in variable sized fixed partitioning, dynamic partitioning requires more sophisticated memory management techniques. The partitions used are of variable length. That is when a process is brought into main memory, it allocates exactly as much memory as it requires. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this method when a partition is free a process is selected from the input queue and is loaded into the free partition. When the process terminates the partition becomes available for another process. This method was used by IBM's mainframe operating system, OS/MVT (Multiprogramming with variable number of tasks) and it is no longer in use now.

Let us consider the following scenario:

Code:

Process Size (in kB)	Arrival time (in milli sec)	Service time (in milli sec)
P1	350	0	40
P2	400	10	45
P3	300	30	35
P4	200	35	25

Figure below is showing the allocation of blocks in different stages by using dynamic partitioning method. That is the available main memory size is 1 MB. Initially the main memory is empty except the operating system shown in Figure a. Then process 1 is loaded as shown in Figure b, then process 2 is loaded as shown in Figure c without the wastage of space and the remaining space in main memory is 146K it is free. Process 1 is swaps out shown in Figure d for allocating the other higher priority process. After allocating process 3, 50K whole is created it is called internal fragmentation, shown in Figure e. Now process 2 swaps out shown in Figure f. Process 1 swaps in, into this block. But process 1 size is only 350K, this leads to create a whole of 50K shown in Figure g.

Like this, it creates a lot of small holes in memory. Ultimately memory becomes more and more fragmented and it leads to decline memory usage. This is called 'external fragmentation'. To overcome external fragmentation by using a technique called "compaction". As the part of the compaction process, from time to time, operating system shifts the processes so that they are contiguous and this free memory is together creates a block. In Figure h compaction results in a block of free memory of length 246K.

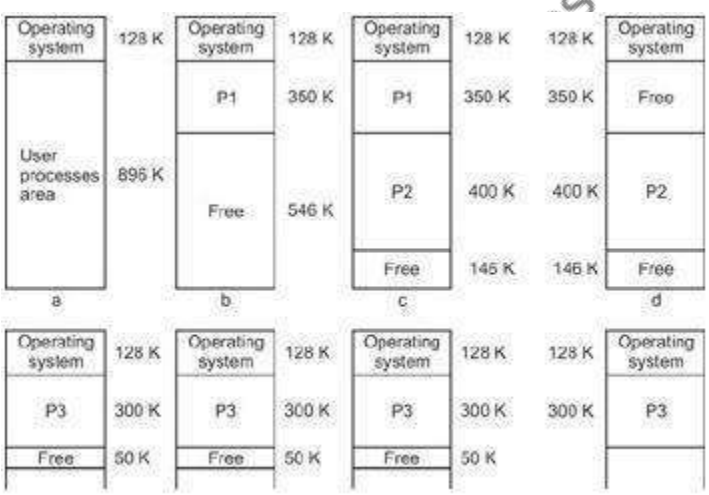


Figure 3.15 Dynamic Partitioning

Advantages

- Partitions are changed dynamically.
- It does not suffer from internal fragmentation.

Disadvantages

- It is a time-consuming process (i.e., compaction).
- Wasteful of processor time, because from time to time to move a program from one region to another in main memory without invalidating the memory references.

Placement Algorithm

If the free memory is present within a partition, then it is called "internal fragmentation". Similarly, if the free blocks are present outside the partition, then it is called "external fragmentation". Solution to the "external fragmentation" is compaction.

Solution to the "internal fragmentation" is the "placement" algorithm only.

Because memory compaction is time-consuming, when it is time to load or swap a process into main memory

and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate by using three different placement algorithms.
This is shown in the figure below:

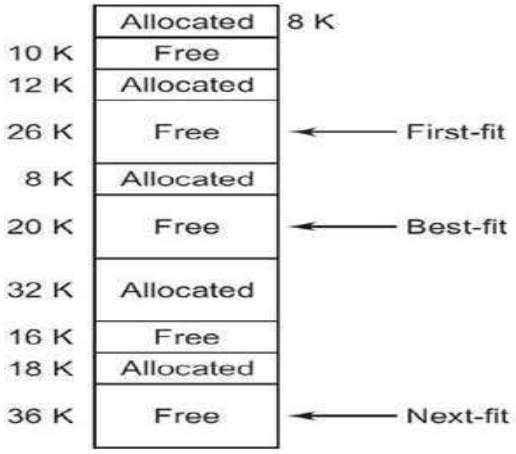


Figure 3.16 Placement Algorithm

- Best-fit: - It chooses the block that is closest in size to the given request from the beginning to the ending free blocks. We must search the entire list, unless it is ordered by size. This strategy produces the smallest leftover hole.
- Worst-fit: - It allocates the largest block. We must search the entire the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.
- First-fit: - It begins to scan memory from the beginning and chooses the first available block which is large enough. Searching can start either at the beginning of the set of blocks or where the previous first-fit search ended. We can stop searching as soon as we find a free block that is large enough.
- Last-fit: - It begins to scan memory from the location of the last placement and chooses the next available block. In the figure below the last allocated block is 18k, thus it starts from this position and the next block itself can accommodate this 20K block in place of 36K free block. It leads to the wastage of 16KB space.

First-fit algorithm is the simplest, best and fastest algorithm. Next-fit produce slightly worse results than the first-fit and compaction may be required more frequently with next-fit algorithm. Best-fit is the worst performer, even though it is to minimize the wastage space. Because it consumes the lot of processor time for searching the block which is close to its size.

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be an allocated to memory blocks considering their small size and memory blocks remains unused this problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory left unused, as it cannot be a used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −

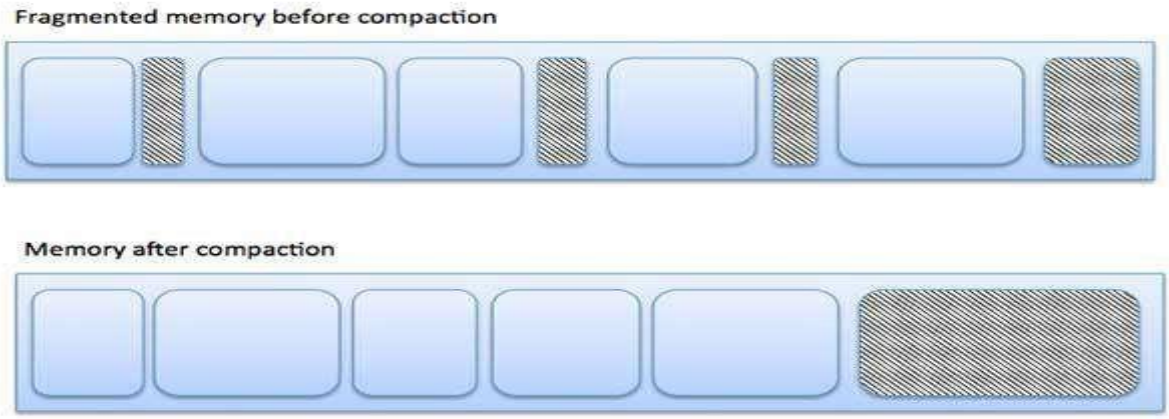


Figure 3.17 Fragmentation

External fragmentation can be a reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = log2 2^{27} = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 * 2^{20}$ words, then Physical Address = log2 2^{24} = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)

Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2

Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3

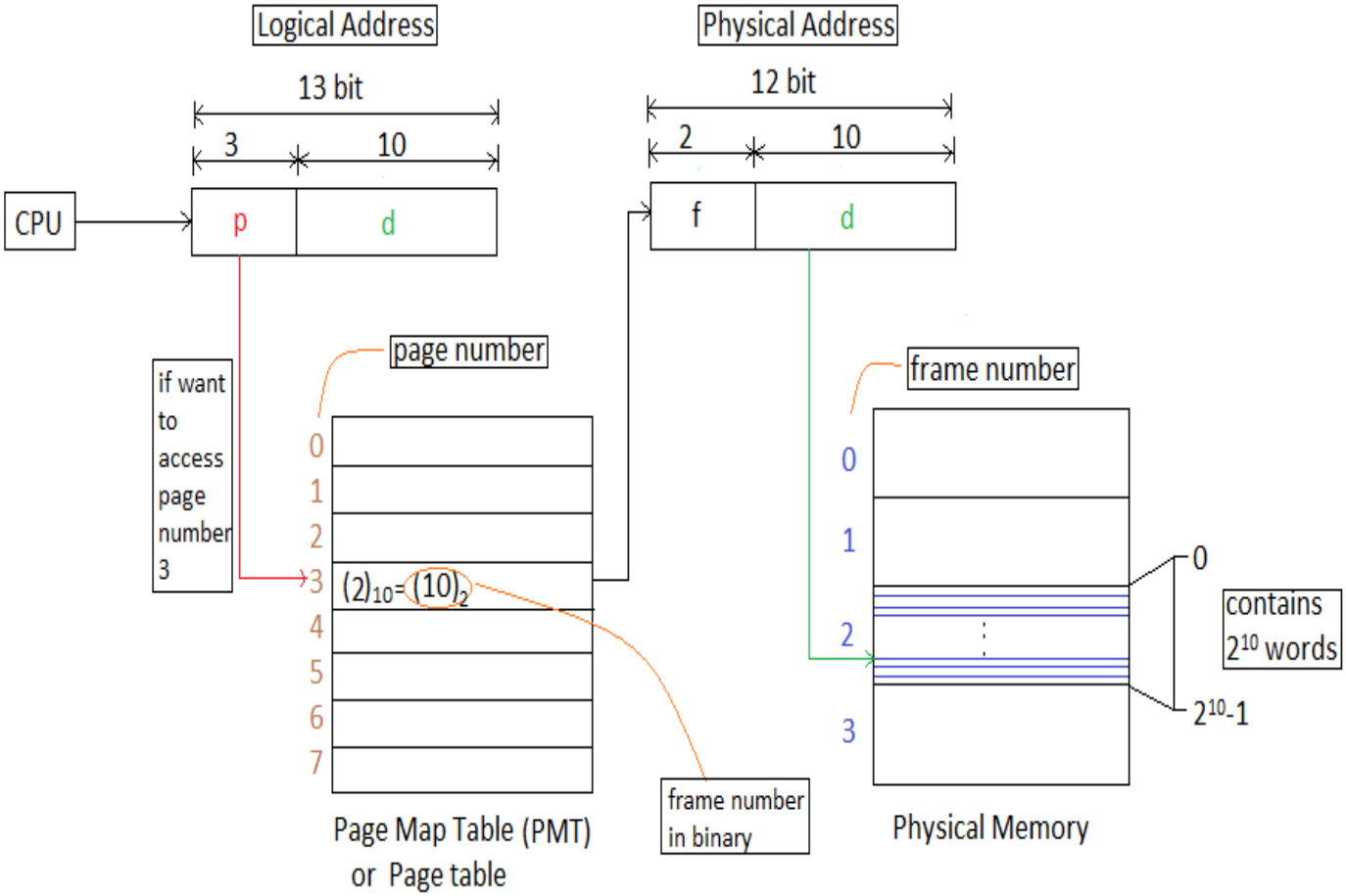


Figure 3.18 Paging

Address generated by CPU is divided into

- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

- **Frame number (f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset (d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Paging Issues:

- Page Table Structure. Where to store page tables?
- Issues in page table design: In a real machine, page tables stored in physical memory. Several issues arise like
  - How much memory does the page table take up?
  - How to manage the page table memory. Contiguous allocation? Blocked allocation?
  - What about paging the page table?
- On TLB misses, OS must access page tables. Issue: how the page table design affects the TLB miss penalty.

- Real operating systems provide the abstraction of sparse address spaces.
- Issue: how well does a particular page table design support sparse address space.

Implementation Issues of Paging

Page Size

- Basic page size is determined by hardware.
- An OS can allocate pages in (small) groups, simulating larger pages for some purposes.
- Most common size is 4K; systems have used 512 bytes to 64K.
- Advantages of larger pages.
    - Smaller page table, and fewer TLB entries needed.
    - For page I/O, less time per byte moved. (I/O operations have a fixed time plus a variable time.)
- Advantages of smaller pages.
    - Reduced internal fragmentation in the last page.
    - For page I/O, less time per page.
- When a reference brings a page into memory, less chance some of its contents is unrelated and not actually needed.

TLB (translation Look - aside buffer)

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB (translation Look - aside buffer), a special, small, fast look up hardware cache.

- The TLB is associative, high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then corresponding value is returned.

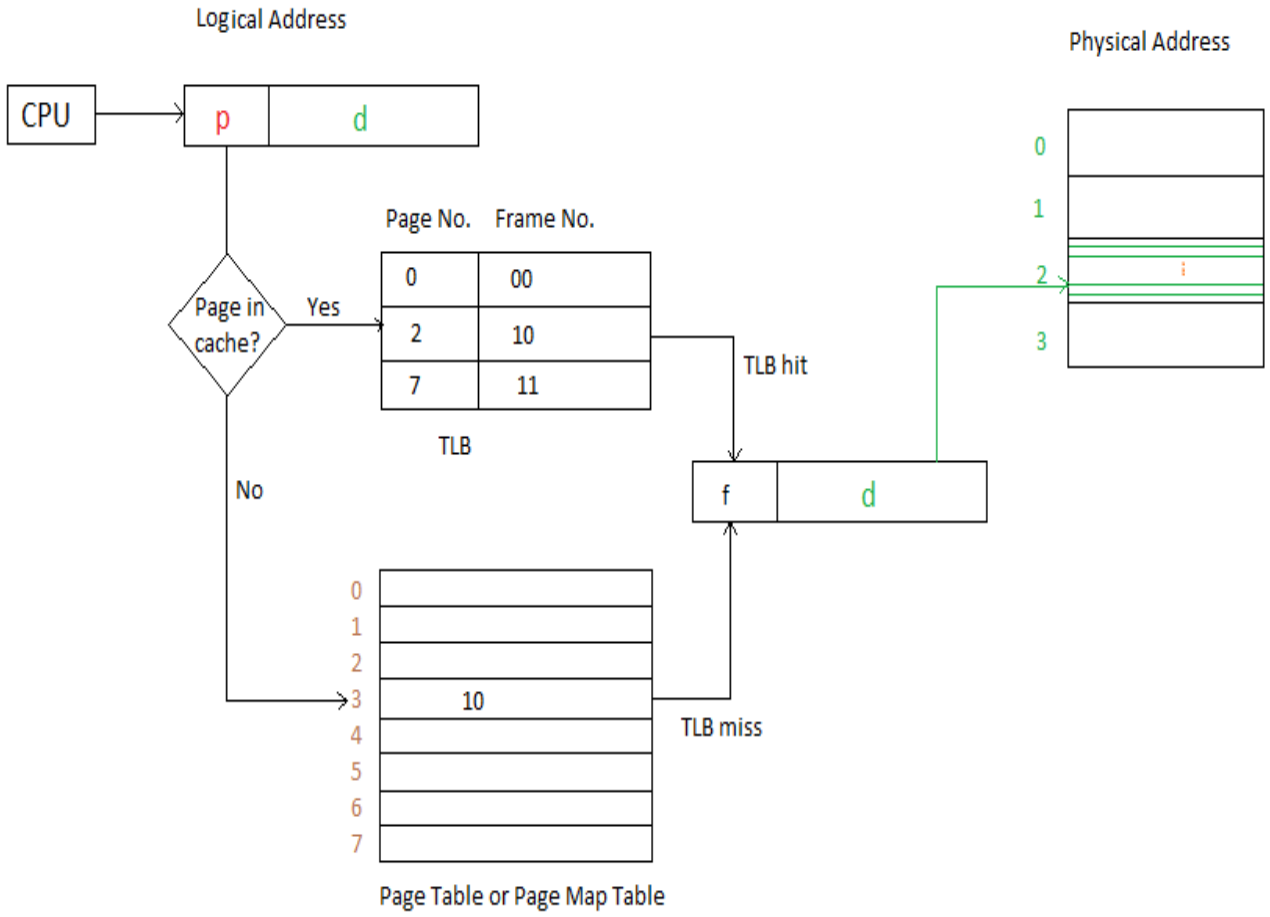


Figure 3.19 TLB

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Page Fault

A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM. Though the term "page fault" sounds like an error, page faults are common and are part of the normal way computers handle virtual memory. In programming terms, a page fault generates an exception, which notifies the operating system that it must retrieve the memory blocks or "pages" from virtual memory in order for the program to continue. Once the data is moved into physical memory, the program continues as normal. This process takes place in the background and usually goes unnoticed by the user.

So when page fault occurs then following sequence of events happens:

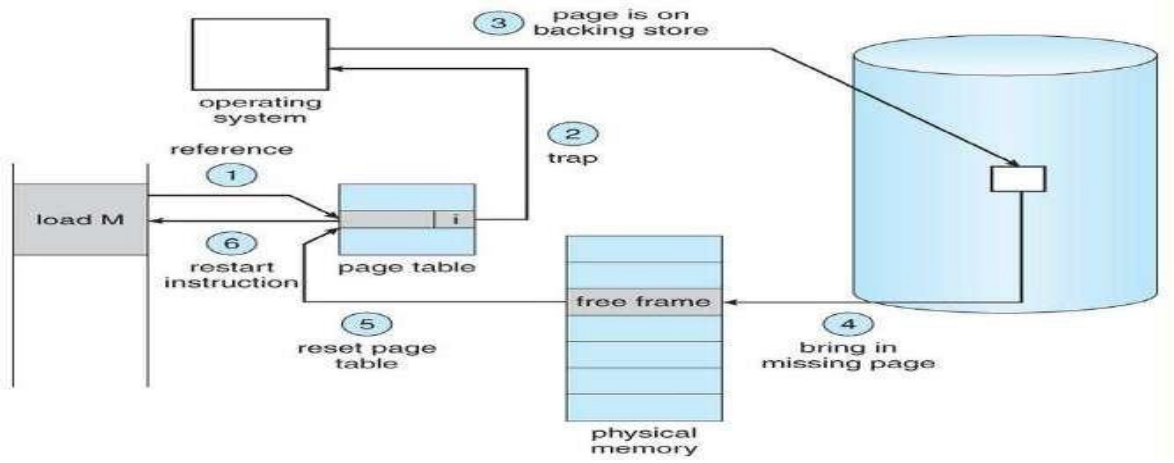


Figure 3.20 Page fault

Segmentation

- Segmentation is a memory management technique in which each job is a divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is a different logical address space of the program.
- When a process is to be an executed, its corresponding segmentation is loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.
- Segmentation memory management works very similar to paging but here segments are of variable-length whereas in paging pages are of fixed size.
- A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

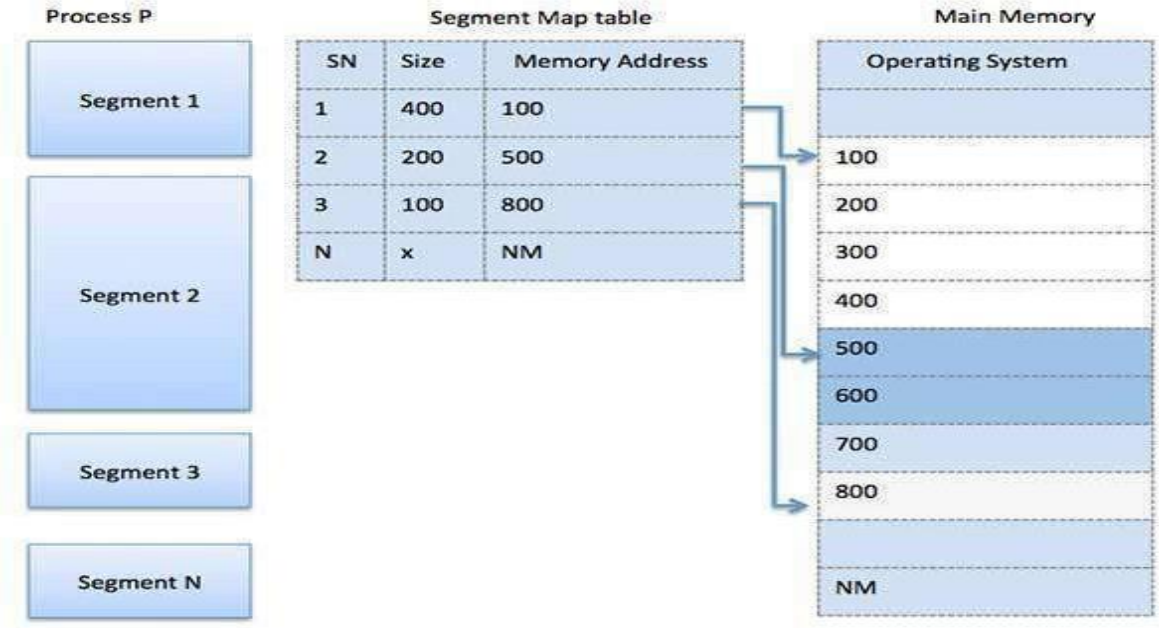


Figure 3.21 Segmentation

Segmentation with paging

Treat virtual address space as a collection of segments (logical units) of arbitrary sizes.

- Treat physical memory as a sequence of fixed size page frames.
- Segments are typically larger than page frames,
  - Map a logical segment onto multiple page frames by paging the segments

Addresses in a Segmented Paging System

- A virtual address becomes a segment number, a page within that segment, and an offset within the page.
- The segment number indexes into the segment table which yields the base address of the page table for that segment.
- Check the remainder of the address (page number and offset) against the limit of the segment.
- Use the page number to index the page table. The entry is the frame. (The rest of this is just like paging.)
- Add the frame and the offset to get the physical address.

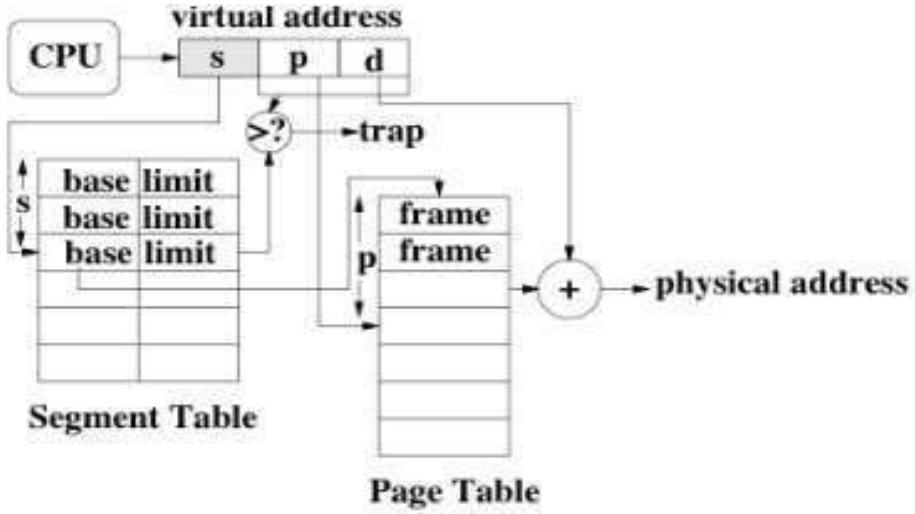


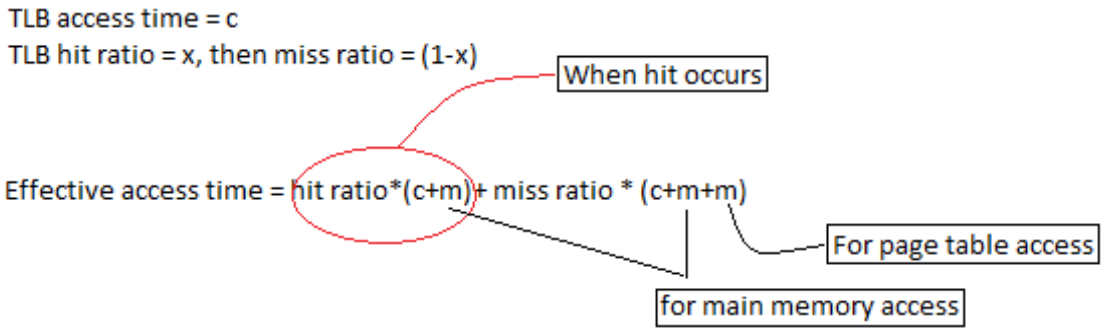
Figure 3.22 Segmented Paging

Effective access time

Main memory access time = m

If page table are kept in main memory,

Effective access time = m(for page table) + m(for particular page in page table)



The percentage of times that the page number of interest is found in the TLB is called the hit ratio. An 80-percent hit ratio, for example, means that we find the desired page number in the TLB 80 percent of the time. If it takes 100 nanoseconds to access memory, then a mapped-memory access takes 100 nanoseconds when the page number is in the TLB. If we fail to find the page number in the TLB then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of 200 nanoseconds. (We are assuming that a page-table lookup takes only one memory access, but it can take more, as we shall see.)

To find the effective memory-access time, we weight the case by its probability: effective access time = 0.80 × 100 + 0.20 × 200 = 120 nanoseconds

Department of Information Technology
Class Notes

Unit 4

Concept of virtual memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory; more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −

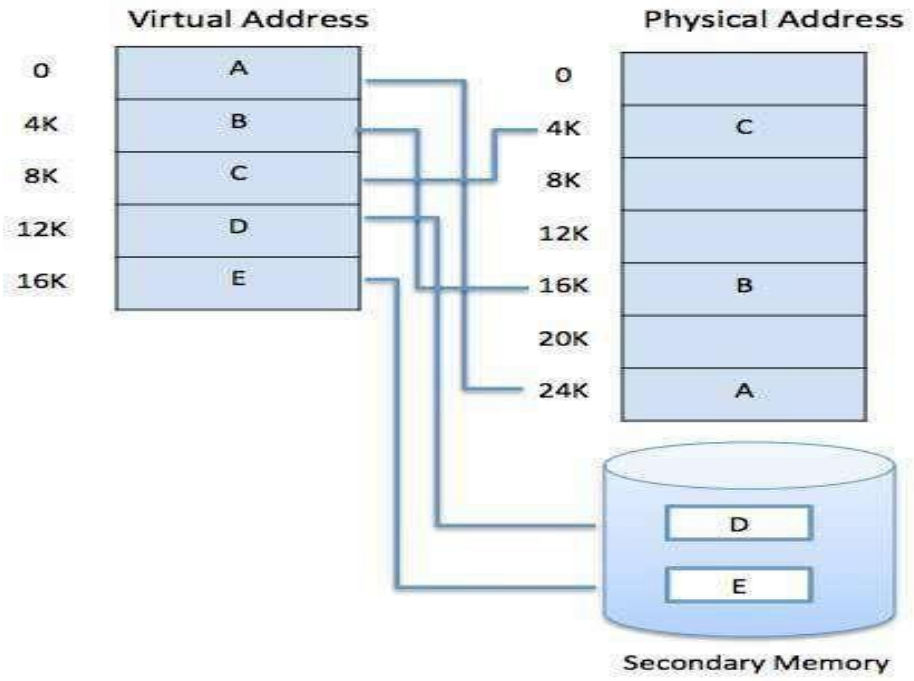


Figure 4.1 MMU

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Cache memory organization

A cache memory is a fast random access memory where the computer hardware stores copies of information currently used by programs (data and instructions), loaded from the main memory. The cache has a significantly shorter access time than the main memory due to the applied faster but more expensive

implementation technology. The cache has a limited volume that also results from the properties of the applied technology. If information fetched to the cache memory is used again, the access time to it will be much shorter than in the case if this information were stored in the main memory and the program will execute faster.

Time efficiency of using cache memories results from the locality of access to data that is observed during program execution.

Time locality: Time locality consists in a tendency to use many times the same instructions and data in programs during neighboring time intervals.

Space locality: Space locality is a tendency to store instructions and data used in a program in short distances of time under neighboring addresses in the main memory.

- Due to these localities, the information loaded to the cache memory is used several times and the execution time of programs is much reduced.
- Cache can be implemented as a multi-level memory. Contemporary computers usually have two levels of caches. In older computer models, a cache memory was installed outside a processor (in separate integrated circuits than the processor itself).
- The access to it was organized over the processor external system bus. In today's computers, the first level of the cache memory is installed in the same integrated circuit as the processor.
- It significantly speeds up processor's co-operation with the cache. Some microprocessors have the second level of cache memory placed also in the processor's integrated circuit.
- The volume of the first level cache memory is from several thousands to several tens of thousands of bytes. The second level cache memory has volume of several hundred thousand bytes.
- A cache memory is maintained by a special processor subsystem called cache controller.

Read implementation in cache memory on hit

If there is a cache memory in a computer system, then at each access to a main memory address in order to fetch data or instructions, processor hardware sends the address first to the cache memory. The cache control unit checks if the requested information resides in the cache. If so, we have a "hit" and the requested information is fetched from the cache. The actions concerned with a read with a hit are shown in the figure below.

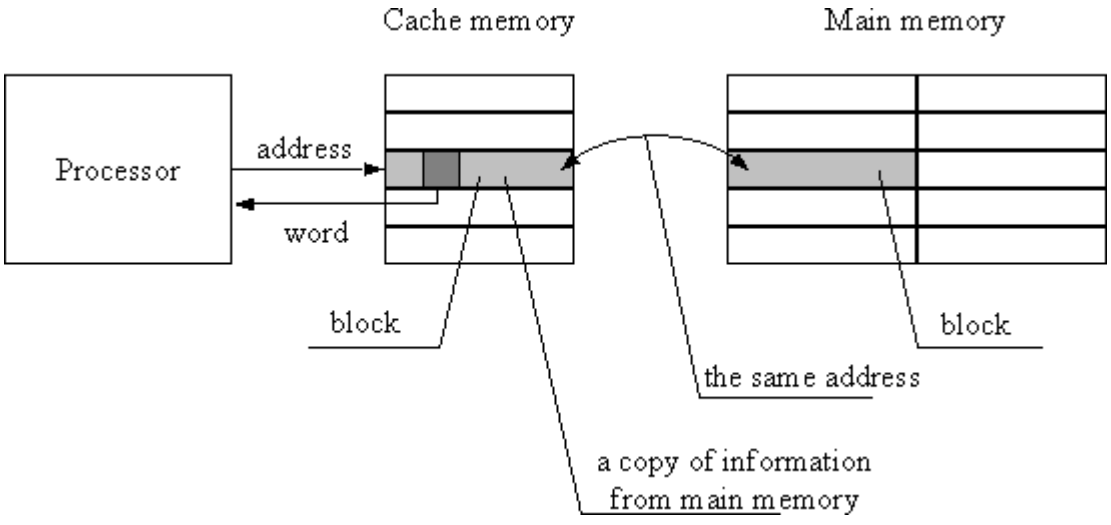


Figure 4.2 Read implementation in cache memory on hit

Read implementation in cache memory on miss

If the requested information does not reside in the cache, we have a "miss" and the necessary information is fetched from the main memory to the cache and to the requesting processor unit. The information is not copied in the cache as single words but as a larger block of a fixed volume. Together with information block, a part of the address of the beginning of the block is always copied into the cache. This part of the address is next used at readout during identification of the proper information block. The actions executed in a cache memory on "miss" are shown below.

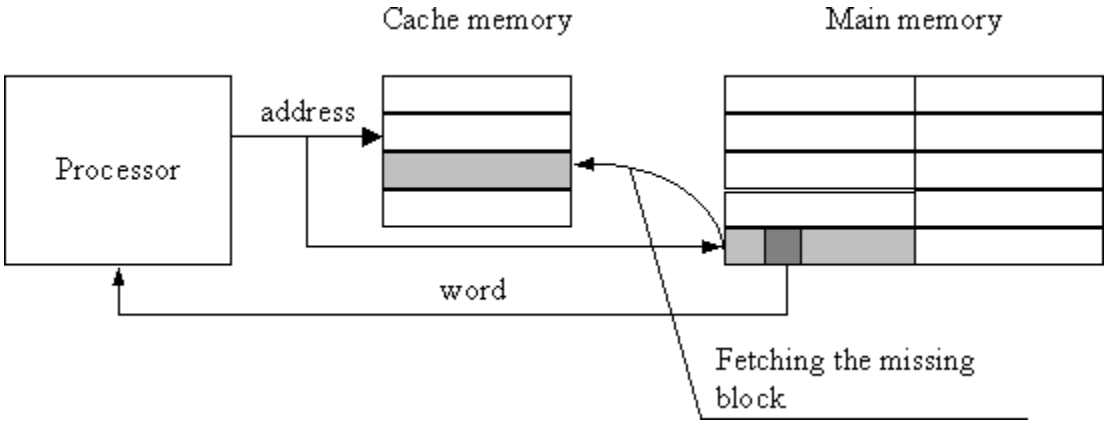


Figure 4.3 Read implementation in cache memory on miss

Memory updating methods after cache modification:

A cache memory contains copies of data stored in the main memory. When a change of data in a cache takes place (ex. a modification due to a processor write) the contents of the main memory and cache memory cells with the same address, are different. To eliminate this lack of data coherency two methods are applied:

- **Write through**, the new cache contents is written down to the main memory immediately after the write to the cache memory,
- **Write back,** the new cache contents are not written down to the main memory immediately after the change, but only when the given block of data is replaced by a new block fetched from the main memory or an upper level cache. After a data write to the cache, only state bits are changed in the modified block, indicating that the block has been modified (a dirty block).

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

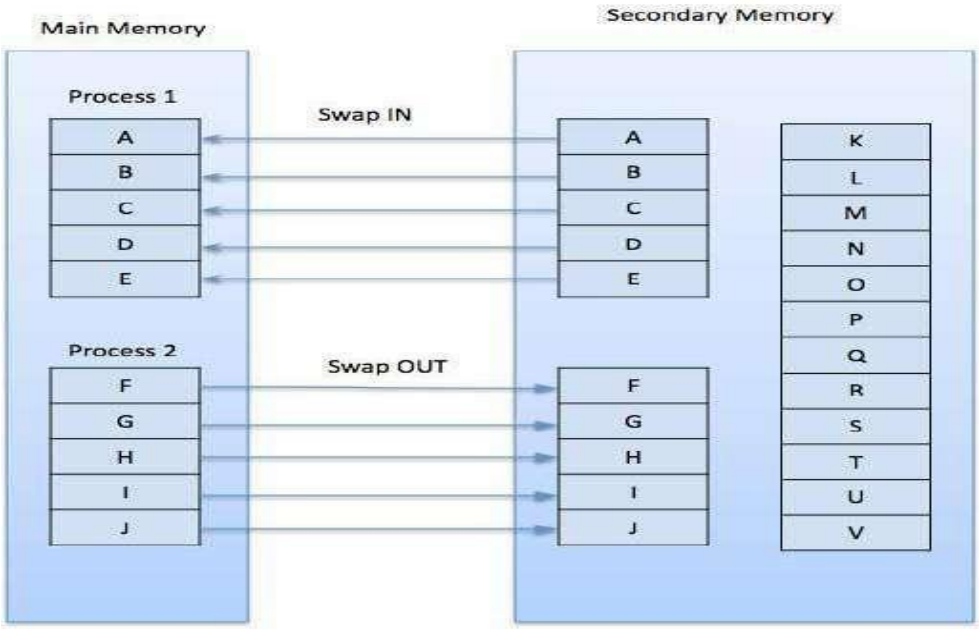


Figure 4.4 Demand Paging

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.

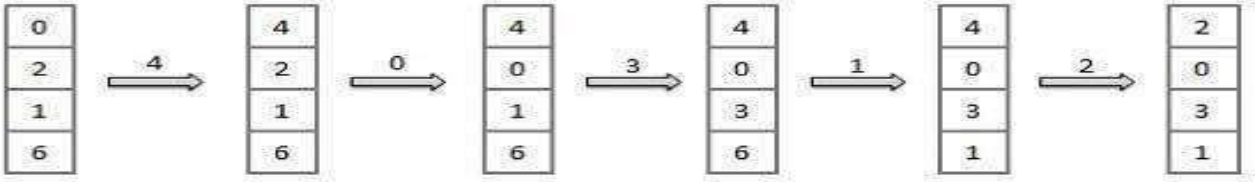
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First in First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



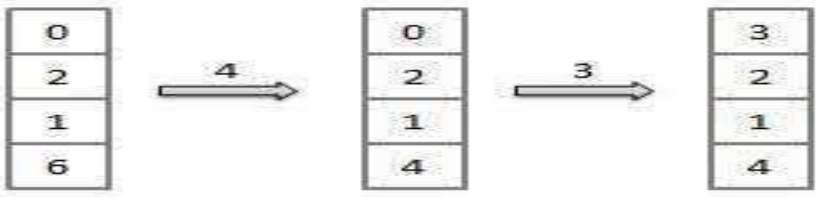
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



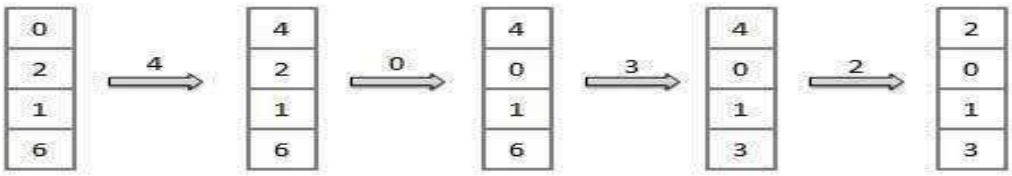
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used (LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used (MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Maintain 3 free frames at all times
- Consider a machine where all memory-reference instructions have only 1 memory address → need 2 frames of memory
    - Now consider indirect modes of addressing
    - Potentially every page in virtual memory could be touched, and the entire virtual memory must be in physical memory
    - Place a limit the levels of indirection
- The minimum number of frames per process is defined by the computer architecture
- The maximum number of frames is defined by the amount of available physical memory

Allocation Algorithms

- m frames, n processes
- Equal allocation: give each process m/n frames
- Alternative is to recognize that various processes will need different amounts of memory
    - Consider
        - 1k frame size
        - 62 free frames
        - student process requiring: 10k
        - interactive database requiring: 127k

It makes no sense to give each process 31 frames the student process needs no more than 10 so the additional 21 frames are wasted

- proportional allocation:
    - m frames available
    - Size of virtual memory for process pi is si

- S = Sigma si
- ai = (si/S) * m

Student process gets 4 frames = (10/137)*62
Database gets 57 frames = (127/137)*62

Global vs. Local Allocation

- **Global**: one process can select a replacement frame from the set of all frames (i.e., one process can take a frame from another or itself) (e.g., high priority processes can take the frames of low priority processes)
- **Local**: each process can only select from its own set of allocated frames
- local page replacement is more predictable; depends on no external factors
- A process which uses global page replacement cannot predict the page fault rate; may execute in 0.5 seconds once and 10.3 on another run
- Overall, global replacement results in greater system throughput

Thrashing

- **simple thrashing**: 1 process of 2 pages only allocated 1 frame
- High page activity is called thrashing
- A process is thrashing if it spends more time paging than executing

Scenario:

- The process scheduler sees that CPU utilization is low.
- So we increase the degree of multiprogramming by introducing a new process into the system.
- One process now needs more frames.
- It starts faulting and takes away frames from other processes. (i.e., global-page replacement).
- These processes need those pages and thus they start to fault, taking frames from other processes.
- These faulting processes must use the paging device to swap pages in and out.
- As they queue up on the paging device, the ready-queue empties.
- However, as processes wait on the paging device, CPU utilization decreases.
- The process scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming, ad infinitum.
- The result is thrashing, page-fault rates increase tremendously, effective memory access time increases, no work is getting done, because all the processes are spending their time paging.
- We can limit the effects of thrashing by using a local replacement algorithm (or priority replacement algorithm)
    - But this only partially solves the problem
      If one process starts thrashing, it cannot steal frames of another process and cause the later to thrash.
    - However, the thrashing processes will be in the paging device queue which will increase the time for a page fault to be serviced and, therefore, the effective access time will increase even for those processes not thrashing.
- To really prevent thrashing, we must provide processes with as many frames as they need.
        - But how do we know how many frames a process "needs"?
        - Look at how many frames a process actually "uses".

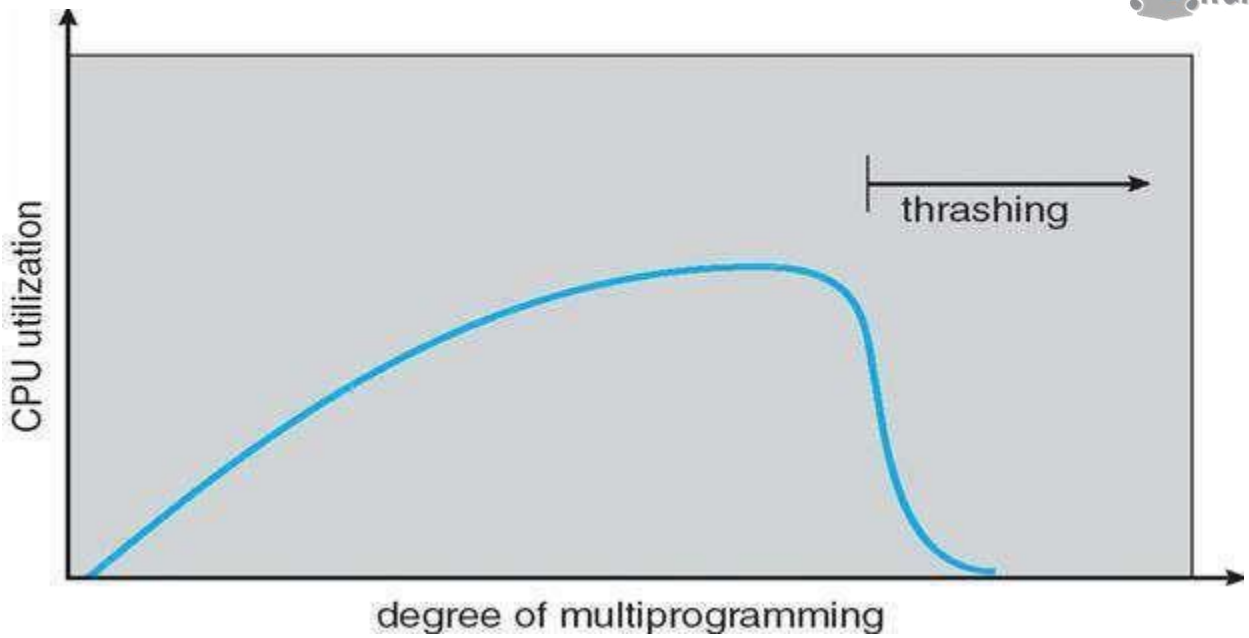


Figure 4.5 Thrashing

Demand Segmentation: Same idea as demand paging applied to segments.

- If a segment is loaded, base and limit are stored in the STE and the valid bit is set in the PTE.
- The PTE is accessed for each memory reference (not really, TLB).
- If the segment is not loaded, the valid bit is unset. The base and limit as well as the disk address of the segment is stored in the OS table.
- A reference to a non-loaded segment generates a segment fault (analogous to page fault).
- To load a segment, we must solve both the placement question and the replacement question (for demand paging, there is no placement question).

The following table compares demand paging with demand segmentation.

Consideration	Demand Paging	Demand Segmentation
Programmer aware	No	Yes
How many addr spaces	1	Many
VA size > PA size	Yes	Yes
Protect individual procedures separately	No	Yes
Accommodate elements with changing sizes	No	Yes
Ease user sharing	No	Yes
Why invented	let the VA size exceed the PA size	Sharing, Protection, independent addr spaces
Internal fragmentation	Yes	No, in principle
External fragmentation	No	Yes
Placement question	No	Yes

Replacement question	Yes	Yes

Role of OS in Security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If an unauthorized user runs a computer program, then he/she may cause severe damage to computer or data stored in it. So a computer system protected against unauthorized access, malicious access to system memory, viruses, worms etc. We are going to discuss following topics.

- Authentication
- One Time passwords
- Program Threats
- System Threats
- Computer Security Classifications

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system, which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username/Password** −User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it used again. One-time password implemented in various ways

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding few alphabets randomly chosen.
- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id, which generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it known as Program Threats. One of the common examples of program threat is a program installed in a computer, which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded

in a program. As user accesses the program, the virus starts embedded in other files/ programs and can make system unusable for user

System Threats

System threats refer to misuse of system services and network connections to put user in trouble. System threats used to launch program threats on a complete network called as program attack. System threats create such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

- **Worm** − Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms' processes can even shut down an entire network.
- **Port Scanning** − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

Computer Security Classifications

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria, there are four security classifications in computer systems: A, B, C, and D. this is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level uses formal design specifications and verification techniques. Grants a high degree of assurance of process security
2	**Type B** Provides mandatory protection system have all the properties of a class C2 system. Attaches a sensitivity label to each object It is of three types. **B1** − Maintains the security label of each object in the system. Label is used for making decisions to access control. **B2** − Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. **B3** − Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.
3	**Type C** Provides protection and user accountability using audit capabilities It is of two types. **C1** − Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. **C2** − Adds an individual-level access control to the capabilities of a Cl level system.
4	**Type D**

	Lowest level Minimum- protection MS-DOS, Window 3.1 fall in this category

Virus in details:

- A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.
- Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures (such as the boot block.)
- Viruses are delivered to systems in a virus dropper, usually some form of a Trojan horse, and usually via e-mail or unsafe downloads.

Figure 4.6 shows typical operation of a boot sector virus:

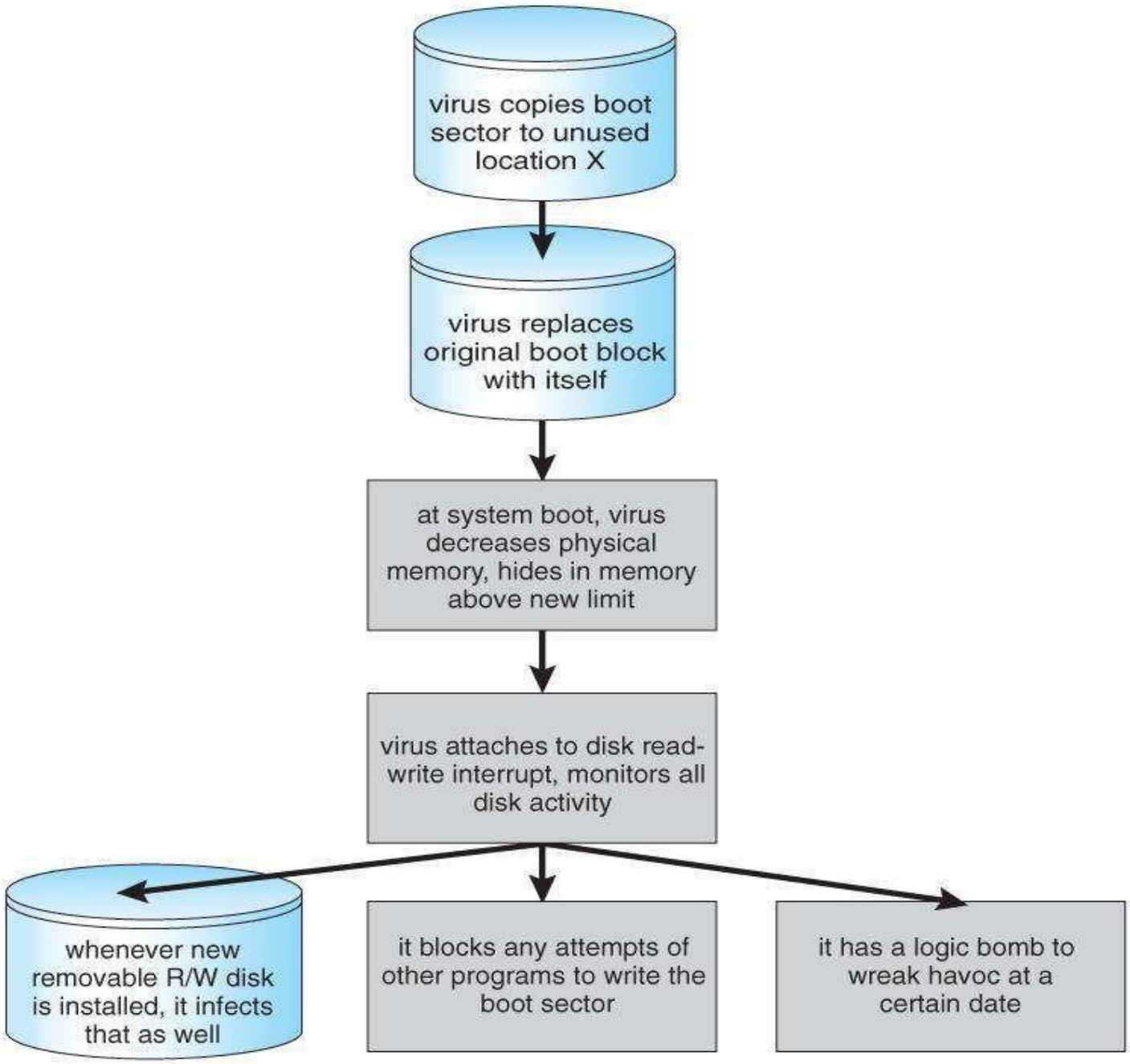


Figure 4.6 a boot-sector computer virus.

Security Techniques

Design Principles Saltzer and Schroeder (1975) identified a core set of principles to operating system security design:

Least privilege: Every object (users and their processes) should work within a minimal set of privileges; access rights should be obtained by explicit request, and the default level of access should be "none".

Economy of mechanisms: Security mechanisms should be as small and simple as possible, aiding in their verification. This implies that they should be integral to an operating system's design, and not an afterthought.

Acceptability: Security mechanisms must at the same time be robust yet non-intrusive. An intrusive mechanism is likely to be counter-productive and avoided by users, if possible.

Complete: Mechanisms must be pervasive and access control checked during all operations — including the tasks of backup and maintenance.

Open design: An operating system's security should not remain secret, nor be provided by stealth. Open mechanisms are subject to scrutiny, review, and continued refinement.

Security breaches: The OS must protect itself from security breaches, such as runaway processes (denial of service), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.

Stack and Buffer Overflow: This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow.

C program with buffer-overflow condition

Consider what happens in the following code, for example, if argv[1] exceeds 256 characters:

- The strcpy command will overflow the buffer, overwriting adjacent areas of memory.
- (The problem could be avoided using strncpy, with a limit of 255 characters copied plus room for the null byte.)

```
#include
#define BUFFER_SIZE 256
 int main( int argc, char * argv[ ] )
{
      char buffer[ BUFFER_SIZE ];
      if ( argc < 2 )
         return -1;
      else {
         strcpy( buffer, argv[ 1 ] );
         return 0;
      }
   }
```

System Protection:

Goals of Protection

- Obviously to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies,* which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

Principles of Protection

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Domain of Protection

- A computer can be viewed as a collection of *processes* and *objects* (both HW & SW).
- The ***need to know principle*** states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure

- A **protection domain** specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An **access right** is the ability to execute an operation on an object.
- A domain is defined as a set of < object, {access right set} > pairs, as shown below. Note that some domains may be disjoint while others overlap.

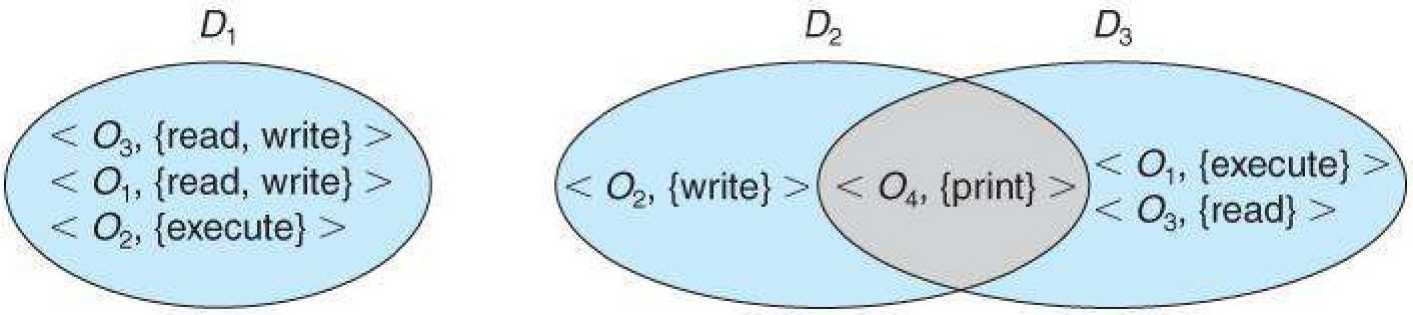


Figure 4.7 systems with three protection domains.

- The association between a process and a domain may be static or dynamic.
- If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.
- If the association is dynamic, then there needs to be a mechanism for domain switching.
- Domains may be realized in different fashions - as users, or as processes, or as procedures. E.g. if each user corresponds to a domain, then that domain defines the access of that user, and changing domains involves changing user ID.

Password Management

There are several forms of software used to help users or organizations better manage passwords:

- Intended for use by a single user:
    - Password manager software is used by individuals to organize and encrypt many personal passwords using a single login. This often involves the use of an encryption keyas well. Password managers are also referred to as **password wallets**.
- Intended for use by a multiple users/groups of users:

- Password synchronization software is used by organizations to arrange for different passwords, on different systems, to have the same value when they belong to the same person.
- Self-service password reset software enables users who forgot their password or triggered an intruder lockout to authenticate using another mechanism and resolve their own problem, without calling an IT help desk.
- Enterprise Single sign on software monitors applications launched by a user and automatically populates login IDs and passwords.
- Web single sign on software intercepts user access to web applications and either inserts authentication information into the HTTP(S) stream or redirects the user to a separate page, where the user is authenticated and directed back to the original URL.
- Privileged password management (used to secure access to shared, privileged accounts).

Privileged password management

Privileged password management is a type of password management used to secure the passwords for login IDs that have elevated security privileges. This is most often done by periodically changing every such password to a new, random value. Since users and automated software processes need these passwords to function, privileged password management systems must also store these passwords and provide various mechanisms to disclose these passwords in a secure and appropriate manner. Privileged password management is related to privileged identity management.

Examples of privileged passwords

There are three main types of privileged passwords. They are used to authenticate:

- **Local administrator accounts**
  On UNIX and Linux systems, the root user is a privileged login account. On Windows, the equivalent is administrator. In general, most operating systems, databases, applications and network devices include an administrative login, used to install software, configure the system, manage users, apply patches, etc. On some systems, different privileged functions are assigned to different users, which mean that there are more privileged login accounts, but each of them is less powerful.
- **Service accounts**
  On the Windows operating system, service programs execute in the context of either system (very privileged but has no password) or of a user account. When services run as a non-system user, the service control manager must provide a login ID and password to run the service program so service accounts have passwords. On UNIX and Linux systems, init and inetd can launch service programs as non-privileged users without knowing their passwords so services do not normally have passwords.
- **Connections by one application to another**
  Often, one application needs to be able to connect to another, to access a service. A common example of this pattern is when a web application must log into a database to retrieve some information. These inter-application connections normally require a login ID and password and this password.

Department of Information Technology
Class Notes

Unit 5
Disk Management: Physical Disk Structure

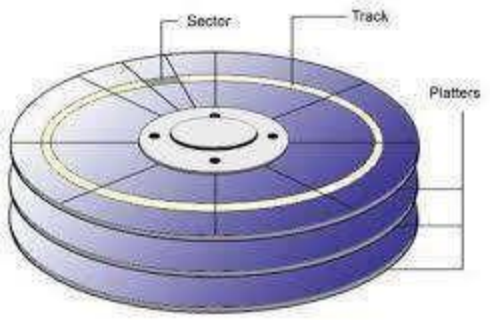


Figure 5.1 Disk Structure

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track (top platter) on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to inner most.
  -

Disk Access Time

Two major components

- Seek time is the time for the disk to move the heads to the cylinder containing the desired sector
  - Typically 5-10 milliseconds
- Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head
  - Typically, 2-4 milliseconds
- One minor component
  - Read/write time or transfer time– actual time to transfer a block, less than a millisecond

Disk Scheduling

- Should ensure a fast access time and disk bandwidth
- Fast access
  - Minimize total seek time of a group of requests
  - If requests are for different cylinders, average rotation latency has to be incurred for each anyway, so minimizing it is not the primary goal (though some scheduling possible if multiple requests for same cylinder is there)
- Seek time =seek distance
- Main goal : reduce total seek distance for a group of requests
- Auxiliary goal: fairness in waiting times for the requests
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- Several algorithms exist to schedule the servicing of disk I/O requests.

TYPES OF DISK SCHEDULING ALGORITHMS

Although there are other algorithms that reduce the seek time of all requests, we will only concentrate on the following disk scheduling algorithms:

1. First Come-First Serve (FCFS)
2. Shortest Seek Time First (SSTF)
3. Elevator (SCAN)
4. Circular SCAN (C-SCAN)
5. LOOK
6. C-LOOK

These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. **First Come -First Serve (FCFS):** All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

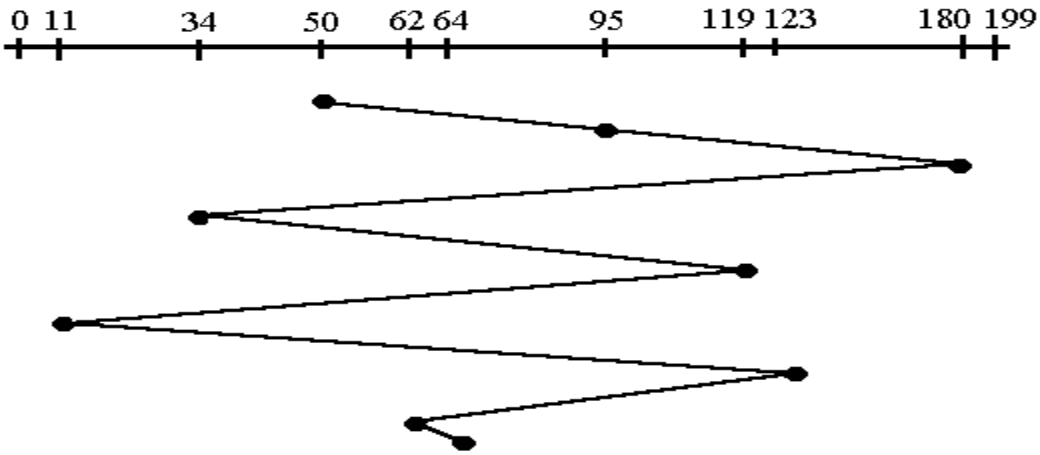


Figure 5.2 FCFS

2. Shortest Seek Time First (SSTF)

In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one.

There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.

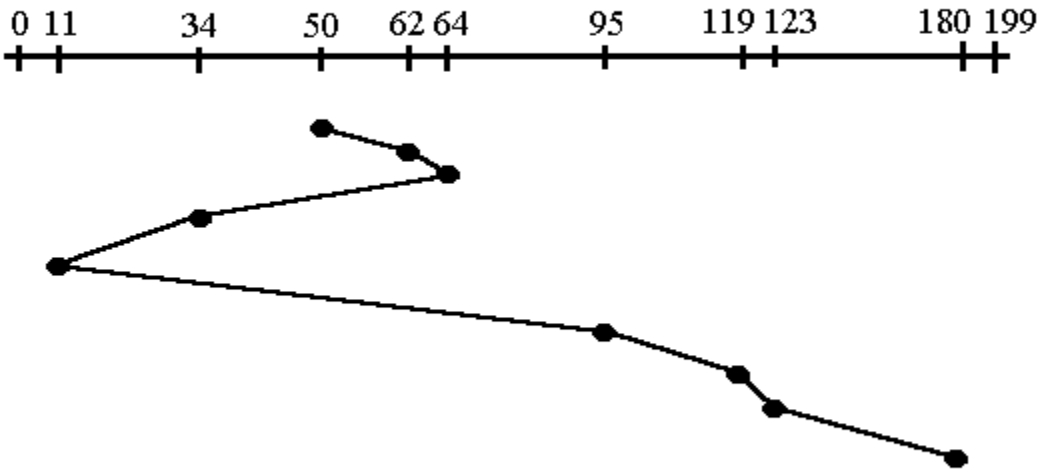


Figure 5.3 SSTF

3.Elevator (SCAN)

This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.

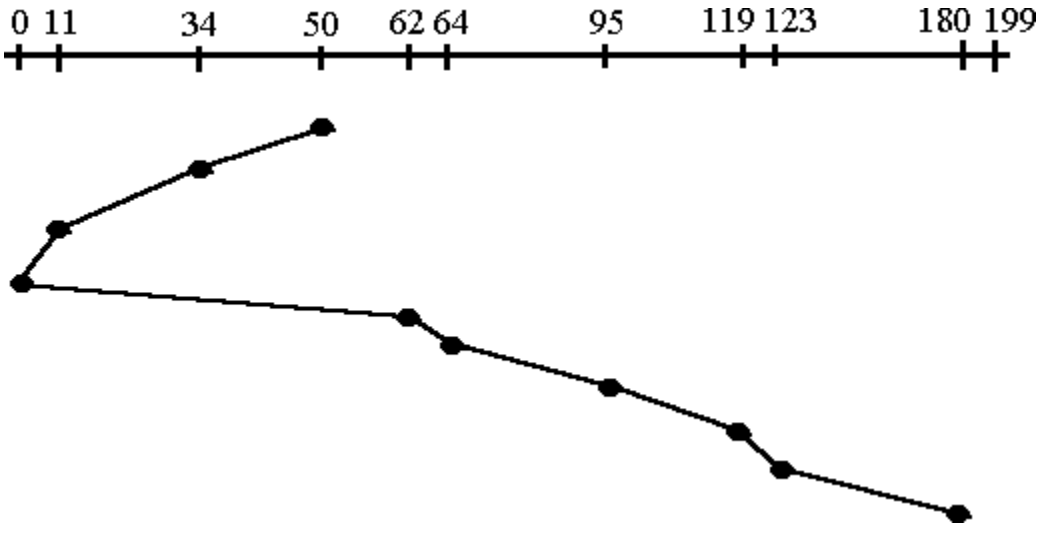


Figure 5.4 Scan

4. Circular Scan (C-SCAN)

Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the most sufficient.

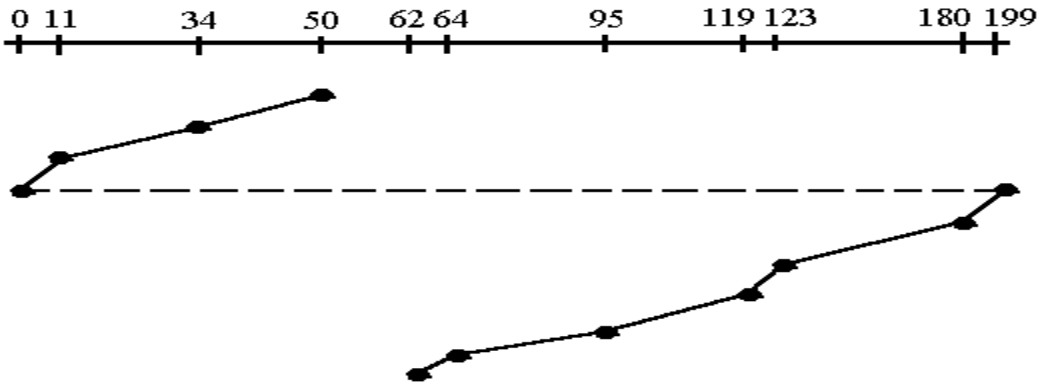


Figure 5.5 C - Scan

5. LOOK: It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

- Work Queue: 23, 89, 132, 42, 187
- there are 200 cylinders numbered from 0 - 199
- the diskhead stars at number 100

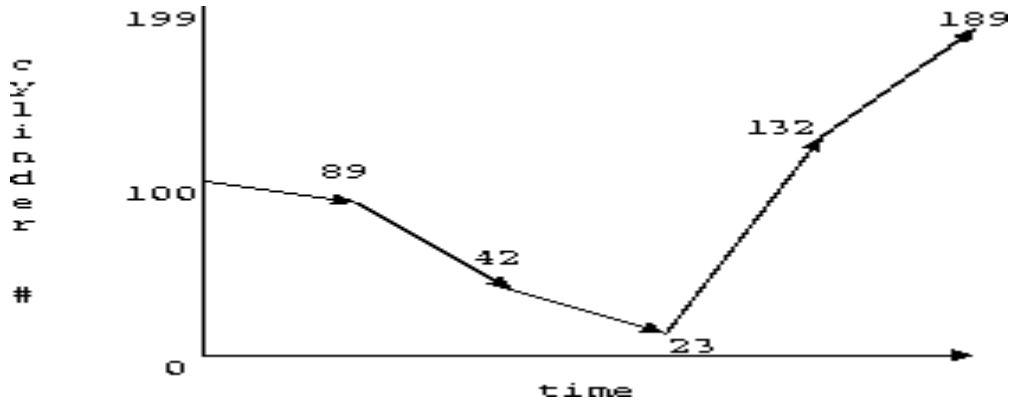


Figure 5.6 LOOK

6. C-LOOK

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.
From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

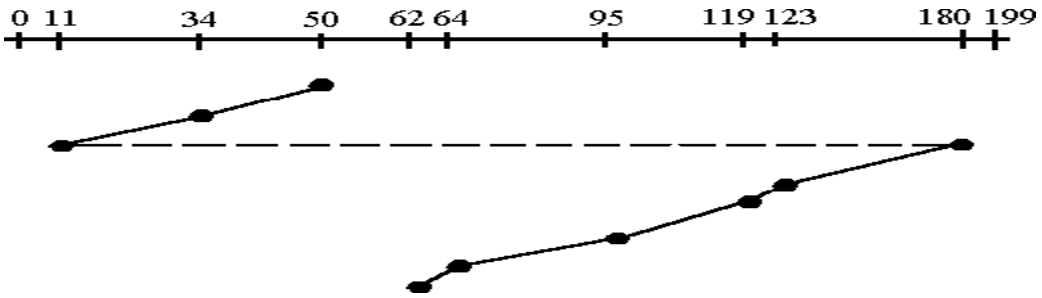


Figure 5.7 C-LOOK

File System

A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

File Structure

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine
- When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure

A file has various kinds of structure. Some of them can be:

- Simple Record Structure with lines of fixed or variable lengths.
- Complex Structures like formatted document or re loadable load files.
- No Definite Structure like sequence of words and bytes etc.

File and Directory concept:

FILE DIRECTORIES:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information especially that is concerned with storage is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Information contained in a device directory is:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed

- Date last updated
- Owner id
- Protection information

Attributes of a File

Following are some of the attributes of a file:

- **Name**: It is the only information which is in human-readable form.
- **Identifier**: The file is identified by a unique tag(number) within file system.
- **Type**: It is needed for systems that support different types of files.
- **Location:** Pointer to file location on device.
- **Size**: The current size of the file.
- **Protection:** This controls and assigns the power of reading, writing, executing.
- **Time, date, and user identification:** This is the data for protection, security, and usage monitoring.

Operation on File

There are many file operations that can be perform by the computer system. Here are the list of some common file operations:

- File Create operation
- File Delete operation
- File Open operation
- File Close operation
- File Read operation
- File Write operation
- File Append operation
- File Seek operation
- File Get attribute operation
- File Set attribute operation
- File Rename operation

Now let's describe briefly about all the above most common operations that can be performed with files.

File Create Operation:

- The file is created with no data.
- The file create operation is the first step of the file.
- Without creating any file, there is no any operation can be performed.

File Delete Operation:

- File must have to be deleted when it is no longer needed just to free up the disk space.
- The file delete operation is the last step of the file.
- After deleting the file, it doesn't exist.

File Open Operation: The process must open the file before using it.

File Close Operation: The file must be closed to free up the internal table space, when all the accesses are finished and the attributes and the disk addresses are no longer needed.

File Read Operation:

The file read operation is performed just to read the data that are stored in the required file.

File Write Operation:

The file write operation is used to write the data to the file, again, generally at the current position.

File Append Operation:

The file append operation is same as the file write operation except that the file append operation only add the data at the end of the file.

File Seek Operation:

For random access files, a method is needed just to specify from where to take the data. Therefore, the file seek operation performs this task.

File Get Attribute Operation:
The file get attributes operation are performed by the processes when they need to read the file attributes to do their required work.
File Set Attribute Operation:
The file set attribute operation used to set some of the attributes (user settable attributes) after the file has been created.
File Rename Operation:
The file rename operation is used to change the name of the existing file.

File Type
File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −
Ordinary files
- These are the files that contain user information
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files
- These files contain list of file names and other information related to these files.

Special files
- These files are also known as device files
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.
- These files are of two types −
  Character special files − data is handled character by character as in case of terminals or printers.
  Block special files − data is handled in blocks as in the case of disks and tapes.

File Access Methods
The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods. some sre:
- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential Access
- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

Direct Access
- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read / written;
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

Indexed Sequential Access
- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.

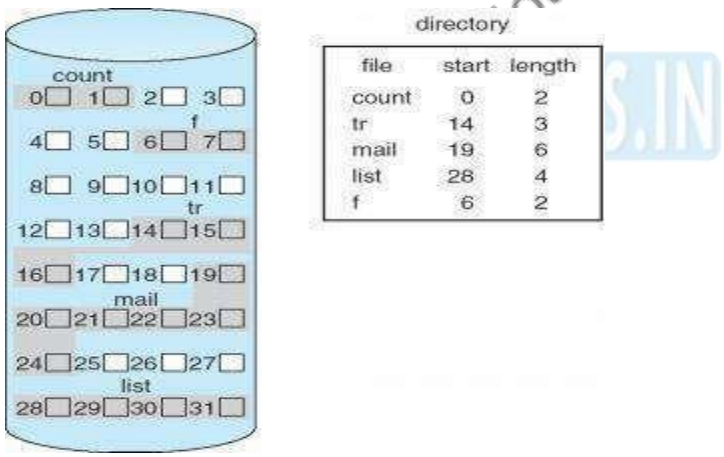


Figure 5.8 contiguous allocation

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation

In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.

The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.

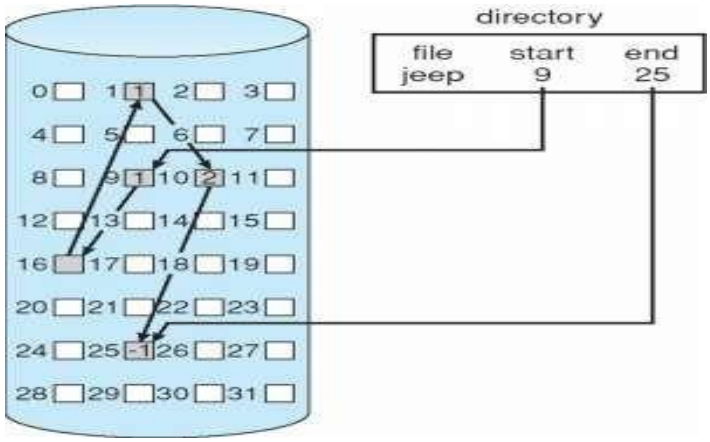


Figure 5.9 Linked List Allocations

Advantages:

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:

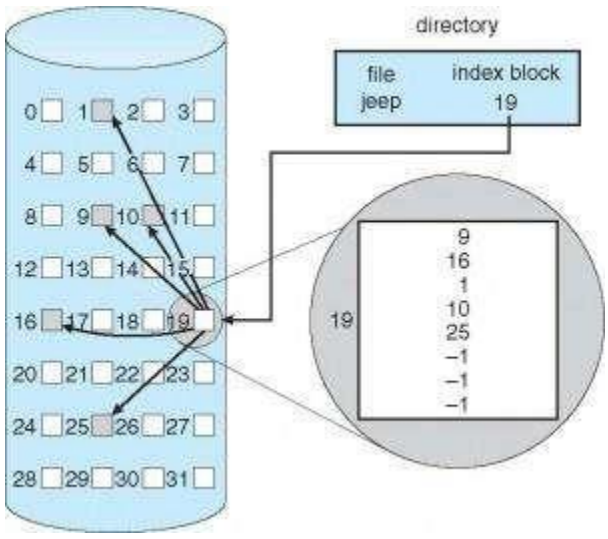


Figure 5.10 Indexed Allocations

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Free Space Management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks – those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.
- This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be 00111100111111000110000001110000 0 …
- The main **advantage** of this approach is it's relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

Example

The Intel family starting with the 80386 and the Motorola family starting with the 68020 (processors that have powered PCs and Macintosh systems, respectively) have instructions that return the offset in a word of the first bit with the value 1. In fact, the Apple Macintosh operating system uses the bit-vector method to allocate disk space.

The calculation of the block number is

(Number of bits per word) x (number of 0-value words) + offset of first 1 bit.

Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs). Keeping it in main memory is possible for smaller disks, such as on microcomputers, but not for larger ones.

A 1.3-GB disk with 512-byte blocks would need a bitmap of over 332 KB to track its free blocks. Clustering the blocks in groups of four reduces this number to over 83 KB per disk.

2. Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.
- The FAT method incorporates free-block accounting data structure. No separate method is needed.

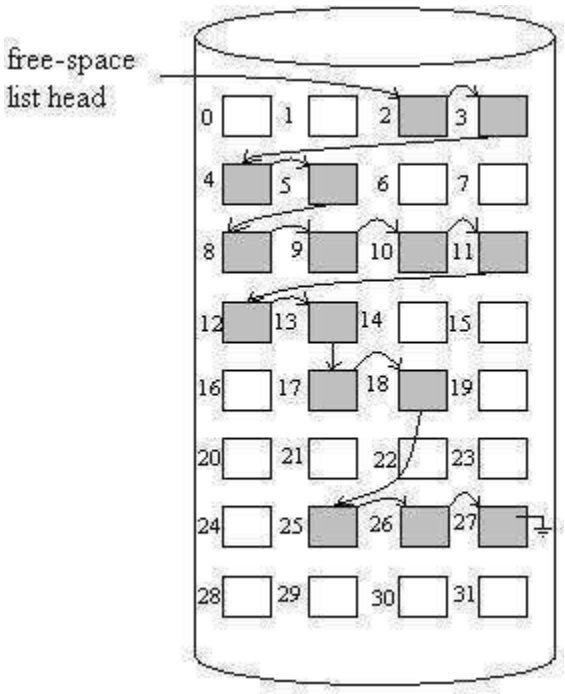


Figure 5.11 Link list space managements

3. Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first n-1 of these blocks is actually free.
- The last block contains the addresses of another n free block, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than one.

Directory

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory:

- **Name**: The name visible to user.
- **Type**: Type of the directory.
- **Location**: Device and location on the device where the file header is located.
- **Size**: Number of bytes/words/blocks in the file.
- **Position**: Current next-read/next-write pointers.
- **Protection**: Access control on read/write/execute/delete.
- **Usage**: Time of creation, access, modification etc.
- **Mounting**: When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

Operations performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

Directory Structure

SINGLE-LEVEL DIRECTORY

In this a single directory is maintained for all the users.

- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.

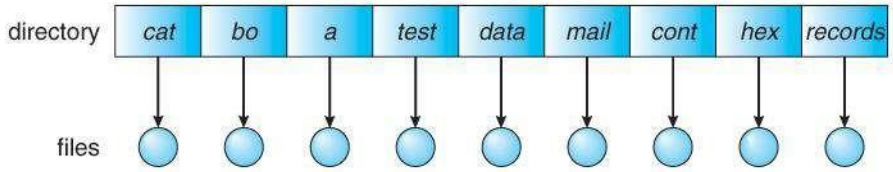


Figure 5.12 SINGLE-LEVEL DIRECTORY

Two-Level Directory

- Each user gets their own directory space
- File names only need to be unique within a given user's directory.
- A master file directory is used to keep track of each user's directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files.
- Systems may or may not allow users to access other directories besides their own
  - If access to other directories is allowed, then provision must be made to specify the directory being accessed.
  - If access is denied, then special consideration must be made for users to run programs located in system directories. A search path is the list of directories in which to search for executable programs, and can be set uniquely for each user.

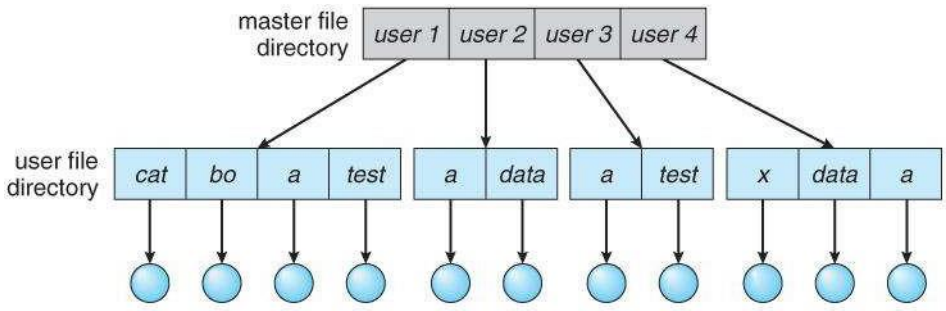


Figure 5.13 Two-Level Directory

Tree-Structured Directories

- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar.
- Each user / process has the concept of a **current directory** from which all (relative) searches take place.
- Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory.)
- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
- One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

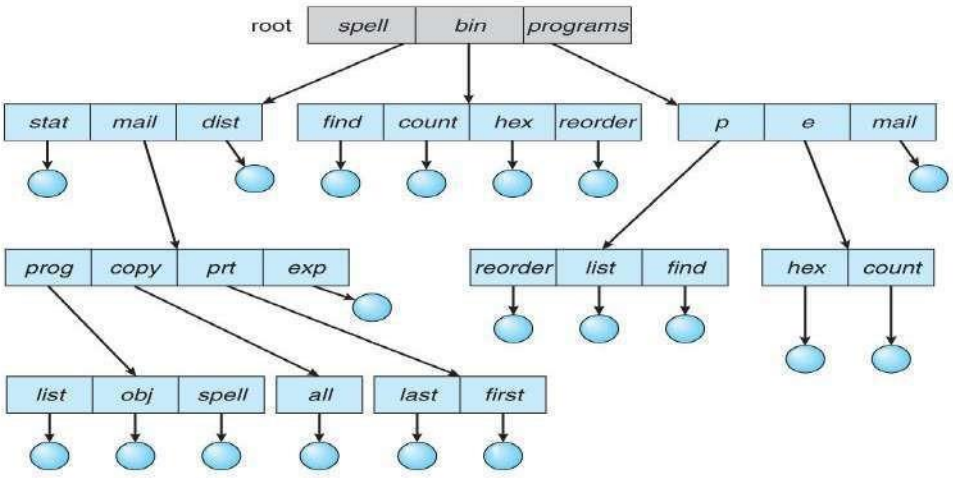


Figure 5.14 Tree-Structured Directories

Protection:

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List
- Access Lists and Groups
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux: RWX
  - owner access 7 ⇒ 1 1 1
  - group access 6 ⇒ 1 1 0
  - public access 1 ⇒ 0 0 1

File Sharing Implement Issue: Sharing of files on multi-user systems is desirable

- Sharing may be done through a protection scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system

- User IDs identify users, allowing permissions and protections to be peruser
- Group IDs allow users to be in groups, permitting group access rights
- Owner of a file / directory
- Group of a file / directory

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using distributed file systems
  - Semi automatically via the world wide web
- Client-server model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - NFS is standard UNIX client-server file sharing protocol
  - CIFS is standard Windows protocol
  - Standard operating system file calls are translated into remote calls

File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called metadata
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

Linux File System

A file is a named collection of related information that is a recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records .Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

Kernel − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low-level hardware details to system or application programs.

System Library − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.

System Utility − System Utility programs are responsible to do specialized, individual level tasks.

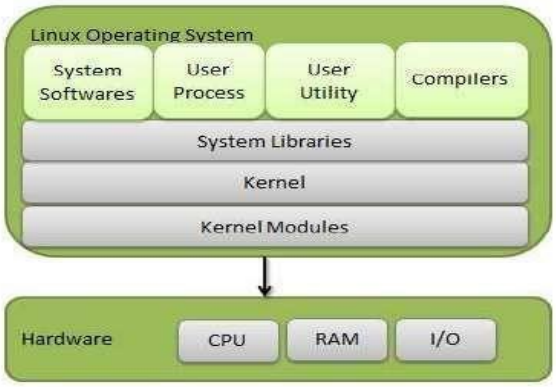


Figure 5.15 Component of Linux

Kernel Mode vs. User Mode

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each process and provides system services to processes, provides protected access to hardware to processes.

Support code, which is not required to run in kernel mode, is in System Library. User programs and other system programs works in User Mode, which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low-level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

Portable − Portability means software can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.

Open Source − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

Multi-User − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

Multiprogramming − Linux is a multiprogramming system means multiple applications can run at same time.

Hierarchical File System − Linux provides a standard file structure in which system files/ user files are arranged.

Shell − Linux provides a special interpreter program which can be used to execute commands of the operating system. It 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.

Architecture

The following illustration shows the architecture of a Linux system −

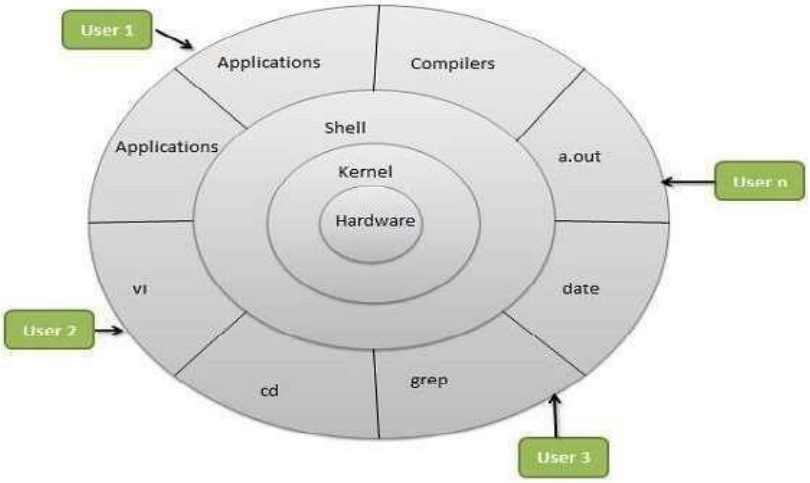


Figure 5.16 Architecture of Linux System

The architecture of a Linux System consists of the following layers −

Hardware layer − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

Kernel − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.

Shell −An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.

Utilities – Utility programs that provide the user most of the functionalities of an operating systems

FAT

A file allocation table (FAT) is a table that an operating system maintains on a hard disk that provides a map of the clusters (the basic units of logical storage on a hard disk) that a file has been stored in. When you write a new file to a hard disk, the file is stored in one or more clusters that are not necessarily next to each other; they may be rather widely scattered over the disk. A typical cluster size is 2,048 bytes, 4,096 bytes, or 8,192 bytes. The operating system creates a FAT entry for the new file that records where each cluster is located and their sequential order. When you read a file, the operating system reassembles the file from clusters and places it as an entire file where you want to read it. For example, if this is a long Web page, it may very well be stored on more than one cluster on your hard disk.

A file allocation table (FAT) is a file system developed for hard drives that originally used 12 or 16 bits for each cluster entry into the file allocation table. It is used by the operating system (OS) to manage files on hard drives and other computer systems. It is often also found on in flash memory, digital cameras and portable devices. It is used to store file information and extend the life of a hard drive.

Most hard drives require a process known as seeking; this is the actual physical searching and positioning of the read/write head of the drive. The FAT file system was designed to reduce the amount of seeking and thus minimize the wear and tear on the hard disc.

FAT was designed to support hard drives and subdirectories. The earlier FAT12 had a cluster addresses to 12-bit values with up to 4078 clusters; it allowed up to 4084 clusters with UNIX. The more efficient FAT16 increased to 16-bit cluster address allowing up to 65,517 clusters per volume, 512-byte clusters with 32MB of space, and had a larger file system; with the four sectors it was 2,048 bytes.

FAT16 was introduced in 1983 by IBM with the simultaneous releases of IBM's personal computer AT (PC AT) and Microsoft's MS-DOS (disk operating system) 3.0 software. In 1987 Compaq DOS 3.31 released an expansion of the original FAT16 and increased the disc sector count to 32 bits. Because the disc was designed for a 16-bit assembly language, the whole disc had to be altered to use 32-bit sector numbers.

In 1997 Microsoft introduced FAT32. This FAT file system increased size limits and allowed DOS real mode code to handle the format. FAT32 has a 32-bit cluster address with 28 bits used to hold the cluster number for up to approximately 268 million clusters. The highest level division of a file system is a partition. The partition is divided into volumes or logical drives. Each logical drive is assigned a letter such as C, D or E.

A FAT file system has four different sections, each as a structure in the FAT partition.

Boot Sector: This is also known as the reserved sector; it is located on the first part of the disc. It contains: the OS's necessary boot loader code to start a PC system, the partition table known as the master boot record (MRB) that describes how the drive is organized, and the BIOS parameter block (BPB) which describes the physical outline of the data storage volume.

FAT Region: This region generally encompasses two copies of the File Allocation Table which is for redundancy checking and specifies how the clusters are assigned.

Data Region: This is where the directory data and existing files are stored. It uses up the majority of the partition.

Root Directory Region: This region is a directory table that contains the information about the directories and files. It is used with FAT16 and FAT12 but not with other FAT file systems. It has a fixed maximum size that is configured when created. FAT32 usually stores the root directory in the data region so it can be expanded if needed.

Introduction to Distributed System:

- A distributed operating system is an operating system that runs on several machines whose purpose is to provide a useful set of services, generally to make the collection of machines behave more like a single machine. The distributed operating system plays the same role in making the collective resources of the machines more usable that a typical single-machine operating system plays in making

that machine's resources more usable. Usually, the machines controlled by a distributed operating system are connected by a relatively high quality network, such as a high speed local area network. Most commonly, the participating nodes of the system are in a relatively small geographical area, something between an office and a campus.

- Distributed operating systems typically run cooperatively on all machines whose resources they control. These machines might be capable of independent operation, or they might be usable merely as resources in the distributed system. In some architectures, each machine is an equally powerful peer as all the others. In other architectures, some machines are permanently designated as master or are given control of particular resources. In yet others, elections or other selection mechanisms are used to designate some machines as having special roles, often controlling roles.
- Sometimes distinctions are made between parallel operating systems, distributed operating systems, and network operating systems, though the latter term is now a bit archaic. The distinctions are perhaps arbitrary, though they do point out differences in the design space for making operating systems control operations across multiple processing engines.
    - A parallel operating system is usually defined as running on specially designed parallel processing hardware. It usually works on the assumption that elements of the hardware (such as the memory) are tightly coupled. Often, the machine is expected to be devoted to running a single task at very high speed.
    - A distributed operating system is usually defined as runing on more loosely coupled hardware. Unlike parallel operating systems, distributed operating systems are intended to make a collection of resources on multiple machines usable by a set of loosely cooperating users running independent tasks.
    - Network operating systems are sometimes regarded as systems that attempt merely to make the network connecting the machines more usable, without regard for some of the larger problems of building effective distributed systems.