Bachelor of Computer Applications (BCA)

Operating System

Yashwantrao Chavan Maharashtra Open University
Dnyangangotri, Near Gangapur Dam, Nashik 422 222

ज्ञानगंगा घरोघरी

Unit No. & Name	Details
Unit 1 History of The Operating Systems	• Introduction(What is OS, Important of OS, Features, Uses, Applications) • Evolution of OS (proprietary, CP/M, DOS, UNIX, Windows and other, Command line to GUI, Portability, ClientServer) • Types of Operating System(multiprogramming systems, batch systems , time sharing systems; operating systems for personal computers & workstations, process control & real timesystems.) • User's View of the Operating System

Unit 2 Operating System –Functions And Structure	• Different Services of the Operating Systems
	  ○ Information Management
	  ○ Process Management
	  ○ Memory Management
	• Uses of System Calls
	• Operating System Structure (Monolithic (Simple) Operating System, Layered Operating System, Microkernel Operating System, Exokernel Operating system),
	• Virtual Machine
	• Booting

Unit 3 Information Management	• DiskBasics • Direct Memory Access(DMA) • File System (Block and Block numbering Scheme, File Support Levels, Writing/Reading a Record, Relationship between the Operating System and DMS, File Directory Entry, Open/Close Operations, Disk Space Allocation Methods, Directory Structure: User's View, Implementation of a Directory System) • Device Driver (DD) (Basics, Path Management, Submodules of DD

Unit4 Process Management	• Process, • Evolution of Multiprogramming • Context Switching, • Process States, Process State Transitions, Process Control Block (PCB), Process Hierarchy, Operation on a Process, Create/ Kill/ Dispatch a Process, Change the Priority of a Process, • Block / Time Up /Wake Up a Process, Suspend/ Resume Operations, • Process Scheduling(Objectives, Concepts of Priority and Time Slice, Scheduling philosophies, Scheduling Levels, Scheduling Policies (For Short Term scheduling)), • Multithreading (Models, Implementation of Threads)
Unit 5 Inter Process Communication	• The Producer-Consumer Problems, Solutions to the Producer-Consumer Problems (Interrupt Disabling/Enabling, Lock-flag, • What are Primitives for Mutual Exclusion? • Classical IPC problems • Semaphores • Alternating Policy • Peterson's Algorithm
Unit 6 I/O Management And Deadlock	• I/O Procedure, I/O Scheduler, Device Handler, Interrupt Service Routine(ISR) • Terminal I/O(Terminal Hardware, Terminal Software) • Organizing Data on theCD-ROM, DVD-ROM • Graphical Representation of a Deadlock, • Deadlock Prerequisites • Deadlock Strategies (Ignore a Deadlock, Detect a Deadlock, Recover from a Deadlock, Prevent a Deadlock, Avoid a Deadlock)

Unit 7 Memory Management	• Single Contiguous Memory Management • Fixed Partitioned Memory Management • Variable Partitions(Allocation Algorithms, Swapping, Relocation and Address Translation, Protection and Sharing, Evaluation) • Non-Contiguous Allocation – General Concepts, Paging (Allocation Algorithms, Swapping, Relocation and Address Translation), Segmentation (Swapping, Address Translation and Relocation, Sharing and Protection) • Combined Systems • Virtual Memory Management Systems (Relocation and Address Translation, Swapping, Relocation and Address Translation, Protection and Sharing, Evaluation, Design Consideration for Virtual Systems)
Unit 8 Protection and Security	• Protection and Security Policy mechanism • Authentication • Internal access Authorization

Yashwantrao
Chavan
Maharashtra
Open University

CMP507
Operating System

Operating System

Yashwantrao Chavan Maharashtra Open University
Dnyangangotri, Near Gangapur Dam
Nashik-422222

Yashwantrao Chavan Maharashtra Open University

Vice-Chancellor: Prof. E. Vayunandan

SCHOOL OF COMPUTER SCIENCE

Dr. Pramod Khandare Director School of Computer Science Y.C.M.Open University Nashik	**Shri. Madhav Palshikar** Associate Professor School of Computer Science Y.C.M.Open University Nashik	**Dr. P.V. Suresh** Director School of Computer and Information Sciences I.G.N.O.U. New Delhi
Dr. Pundlik Ghodke General Manager R&D, Force Motors Ltd. Pune.	**Dr. Sahebrao Bagal** Principal, Sapkal Engineering College Nashik	**Dr. Madhavi Dharankar** Associate Professor Department of Educational Technology S.N.D.T. Women's University, Mumbai
Dr. Urmila Shrawankar Associate Professor, Department of Computer Science and Engineering G.H. Raisoni College of Engineering Hingana Road, Nagpur	**Dr. Hemant Rajguru** Associate Professor, Academic Service Division Y.C.M.Open University Nashik	**Shri. Ram Thakar** Assistant Professor School Of Continuing Education, Y.C.M.Open University Nashik
Mrs. Chetna Kamalskar Assistant Professor School of Science and Technology Y.C.M.Open University, Nashik	**Smt. Shubhangi Desle** Assistant Professor Student Service Division Y.C.M.Open University Nashik	

Writer	Editor	Co-ordinator	Director
1. Mr. Tushar Kute Assistant Professor, Researcher, Computer Science, MITU, Skillologics, Pune 2. Ms. Monali R. Borade Academic Co-ordinator School of Computer Science Y.C.M. Open University, Nashik 3. Ms. Asmita Hendre Assistant Professor, visiting faculty Sinhgad Institute of Management, Pune YCMOU, Pune	Mr.Wipin Wani Assistant Professor, GuruGobind singh College of Engineering & Research Centre, Nashik	Ms. Monali R. Borade Academic Co-ordinator School of Computer Science, Y.C.M. Open University, Nashik	**Dr. Pramod Khandare** **Director** School of Computer Science, Y.C.M. Open University, Nashik

Production

Unit 1 History of the Operating Systems

Objectives:

- To study concepts, features, and applications of operating systems.
- To study Evaluation of Operating System
- To study User view of operating systems.
- To study User View of Operating System

1.1 Introduction

1.1.1What is OS?

An intermediary/interface between the user of a computer and the computer hardware is call as an operating system which is actually a software program. Operating system acts as an environment where the user can conveniently and effectively execute programs. In other words, an OS is a software program that provides a platform/environment in which a user can execute/run programs and controls all the computer resources/hardware.

Operating systems are necessary to run programs on every computer. Applications like Chrome, MS Word, Games, etc. needs some environment in which it will run and perform its task. Through the operating system users, can communicate with the computer without knowing computer language.Users cannot effectively use computer or mobile devices without having an operating system

1.1.2 Features of Operating System

1. Convenience: An operating system makes computer more simple or convenient to use by hiding the hardware details of the computer. It's easy to use as it has a GUI interface.

2. Memory Management: The operating system manages memory. Which actual knows each and every thing about primary memory; that is which part of the memory is used by which program. it allocates memory whenever some program requests for memory.

3. Processor Management and program execution: It is responsible for processor management; It allocates the program to the processor (CPU) and also de-allocates it when a program runs out of the CPU needs. OS executes the program once it gets loaded into memory. The program must be able to end its execution, either normally or abnormally.

4. Device Management/I/O operation: The information about all devices is maintained by operating system. It is also called the I/O controller. The operating system also decides which devices are used to which program, when, and for how long.

5. Communication: operating system provides two ways for communication: shared memory and message passing.When two processes need to communicate or transfer data, the operating system provides communication regardless of whether both the processes are on the same or different CPU.

6. Error Detection/handling: There are various reasons which causes errors to occur in CPU, it mats occur in in I/O devices or in the memory hardware. The operating system constantly needs to monitor the possible errors. It should take the appropriate actions to provide correct and consistent computing operations.

7. Security and Reliability: This feature is responsible to handle and to prevents unauthorized access to any program. It uses passwords and other technologies. It is very reliable because any virus and harmful code can be detected in it with the help of proper security enforcing techniques.

1.1.3 Uses of Operating systems

For the operation of any computer the operating system is essential. Anoperating system performs three main functions:

(1) Various computers' resources, such as the central processing unit, memory, disk drives, and printers are managed by Operating system.

(2) Establishing a user interface

(3) For applications software executing and providing services. Today in each and every sector uses a computer which required operating system, such as banks, schools, hospitals, companies, mobiles, etc. No device can operate without an operating system because it controls all the user's commands. For example-

- **LINUX/UNIX** operating system is often used in banks because of the security features provided by LINUX.
- Mobile phones use lightweight operating systems such as **Symbian OS, Windows Mobile, iOS, and Android OS**.

1.1.4 Applications of Operating System

1. Embedded systems (Often used in home appliances)

2. Automobile engine controllers' system

3. Industrial robots, Medical Robots and Research

4. Spacecraft Control System

5. Industrial Control

6. Large-scale computing systems such as green computing, mobile computing, soft computing etc.

7. Diagnostic mode of a computer OS.

8. Real time systems are used in: Airline reservation systems, Air traffic control systems, Systems that provide immediate updating, Defense application systems like RADAR, Networked Multimedia Systems, and Command Control Systems.

1.2 Evaluation of Operating System

Operating systems have been evolving through the years. Following table shows a summarized history of operating system.

Generation	Year	Electronic devices used	Types of OS and devices
First	1945-55	Vacuum tubes	Plug board
Second	1955-65	Transistors	Batch systems
Third	1965-1980	Integrated circuits(IC)	Multiprogramming
Fourth	Since 1980	Large scale integration	PC

1.2.1 CP/M

CP/M stands for **"Central Program for microcomputers**" was almost the first OS developed for microcomputer platform based on Intel 8080 machine in 1974 by Gary Kildall. Initially it was developed only as a file system to support all the file related operations for PL/M (High level Programming language for 8080 machine) compiler. Later utilities like editors, debuggers etc. were added to it. CP/M became popular because of its user friendliness, time sharing and real-time capabilities however it had slower disk operations and couldn't support networking capabilities. Hence later DOS (Disk Operating System) came in picture.

1.2.2 Proprietary Operating System

Proprietary term describes something owned by a specific individual or company. Usually, proprietary software like operating systemsare "closed-sourced" i.e. It's not open source, freely available or freely licensed. It is designed, customized, developed and sold by only one company.

For example-MAC OS X operating system for apple machines developed by Apple Company, which is strictly allowed to run on apple hardware.

OS has various proprietary features such as simplified user experience; complex graphics, limited customizability, interoperability, development and resource support from manufacturer etc. These types of OS are usually costly compared to open-source OS.

1.2.3 DOS/MS-DOS

A company called "Seattle Computer" developed an OS called QDOS for Intel 8086 m/c. Later Microsoft Corporation acquired rights to it and developed MS-DOS (Microsoft Disk Operating system).

The main features of MS-DOS are:

It is a character user interface (CUI) based OS.

It is a command line operating system.

It supports single-user, single-tasking.

It is a 16-bit OS.

Ms-DOS has various features like user-friendliness, faster disk operations, compiler support for various high-level languages like BASIC, COBOL and C language and networking capabilities like LAN makes MS-DOS popular. It became immensely popular for software development. Later it is modified by adding GUI (Graphical User Interface) and new version of MS-DOS which is Windows (MS Windows) was introduced.

1.2.4 UNIX, LINUX

UNIX was initially developed as a single user operating system. It was developed much before Microsoft developed DOS. The design of DOS is based on the features of UNI X operating system. It is popular because of features like multiple user support simultaneously. It is a CUI based OS that supports multiprogramming, time-sharing and multitasking.

It's written in high level language; the code developed in UNIX can be easily modified and compiled on any system even though the hardware is different hence portability and machine independence are the important features of UNIX.

Later **LINUX** an open-sourced OS was developed based on features of UNIX. It doesn't needa licence and it follows open development model which allows any programmers to access the source code and modifythe operating system. Apart from supporting UNIX features it provides high security and cross-platform capabilities. Hence, it's extremely powerful, popular and inexpensive operating system.

1.2.5 Microsoft Windows and others

Microsoft's windows family consists of various graphical user interface (GUI) based OS like Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP and latest versions. The most popular windows OS currently in use are windows XP and its later versions like Windows 8, windows 10 which supports greater networking capabilities, user friendliness, more stability and security, 32-bit/64-bit operations, multitasking and multiprogramming, plug and play features ,compiler support for several advance languages and software development, faster performance.

Some higher versions of operating systems can be customized for parallel processing and advanced graphics capabilities, portability and interoperability and distributed environment.

1.2.6 Command Line to GUI

A **command line interface (CLI)** is a text-based user interface (UI) used to view and manage computer files. Users interacted with an OS or applications using a keyboard before the invention of the mouse. Users ran the command line interface to run tasks or programs through commands.

The command line interface typically looks like a black box with white text. The user responds to a prompt/text in the command line interface by typing a command. The

output/response of a command can be tables, messages, lists, or some other information/confirmation. The software that operates the command line interface is called shell, also known as a command language interpreter. The CLI Commands were difficult to remember and it lacked user friendliness.

The **graphical user interface (GUI)** is the most popular user interface today. In GUIs, commands are executed by using windows, icons, and menus. The advantage of a GUI is, it provides pictorial/visual view of the available functions and it's simple & easy to use. Many GUIs permit navigation and execution using a keyboard, however, the most common way to navigate through a GUI is using a mouse. Microsoft Word is one such example of a GUI-based application that is commonly used. Page layouts and styles can be changed by selecting corresponding icons with the help of a mouse or keyboard. The GUI is more user-friendly than CLI.

1.2.7 Client-Server

1. Client OS:

It is an operating system that operates within a desktop. It is used to obtain services from a server. It run on the client devices like laptop, computer and is very simple operating system.

2. Server OS:

It is an operating system that is designed to be used on server. It is used to provide services to multiple clients. It can serve multiple clients at a time and is very advanced operating system.

Client/Server communication involves clients sending requests to the server and the server responding to the client's requests. A single server generally communicates with multiple clients simultaneously.

1.3 Types of OS

An operating system may process its workload serially or concurrently. Usually, the computer systems are either dedicated to a single program until the program finishes its execution or dynamically reassigned to a group of active programs which are in different stages of execution. Several variations of both serial and multiprogrammed operating systems exist.

1.3.1 Batch Operating System

- A few computer systems only performed one task at a time. They had a list of instructions to carry out and these would be carried out one after the other, this is called a serial system. The mechanics of development and preparation of programs in such environments are quite slow and numerous manual operations involved in the process.
- In the 1970s, the Batch processing were very common.The Jobs were carried out in batches.A predefined sequence of commands, programs, and data combined in a single unit is referred to as a Job. A single computer called the mainframe was used for batch processing. More than one user is allowed to submit their respective jobs for execution in Batch OS.
- In Batch OS, before the program execution starts program and data are collected in a batch. All the jobs sent to the system get put in a queue on a first come first serve basis and then the jobs are executed one at a time. When all the jobs get executed, the users collect their respective output.
- The memory layout for a simple batch system, memory is usually divided into two areas: Operating system and user program area.
- Batch system uses first come first serve scheduling method and are processed in the order of submission.
- When a system starts a job execution, memory is allotted to a job, once the system completes execution, the allotted memory is released and the output is stored into an output spool for printing later. Spooling is similar to buffer queue where all the printing operations/job are put in order.

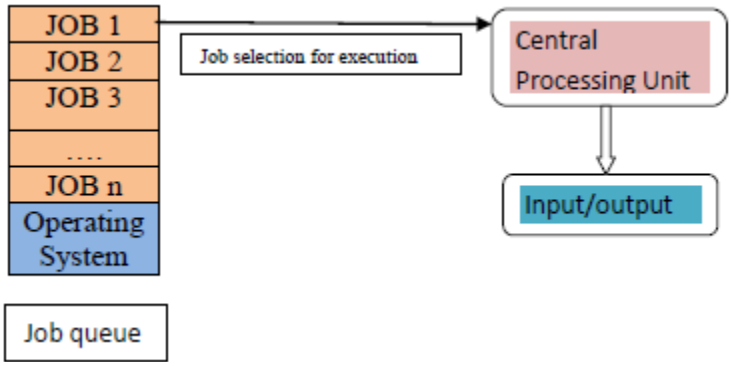


Figure 1.1 Batch OS Job execution

- **Advantages of batch system**

1. Less work load on the user. (More work to the computer)
2. Another job starts executing as soon as the previous job is finished which in turn increases performance.

- **Disadvantages of Batch OS**

1. Starvation

If there are five jobs JOB1, JOB2, JOB3, JOB4, and JOB5 present in the batch. If the execution time of JOB 1 is very high then the other four jobs will have to wait an extreme period of time for execution and resources. This can cause starvation. Starvation is where a job has to wait for resources and execution because other jobs occupy those resources.

2. Not Interactive

Batch Processing is not effective for the jobs that are dependent on the user's input. If a job requires extra input in the batch processing scenario, the user is not able to provide inputs during job execution or modify anything. This is why Batch OS is not interactive.

3. Total execution time can be large.
4. Program debugging is difficult.
5. If a job doesn't get the chance to execute it can enter into infinite loop of waiting.
6. If a job execution corrupts the system all other pending jobs get affected.
7. Lack of protection scheme

1.3.2 Multiprogramming Operating System

- Multiprogramming is an extension to the batch processing where the CPU is always busy and is never idle unless there are no jobs in system. Each process time is divided into two types of system time: CPU time and IO time. In multiprogramming environment, when one process is performing its Input/output, at the same time, the CPU starts executing other processes hence it increases system efficiency.
- When two or more programs are in memory at the same time, sharing the processor is called as multiprogramming OS. Multiprogramming assumes a single processor that is being shared. CPU utilization is increased by organizing jobs in a way that the CPU

always has at least one job to execute. Fig 1.2 shows the memory arrangement for a multiprogramming OS.

- The OS has multiple jobs in memory at the same time. This set of jobs is a subset of the jobs kept in the job pool. A job which enters the system is stored in the memory. Operating system decides which job to pick and starts its execution. Memory management is necessary when multiple programs are running in memory at the same time.

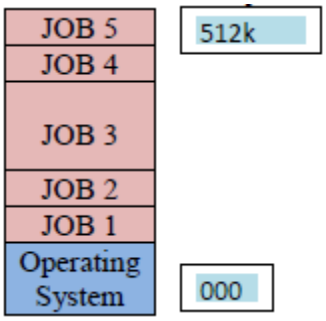


Figure 1.2 Multiprogramming OS job execution

- Multiprogrammed systems is an environment where different resources are utilized effectively by allocating them to different jobs which eliminates the risk of starvation unlike batch OS, however it doesn't provide user interactions with CPU/Computer.
- All active programs and system resources are monitored by Multiprogramming OS on every stage of execution hence the CPU is never in idle state unless there are no jobs.

Advantages:

1. CPU utilization is higher as multiple programs are running.
2. Simultaneous multiple program allocation to CPU is possible which fastens execution time.

Disadvantages:

1. CPU scheduling is necessary.
2. Memory management is needed to accommodate multiple jobs in memory.

1.3.3 Time sharing systems

- Time sharing system supports interactive users. It is also referred to as a multitasking system. It's a logical extension of multiprogramming. Time sharing system uses CPU scheduling and multiprogramming to provide an economical and interactive system for two or more users.
- A time-shared OS implements CPU scheduling and multi-programming to provide every user at least a small portion of a shared computer at once. Every user can have at least one separate program in memory.
- When one program is completing its input/output operations /loading into the memory, OS starts executing other program simultaneously which has been already loaded into memory. Once the execution of current program is completed the next loaded program execution starts. This short period of time during which user gets attention of CPU is known as time slice, time slot or quantum time. It is typically within the range of 10 to 100 milliseconds.
- Time shared OS are complex than multiprogramming OS. Time sharing system provides isolation and protection of co-resident programs through Memory management. In both the systems memory management and security are necessary as, multiple jobs must be kept in memory simultaneously.
- In order to achieve a good response time, jobs may have to swap in and out of disk from main memory where disk serves as a backing store for main memory. A virtual memory is commonly used technique to achieve this objective. Virtual memory is a technique which allows the execution of a job that may not be completely in memory.
- Time sharing system can run several programs simultaneously; hence it also works as a multiprogramming system. But multiprogramming OS is not a time-sharing system. Difference between both the systems is that time sharing system allows more frequent context switches. This gives every user an impression of entire computer being dedicated to their use. In multiprogramming system, a context switch only occurs when the current executing process stalls for some reason.

Advantages:

1. Each task gets an equal opportunity.
2. Software duplication chances are reduced.
3. Reduced CPU idle time.

Disadvantages:

1. Reliability problem.
2. Security and integrity of user programs and data needs to be implemented.
3. Can risk data communication problem.

1.3.4 Process Control and Real Time Operating System

- RTOS is an operating system is designed to serve real time applications or systems which are output critical. It processes data the moment it's inputted, mostly without buffering or delay. Examples of this kind of Real time systems are OS which are used to control autonomous systems like satellites, robots and hydroelectric dams. A real time OS must react to inputs and responds to them quickly. It cannot afford to be late with a response to an event or input or else the output will be useless or failure.
- Real time systems can be of two types: Hard real time system and soft real time system.

A hard real time system: Hard RTOS, the deadlines are extremely important and handled strictly which means a given task must start and complete its execution in specified scheduled time or assigned time duration. Example: Medical critical care system, Aircraft systems, etc.

Soft real time system: is a less restrictive type. Soft Real time RTOS, accepts some delays in processing. In this type of RTOS, there is a specific deadline set for every job. As a delay of a small amount of time is acceptable, deadlines are handled softly by this type of RTOS.
Example: Online Transaction system and Livestock price quotation Systems.

- General real time applications with some examples are listed below

Applications: Example

Detection: Radar system, Burglar alarm
Process monitoring and control: Petroleum, Paper mill
Communication: Telephone switching system
Flight simulation and control: Auto pilot shuttle mission simulator
Transportation: Traffic light system, Air traffic control
Military applications: for example, missiles
Weather forecasting

1.3.5 OS for Personal Computer and Workstations

- During the late 1970, computers had faster CPU, thus creating an even greater disparity between their rapid processing speed and slower I/O access time.
- Multiprogramming schemes to increase CPU use were limited by the physical capacity of the main memory, which was a limited resource and very expensive. This system includes PC running MS window and the Apple Macintosh. The Apple Macintosh OS support new advance hardware i.e., virtual memory and multitasking with virtual memory, the entire program did not need to reside in memory before execution could begin.
- Linux, a UNIX like OS available for PC, has also become popular recently. The microcomputer was developed for single users in the late 1970. Physical size was smaller than the minicomputers of that time, though larger than the microcomputers of today.
- Microcomputer grew to accommodate software with large capacity and greater speeds. The distinguishing characteristics of a microcomputer are its single user status. MS-DOS is an example of a microcomputer operating system.
- The most powerful microcomputers are used by commercial, educational, government enterprises.
- Cost of Hardware for microcomputers is sufficiently low as a single user (individuals) can use a computer solely and almost every system is integrated with networking capabilities.
- **A workstation** is a computer which is faster and more capable than a personal computer and is dedicated to an individual user. It's more commonly used for business or professional purpose (rather than home or recreational use).
- Workstations and applications designed for workstations are commonly used by small engineering companies, architects, graphic designers, and any organization, department, or individual that requires a faster microprocessor, a large amount of random-access memory (RAM), and special features such as high-speed graphics adapters. UNIX OS is commonly used as the workstation operating system. Workstation OS can also work as a client operating system.

1.4 User View of Operating System

The Operating System is an interface which hides the background processing details which are being performed and presents OS a virtual machine to the user which makes it easier to use. An **operating system** allows the user application programs to interact with the **system** hardware. **Operating system** by itself does not provide any functions but it provides an atmosphere in which different applications and programs can perform tasks.

Operating System provides the following services to the user.

- Execution of a program
- Access to I/O devices
- Controlled access to files
- Error detection (Hardware failures, and software errors)

The user's view of OS depends on the system interface that is used by the users. The different types of user view experiences include:

- If the user is Woking with a personal computer, the OS is specifically designed to facilitate easy interactions and high performance. However, it's not necessary for OS to worry about resource utilization as usually individual personal computers are using all the resources available and there is no sharing or limited sharing.
- In case user is working on a system connected to a mainframe or a minicomputer, the OS mainly focuses on resource utilization as mainframe can have multiple terminals connected to it and OS needs to ensure the uniform distribution or division among resources like CPU, memory, I/O devices etc.
- If the user is using workstations connected through networks, then OS has to focus on individual usage of resources and sharing within the network. This is because the workstations are exclusively using their own resources and also needs to share files with other workstations within the network.
- The operating system handles the usability of the devices including a few remote operations in case of handheld computers or PDA devices. It also focuses on the battery level of the devices.

There are some devices that contain very less or no user views because there is no interaction with the users. Examples are embedded computers in home devices, automobiles etc.

1.5 Exercise Questions

1. What is Operating System?
2. What are features of Operating System?
3. Explain the various uses of Operating System?
4. List the various applications of Operating System?
5. Write a short note on Evolution of Operating System?
6. Explain in short Multiprogramming Operating System.
7. Write a short note on types of operating system.
8. Write a short note on Process Control and Real Time Operating System.
9. Explain in short User View of Operating System.

Unit 2 Operating System –Functions and Structure

Objectives:

- To study different services of Operating system.
- To study Uses of System calls.
- To study Structure of Operating system.
- To study concept of Virtual machine.
- To study concept of booting.

2.1 Different Services of the Operating Systems

Programs and users are supplied different kinds of services by the Operating System. Application programs (programs that run inside an OS) are also provided an environment to execute programs freely. Operating systems provide users a convenient way to run various programs. The services of one operating system can be different than other operating systems. Operating system makes the programming task easier. The Operating systems provides variety of services which are explained below.

2.1.1 Process Management

- **'Process' talks about a program in execution**. Process management is the method the operating system uses to manage processes that run simultaneously. Every When software application program is running each software has one or more processes associated with them. Let us understand this with an example, when we use Google Chrome browser, there is a process running for that browser program. Similarly, there are many processes running for Operating system, which performing various functions.
- A process needs many resources, such as CPU time, memory, files and I/O devices. The resources are either allocated to the process when it is running or assigned to the process when they are created.
- When the process terminates, the operating system reclaims any resources that can be reused. Processes like these should be managed by process management; the execution of a process must be in sequence so, on behalf of the process, at least one instruction should be executed.
- The term process refers to an executing set of machine instructions. Program by itself is not a process. A program is a passive entity.

- The operating system is in charge of the following activities of the process management:

  1. Is responsible to create and to destroy the user and system processes.
  2. Allocating hardware resources among the processes.
  3. Controlling the progress of processes.
  4. Providing mechanisms for process communications.
  5. For handling deadlock it provides mechanisms.

2.1.2 Main Memory Management

- Main Memory is a large arrangement of storage (bytes), which has an address. A sequence of reads or writes on specific memory addresses cane be used to conduct the memory management process. To execute a program, first the program should be mapped to an absolute address and then loaded inside the memory. To execute a program, it must be living in the main Memory.
- The memory management modules of an operating system are concerned with the management of the primary memory (main memory). Memory management include the following functions:

  1. Tracking and recording the status of each location of main memory. In simple words one can say each memory location should be either free or allocated.
  2. Determining allocation policy for memory.
  3. Allocating memory when a process requests. Allocation techniques are used to choose specific location. It is also in control of allocation information updating.
  4. It also de-allocates Memory when it is no longer required by the process or has been terminated. Information must be updated after de-allocation status.
- Memory management is primarily focused on allocation of physical memory to requesting processes. By performance of the memory management module the overall resource utilization and other performance criteria of a computer system are affected. Many memory management schemes are available and the effectiveness of the algorithms is determined by the situation & OS.

- **Secondary Storage Management**: A Computer used mechanism to store information in such a way that this information may be retrieved at a later time, this mechanism is called as storage device, which are used for storing all the data and programs. These programs and data access by computer system must be stored in main memory Size of main memory

is small to store all data and programs. Power failure can cause data loss. For this reason, secondary storage devices are used. Therefore, effective management of disk storage is very important to a computer system.

2.1.3 File Management

- Secondary storage is usually organizing logically related data into named collections called files. In simple and short word one can say, logical collection of information is a file. A computer uses physical media for storing the different information.
- A file may consist of a report, an executable program or a set of commands to the operating system. A file consists of a sequence of bits, bytes, lines or records whose meanings are defined by their creators. Physical media such as secondary storage device is used for storing the files.
- Mediums are controlled by devices. Different types of physical media can have their own characteristics and physical organization. The some of the examples are, magnetic tapes, magnetic disks and optical disks.
- With respect to file management the operating system is responsible for the following:
  1. Creating and deleting of files.
  2. Files are mapped to secondary storage.
  3. Creation and deletion of directories
  4. Using stable storage media to create backup files.
  5. Supporting primitives for manipulating files and directories.
  6. Transmission of file elements between main and secondary storage.

  As one or more layers of the operating system the file management subsystem can be implemented

2.2 Uses of System Calls

The interface between a process and the operating system is a mechanism provided by system call. A computer program can request a service from OS kernel using system calls. If a file system requires the creation or deletion of files. System calls performs read and write operations on files. System call provides different services to users using API (Application Programming Interface). The only entry point for kernel is system calls.

Modern processors provide instructions that can be used as system calls; A system call instruction is an instruction that generates an interrupt that causes the operating system to gain control of the processor.

- **Services Provided by System Calls:**

1. Creation and management of processes
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

- **Types of System Calls:**

1. Process control/creation and management:

a. Create process

b. Terminate process: Terminate the process making the system call

c. Wait: wait for another process to exit

d. Fork: Create a duplicate of the process making a system

e. End: Halt the process execution normally

f .Abort: Halt the process execution abnormally

g. Load: Load the process into memory: : Execute the loaded process

h. Execute: Execute the loaded process.

i. Get process attributes and set process attributes

j. Allocate and free memory

2. File Access, Directory and File system management : create file, open file to read or write, close file, delete a file, stat: get information about a file, unlink: remove file from a directory, get attribute and set file attribute: attributes including file names, file type, protection codes and accounting information etc.

3. I/O Device management:

a. Request device: to ensure exclusive use of device

b. Release device: Release the device after execution is finished

c. Read/Write: Same as file system call

d. Stat: Get information about I/O devices.

4. Information maintenance:

a. Get time and date

b. Set time and date

c. Get process, file or device attributes

d. Set process, file or device attributes

e. Get system data

f. set system data

5. Inter-process communication:

a. Create message queue: Create a queue to hold message Send a message to a message queue

b. Send message: send message to message queue

c. Receive message: Receive a message from a message queue

d. Close connection: Terminates the communication.

2.3 Operating System Structure

Traditionally, operating system architecture is based on the separation of concerns principle in which the OS is divided into independent modules and every module provides some kind of features. This modular designing makes operating system manageable.

The operating system possesses privileges which allow access to protected resources like physical devices and application memory. The privileges are given access to individual modules instead of OS as a whole when they require it which in-turn reduces both accidental and malicious misuse.

- **A kernel: A kernel** is the core component of an operating system. A kernel manages system resources and also acts as a bridge between the software and hardware of the computer.
- It is one of the first programs to load after boot-loader on start-up the computer. The Kernel also offers secure access of the machine's hardware to different programs. When to provide access and how long is also decided by kernel.
- It monitors and facilitates the communication between applications and the data processing being performed at the hardware level by using inter-process communication and system calls. The kernel performs low-level tasks of managing disks, task management and memory management.
- Following figure depicts the **kernel of operating system.**

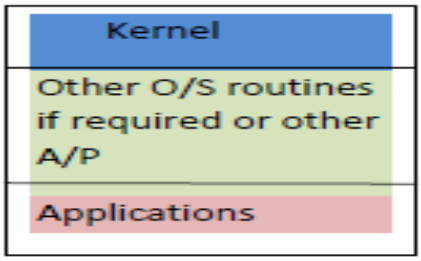


Figure 2.1 Kernel

Different types of kernels are: monolithic kernel, Microkernel, Hybrid kernel, Exo-Kernel, and Nano-Kernel. Monolithic kernel and Microkernel are most common used kernel systems. Kernel space is where the kernel executes different operations and provides its services. User space is a set of memory locations where user processes (i.e., everything other than the kernel) executes. A process is an executing instance of a program.

2.3.1 Monolithic (simple) Operating System

Monolithic Kernel manages system resources between application and hardware, but user services and kernel services are implemented under same address space. It increases the size of OS by increasing size of the kernel. This kernel provides CPU scheduling, memory management, file management and other operating system functions using system calls. As both kernel and user services are implemented under same address space, which makes OS execution faster however if one service fails, the entire system can crash or fail. This is one of the drawback monolithic kernel OS has. When User wants add or modify any new service the entire OS needs modification.

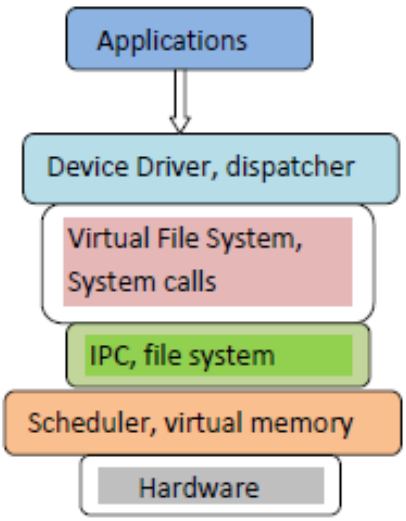


Figure 2.2 Monolithic Kernel

Advantages of Monolithic Kernel

- Important advantage of monolithic kernel is that it provides CPU scheduling, memory management, file management and other operating system functions using system calls.
- A single large process can run entirely in a single address space which is another advantage.
- Design and implementation are simple.
- Expansion using module systems is possible.

Disadvantages of Monolithic Kernel

- One of the major disadvantages of monolithic kernel is one service failure leads to entire system failure.
- User requires to modify entire OS in case of modification or addition of new services to OS
- Module system may not provide runtime loading and unloading.
- It becomes harder to maintain as code base size increases.
- Lower fault tolerance, if core module or section of the kernel fails, the whole thing fails.

2.3.2 Layered Operating System

- A layered design of the OS architecture structures the architecture of OS into different layers with different privileges which helps to achieve robustness. The most privileged layer contains code for interrupt handling and context switching, the layers above contain device drivers, memory management, file systems, user interfaces, and the least privileged layer contains the applications.
- The bottom layer is the hardware layer. Bottom layer number is 0. The topmost layer (layer N) is the user interface. An operating system could be structured to contain many layers. Each layer uses the interface provided by the layer below it and provides a more intelligent interface to the layer above it. Following figure shows layered Kernel.

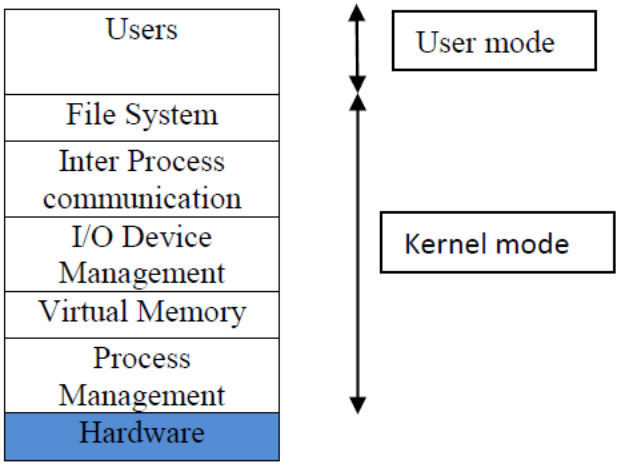


Figure 2.3 Layered OS

- Layered structure provides good modularity. Each layer of the operating system forms a module with a clearly defined functionality and interface with the rest of the operating system. This method helps simplifying debugging and system verification. It also provides facility of information hiding.
- All the data and program are hidden from other modules. Debugging is done at first layer. While debugging, in case an error is found, the error must be on layer which is currently debugging , because the layers below it have already finished debugging. Every lower layer helps to hide the existence of certain data structures, operations and hardware from higher-level layers.
- Careful definition and design of the layers is necessary in layered approach, as a layer is able to use only the layers below it. However layered approaches sometimes tend to be less efficient than other methods.

- **Advantages:**

1. **Modularity:** This design promotes modularity as each layer performs only the tasks it is scheduled to perform.

2. **Easy debugging:** debugging is easy because discrete structuring of layers. For example, if an error occurs in the CPU scheduling layer, the developer is able to debug that particular layer instead of all, unlike the Monolithic system where all the services are present together which makes debugging difficult.

3. **Easy update: A** modification or update in a particular layer does not affect the other layers.

4. **No direct access to hardware:** The innermost layer present in the design is hardware layer hence user can use the services of hardware layer but cannot directly modify or access it, unlike the Simple system in which the user had direct access to the hardware.

5. **Abstraction:** Every layer is responsible for its own functions. So, implementations and functions of the other layers are abstract to it.

- **Disadvantages:**

1. **Complex and careful implementation:** The arrangement of layers needs to be done carefully so that layers above can access layers below them efficiently. For example, the backing storage layer uses the services provided by memory management layer. So, it has to be below memory management layer. This required the arrangement to be modular and great modularity makes implementation process complex.

2. Slower in execution: When a layer interacts with any other layer, the request or massage sent by source layer has to travel through all the layers present in between the source and destination (communicating) layers, which increases response time, this is not the case with Monolithic system which is faster.

2.3.3 Microkernel Operating System

- Microkernel is a software or code which contains minimum number of functions, data, and features to implement an operating system. It provides a minimal number of mechanisms, which is good enough to run the most basic functions of an operating system. as it does not impose a lot of policies, it allows other parts of the operating system to be implemented.
- Microkernels and their user environments are usually implemented in the C++ or C programming languages with a little bit of assembly. Though, with some high-level coding other implementation languages are possible.
- Microkernel fulfils basic operations like memory, process scheduling mechanisms, and inter-process communication.

- The software Which executing at the privileged level is the only Microkernel. Most of the other functionalities of the OS those were important are removed from the kernel-mode and run in the user mode. Some of these functionalities are file servers, device drivers, application, inter-process communication, etc.
- The structure of microkernel is shown in following figure

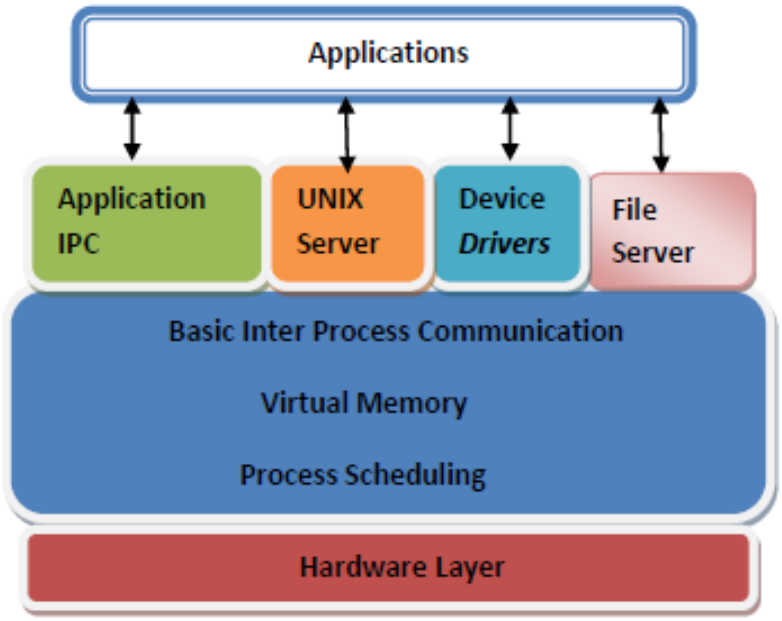


Figure 2.4 Microkernel structure

- **Components of Microkernel**

Only the core functionalities of the system are comprised by a microkernel. If putting component outside would interrupt the functionality of the system, then only it is included in the Microkernel only. All other non-essential components should be put in the user mode.

- **The minimum functionalities required in the Microkernel are:**

1. Memory management mechanisms like address spaces should be included in the Microkernel. It also contains memory protection features.
2. Processor scheduling mechanisms should contain process and thread schedulers.
3. Inter-process communication manages the servers that run their own address spaces.

- **Advantages of Microkernel**

1. Microkernel can function better because architecture is small and isolated.
2. Microkernel's are secure because only those components are included that disrupt the functionality of the system otherwise.

3. The expansion of the system is more accessible, so without disturbing the Kernel it can be added to the system application.
4. Microkernel's are modular, and the different modules can be replaced, reloaded, modified without even touching the Kernel.
5. As compared to monolithic systems Fewer system crashes in Microkernel.
6. Server malfunction is also isolated as any other user program's malfunction.
7. Increased security and stability will result in a decreased amount of code which runs on kernel mode

- **Disadvantage of Microkernel**

1. Compared to the normal monolithic system providing services in a microkernel system are expensive.
2. Context switch or a function call needed when the drivers are implemented as procedures or processes, respectively.
3. The performance of a microkernel system can be indifferent and may lead to some problems.

2.3.5 Exo-kernel Operating system

- Exo-kernel OS is developed by the Massachusetts Institute of Technology (MIT) works on the concept of giving control of OS to the applications. It allows application-level management of hardware resources. It separates the resource protection from management, to provide application-specific customization.
- As functionality of Exo-Kernel is limited to the protection and multiplexing of the raw hardware, non-availability of hardware abstractions on top of which applications can be constructed makes them different from other types of kernels. This separation of hardware protection from hardware management allows application developers to make the most efficient use of the available hardware for each specific program.
- The user mode processes running in Exo-Kernel can directly access kernel resources like process tables, etc.
- Exo-kernels are extremely small. But they always co-exist with library operating system, which provides complete functionalities of conventional OS to application developers.
- An important advantage of Exo-kernel-based OS is that they can include/integrate multiple library OS, each capable of exporting a different API, such as one for Linux and one for

Microsoft Windows, hence it makes simultaneous execution of both Linux and Windows applications possible.

Exo-kernels are very small in comparison with traditional kernels like Micro-kernels and Monolithic Kernels, and they provide direct access to hardware by removing unnecessary abstractions.

2.4 Virtual Machine

- **Virtual Machine** separates the hardware such as CPU, disk drives, memory; NIC (Network Interface Card) etc. Of a personal computer into various execution environments according to requirements therefore giving a feeling that every environment is a single computer. For example, VirtualBox.
- When different processes are executed on OS, it creates an illusion that each process is running on a different processor having its own virtual memory, this is achieved using CPU scheduling and virtual-memory techniques. The virtual machine approach does not provide additional functionalities like system calls and a file system unlike processes but only provides an interface same as basic hardware. Each process is made available a virtual copy of the underlying computer system.
- Virtual machines can be created for multiple reasons; most are closely related to the ability of sharing the similar basic hardware and supporting different execution environments (Operating systems) simultaneously.
- Involvement of disk systems is considered the major drawback of virtual-machine method. For example, let's assume a physical machine having three disk drives wants to support seven virtual machines. In this scenario allocating a disk drive to each virtual machine is not possible, as virtual-machine software will need considerable disk space to provide virtual memory and spooling for its own. That's why virtual disks are needed to fulfill the additional requirement.
- Hence users are provided with their own virtual machines. Using which they can run any of OS or software packages available on the underlying machine. The main focus of virtual-machine software is multi-programming where multiple virtual machines running onto a single physical machine, but it does not need to consider any user-support software. This arrangement can be a useful approach to divide the problem of designing a multi-user interactive system, into two smaller modules or pieces.

Advantages:

1. As each virtual machine is completely isolated from other virtual machines the protection issues are reduced to great extent.
2. Virtual machine can provide an instruction set architecture different from real computers.
3. Can be maintained easily, and provides availability and convenient recovery.

Disadvantages:

1. In case of multiple virtual machines running simultaneously on a host computer, failure or any error in one virtual machine can affect other running virtual machines, depending on the workload.
2. In terms of hardware access virtual machines are not as efficient as real machines.

2.5 Booting

Whenever a computer is turned on the first start-up sequence that starts the **operating system** of a computer is called a **Booting process**. It's a sequence of operations called Basic input output sequence (BIOS) that every computer needs to perform in order to load OS, Initialize hardware and software components. For successful Boot Sequence all the components of a computer system need to work properly or the boot sequence may fail.

System Boot Process

The steps involved in boot sequence are depicted in following diagram

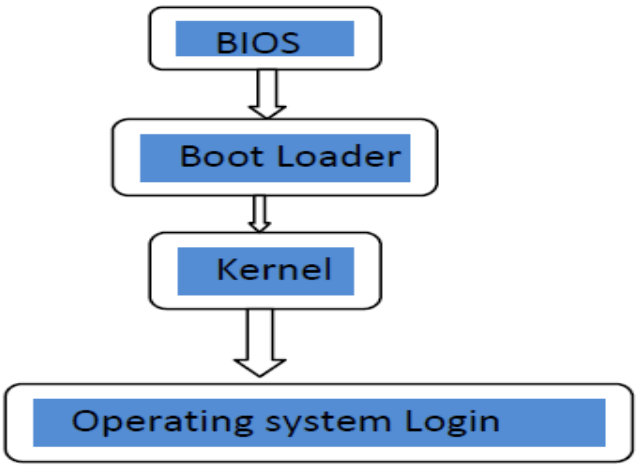


Figure 2.5: System boot process

Here are the steps −

- When the computer is turned on first The CPU initializes itself. This is achieved by the system clock which triggers a series of clock ticks to initialize CPU.
- Immediately after that CPU obtains first instruction of startup program which is stored in system's ROM BIOS. The role of First Instruction is to instruct the system to execute a predetermined POST (Power on Self-Test) from a fixed memory address.
- The first step performed by POST is checking of BIOS chip, CMOS RAM and detection of any kind of battery failure. In case of no battery failures, it continues to initialize CPU.
- In Next step the hardware devices, secondary storage devices like hard drives, ports, mouse, and keyboard are checked by POST to make ensure they are working properly.
- After making sure all the components are working properly by POST, the BIOS starts operating system loading process.
- Most of computer systems stores and loads OS from the C drive onto the hard drive. It is the task of CMOS chip to tell the BIOS the location of OS.
- The boot sequence is the order in which CMOS Looks or searches the different drives to find OS. This sequence can be changed by changing the CMOS setup.
- Once CMOS find appropriate boot drive, it informs the BIOS and then BIOS finds the boot record which starts the beginning of the OS booting process.
- Once the OS is initialized, the BIOS copies files into the memory. After that OS the one who controls the boot process.
- In the end, the OS checks final inventory of the system memory and the device drivers required to control the peripheral devices are loaded. After this the access to system applications is grated to users to perform various tasks.

If the system boot process is not available, the computer users have to download all the software components, including the ones which are unnecessary or not used much. Where the system boot only loads software components which are legitimately needed and unnecessary components are not loaded. This process frees up a lot of memory space and hence saves a lot of booting time.

2.6 Exercise Questions

- List and explain different services of Operating system.
- What is System call? What are the uses of System calls?
- Explain Structure of Operating system.
- Write a short note on Virtual machine.
- Explain in short Exo-kernel Operating system.
- Describe advantages and disadvantages of Virtual machine
- Write a short note on booting.
- Explain in short Microkernel Operating System.
- Explain in short layered Operating System.

Unit 3 Information Management

Objectives:

- To study concept of Information Management in operating system.
- To study concept of Disk Basics
- To study Direct Memory Access
- To study File System in Operating system
- To study Device Drivers concepts and its role in operating system.

3.1 Introduction

Different types of services of operating system which provides for reading and writing records are to be there. Operating system can be measured to be a collection of various such callable programs or services. These services are categorized by three heads.

- Information Management (IM)
- Process Management(PM)*
- Memory Management(MM)

So, let's focus more on Information Management in this chapter.

Information management (IM) is the collection and management of information from one or more sources and the allocation of that information to one or more addressees. This sometimes involves those who have a chance in or a right to that information.

Operating system manages the information or data, or managing the information from which it should be sent or received. File system in operating system keeps the track of information, its location and everything.

It also decides on which user should get resources first and put into effect the protection requirements and also provides accessing routines. Also, necessary information like opening a file, reading a file is taken care by the file system and finally it retrieves the resources like closing a file.

The organization of the information in terms of directories and files, allocating and deallocating the sectors of different types of files, maintaining and enforcing the access controls to ensure that only the right people can have access to the information, and driving various devices are done by these types of system services. These are normally provided by a single user operating system (i.e.,

Windows 95) and also multi user operating system (i.e., Windows 2000, UNIX, Virtual Memory system/VMS, multiple virtual storage /MVs). So, some of the system calls in this category are listed in following Figure.

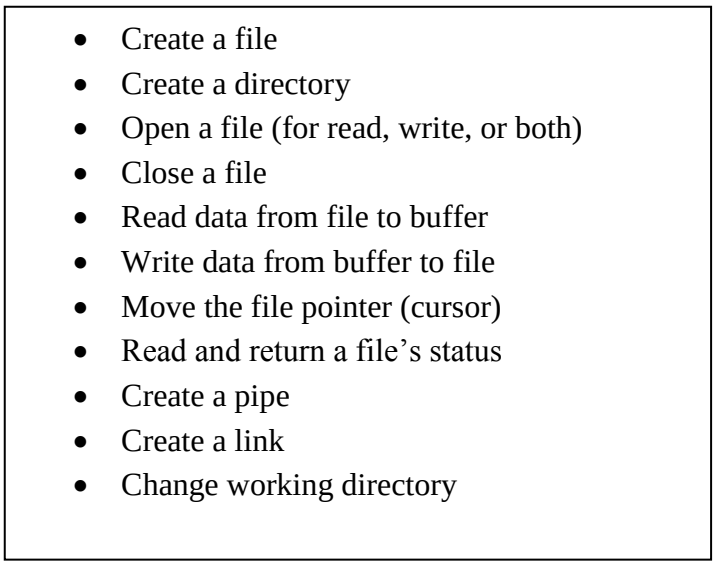


Figure 3.1 System calls related to Information Management

3.1.1 Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. Let us take an example to understand same, to return the current time and date, most systems have a system call. Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on. In addition, system calls are used to access this information and the operating system maintains information about all its processes. Generally, calls are also used to reset the process information using the get process attributes and set process attributes methods.

3.2 Disk Basics

Alternatively known as a floppy disk, a disk is a hard or floppy round, flat, and magnetic platter capable of having information read from and written to it. The most frequently found disks with a computer are the hard disks and floppy disks.

Disk is a very important I/O intermediate used by operating system frequently so it is necessary to understand the functions of it. The operating principle of a floppy disk is similar to that of hard

disk. And a hard disk can be considered as being made of multiple floppy disks put one above the other.

Figure 3.2 Disk (Source: Google/ computerhope.com)

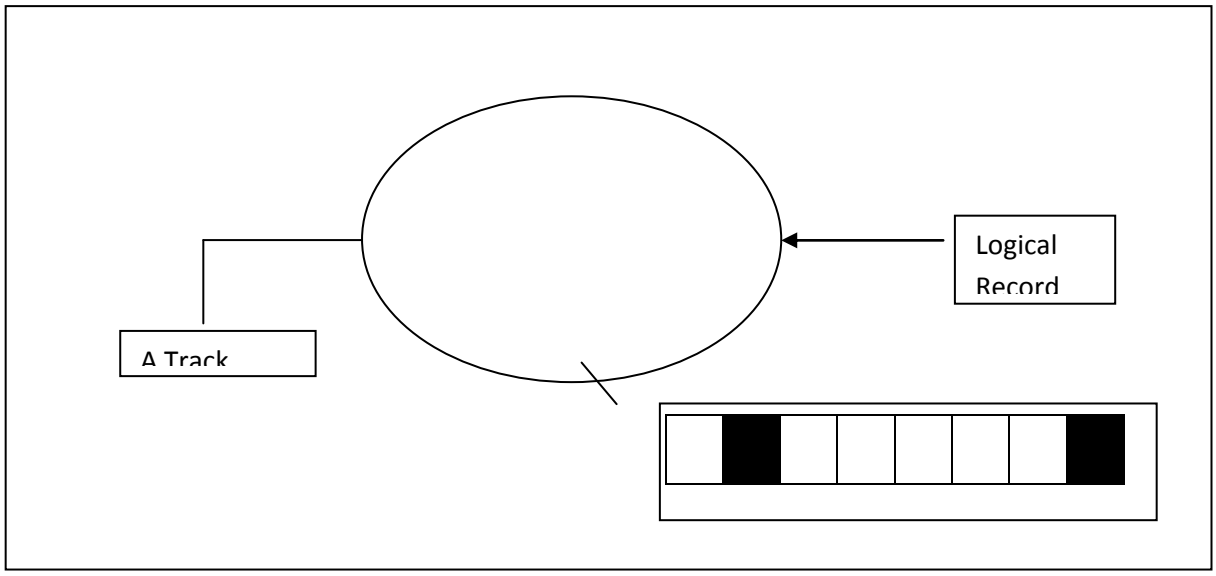


Figure 3.3 Data Recording on the Disk

Disk is nothing but long play music records except that the recording is done in concentric circles and not by spirally. A Floppy disk is made up of a round piece of plastic material, coated with a magnetized recording material. The surface is made up of concentric circles which are called as tracks. Data is recorded on these tracks in bit and it contains magnetized particles of metal is having north and South Pole. It is having only two directions so each particle can act as a binary value of 0 or 1 and 8 switches can record a character with the coding methods (ASCII). A logical record which consists of different fields (data items), each consisting of several characters which is stored in floppy disk.

A round plate which is used to store encoded data is called as Dis. The disks are divided in two types called as *magnetic disks* and *optical disks*.

3.2.1 Magnetic disks

Data is encoded on top of magnetic disks as microscopic magnetized needles on the disk's surface. One can record and erase data on a magnetic disk any number of times, just as with a cassette tape. Magnetic disks are available in different types as follow:

- **Floppy disk**: A typical 5-inch floppy disk can hold 360K or 1.2MB (megabytes). 3 -inch floppies normally store 720K, 1.2MB or 1.44MB of data. Floppy disks or outdated today, and are found on older computer systems.
- Hard **disk:** Hard disks can store data from 20MB to more than 1-TB (terabyte). Hard disks are also 10 to 100 times faster than floppy disks.
- **Removable cartridge**: Hard disks covered in a metal or plastic cartridge called as Removable cartridges, they are removable, they can remove them just like a floppy disk. Removable cartridges are very fast, though usually not as fast as fixed hard disks.

3.2.2 Optical disks

Optical disks record data by burning microscopic holes in the surface of the disk with a laser. To read the disk, another laser beam shines on the disk and detects the holes by changes in the reflection pattern.

Optical disks come in three basic forms:

 CD-ROM: Most optical disks are read-only. They come with already filled data, when someone purchases it. It only allows reading the data from a CD-ROM, modify, delete, or writing of new data is not allowed in CD-ROM.

 WORM: Stands for write-once, read-many. WORM disks can be written on once and then read any number of times; however, you need a special WORM drive to write data onto a WORM disk**.**

 Erasable optical (EO): EO disks allow to read, to written, and to erased the data just similar to magnetic disks.

The machine that turns a disk is called a disk drive. Each disk drive is mounted with one or more heads which are called as read/write heads, which are responsible to read and write the data.

Accessing data from a disk is much slower as compared to accessing data from main memory, but disks are much cheaper. Disks are non-volatile source of memory as they hold the data even when the computer is turned off. Which make disks more popular storage medium choice for storing various types of data. Another storage medium is magnetic tape. Tapes are sequential-access devices and are used only for backup and archiving the data. In tape to access data in the middle of a tape, the tape drive must pass through all the previous data.

A disk can be considered several surfaces, each of which consisting number of tracks which is shown in figure 3.4. The tracks are normally numbered from 0 as outermost track, with the number increasing inwards. Each track is divided into a number of sectors which are having equal size. And a sector can store up to 512 bytes. In double sided floppies it has two surfaces or slides and data can be recorded on two surfaces. So, a given sector is specified by three components of an address made up of surface number, track number and sector number.

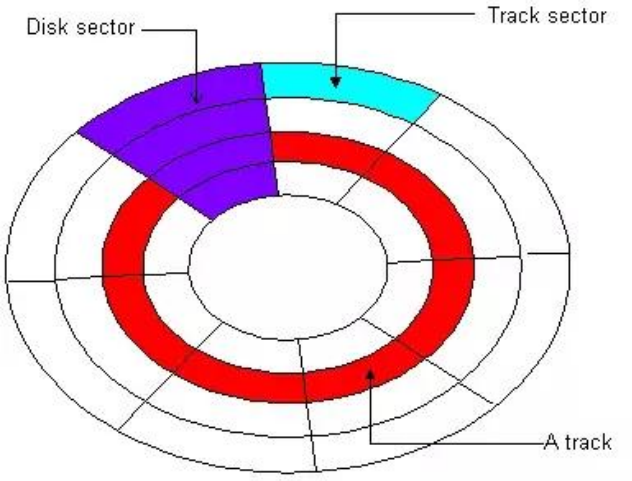


Figure 3.4 Tracks and Sectors (Source: installsetupconfig.com)

3.3 Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. The operating system would spend most of its time handling these interrupts, if a fast device such as a disk generated an interrupt for each byte. So, to reduce this overhead a computer uses direct memory access (DMA) hardware.

Direct Memory Access (DMA) Is the permission or authority granted by the CPU to read from or write to memory without involvement. The exchange of data between main memory and the I/O device is controlled by DMA module itself. Only after entire block has been transferred CPU is get involved at the beginning and end of the transfer and interrupted.

Let us understand with this an example, data stored in the computer's RAM need to be accessed by a sound card, but since it can process the data itself, it may use DMA to bypass the CPU. hat DMA is also supported by Video cards and can also access the system memory and process graphics without needing the CPU.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

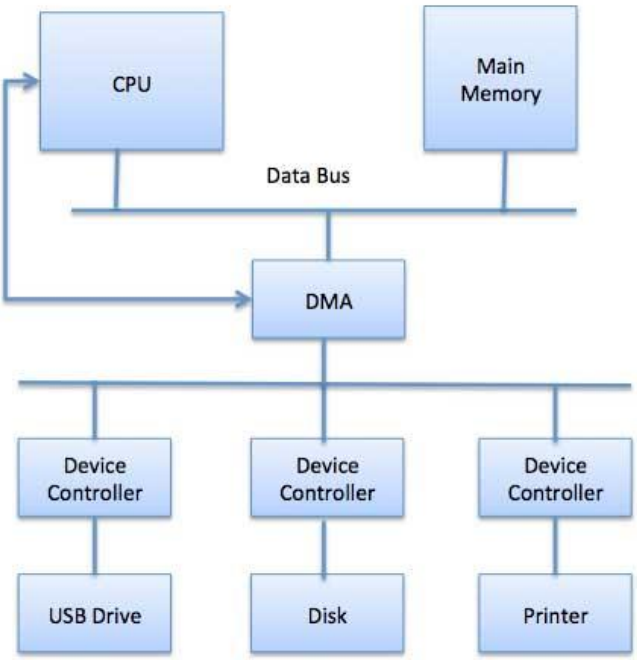


Figure 3.5 Direct Memory Access

3.3.1 Block diagram of DMA controller and DMA Operations

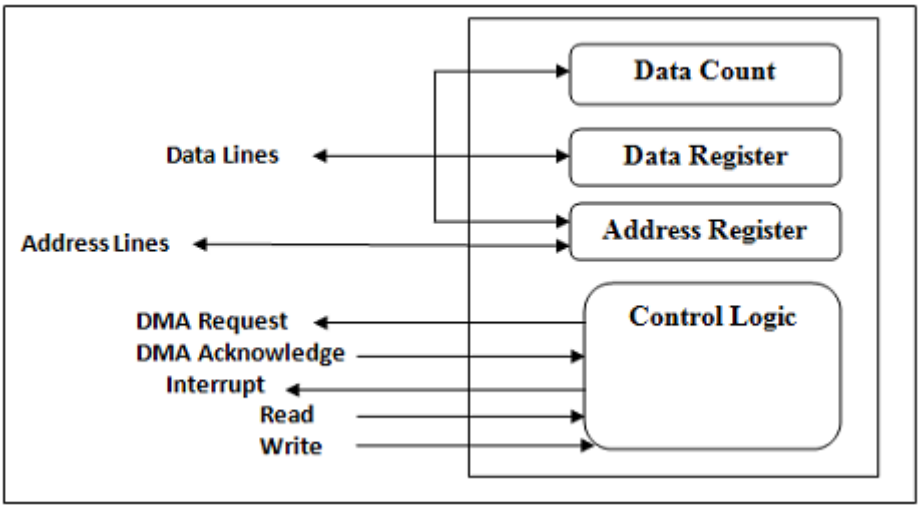


Figure 3.6 Block Diagram of DMA Controller

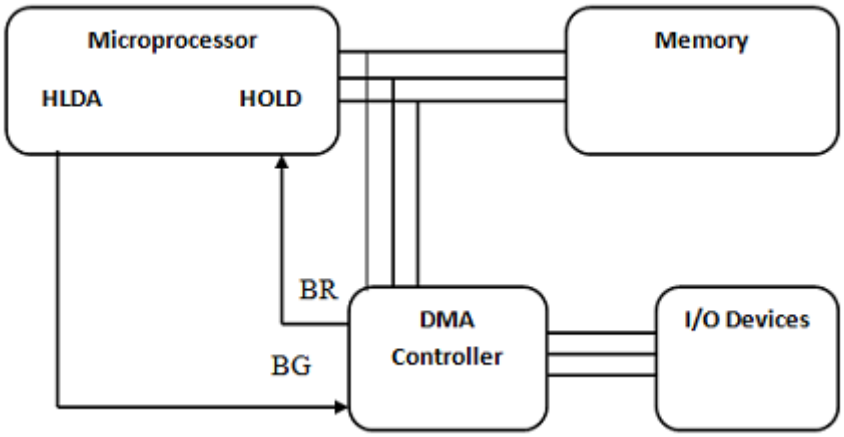


Figure 3.7 DMA Control Operations

3.3.2 DMA Operation:

Direct Memory Access involves transfer of data between I/O devices and memory by an external circuitry system called DMA controller without involving the microprocessor. However, microprocessor itself initiates the DMA control process by providing starting address, size of data block and direction of data flow. To deals with the control functions during DMA operations such as read, write and interrupt DMA contains a control unit. The address register of DMA controller is used to generate address and select I/O device to transfer the data block. The Count registers counts and hold no. of data block transferred. The direction of data transfer is also specified by it.

Steps of DMA Operation:

a) In order to occur an DMA operation by sending a control signal HOLD to the control line, the DMA controller first make a bus request (BR).

b) The microprocessor completes the current instruction execution on receiving the BR through HOLD pin high and then afterward it generates HLDA control signal and sends it to the DMA-Controller. This event switches over the control from microprocessor to DMA Controller. The microprocessor gets idle.

c) As soon as DMA controller receives HLDA (Hold Acknowledged) through Bus Grant (BG) line, it takes the control of system bus and start transferring the data blocks between memory and Input / Output devices, without involving the microprocessor.

d) On completion of data transfer, the DMA controller sends a low signal to the HOLD pin and hence microprocessor makes the HLDA pin low and takes the control over system bus.

DMA Operation Modes:

The DMA Controller operates under three modes:

A**. Burst Mode**: Irrespective of microprocessor requiring the bus, here DMA controller switch over the control to the microprocessor only on completion of entire data transfer. Microprocessor has to be idle during the data transfer.

B. **Cycle-Stealing Mode**: DMA controller hand over the control to microprocessor on transfer of every byte, thereby microprocessor gets the control and become able to process highly prioritized instruction. DMA need to make the BG request for each byte.

C. **Transparent Mode**: In this mode, when microprocessor is executing such instruction that does not requires system bus utilization only then DMA controller can transfer data blocks.

3.3.3 Direct Memory Access Advantages and Disadvantages

Advantages:

1. The read-write task will be speed up when the data will be transferred without the involvement of the processor
2. To read or write a block of data DMA **reduces the clock cycle** requires.
3. **The overhead** of the processor can be **reduced by** Implementing DMA.

Disadvantages

1. It would **cost** to implement a DMA controller in the system, as it is a hardware unit.
2. Cache **consistency** problem can occur while using DMA controller.

3.4 File System

In our daily lives we use files. Normally a file contains records of similar types of information. E.g., Employee file or Sales file or Electricity bill file. If anyone wants to automate various manual functions then the computer must support a facility for a user to define and manipulate the files. And the operating system does these things.

3.4.1 Block and Block numbering Scheme

The smallest amount of data in database that a program can request is called as Block. The smallest amount of data that can be retrieved from storage or memory which is a multiple of an operating system block. Multiple blocks in a database comprise an extent.

The operating system looks at a hard disk as a series of sectors, and numbers them serially starting from 0 to 1. One of the possible ways of doing this is shown in following figure which shows a hard disk.

Let us consider all the tracks on different surfaces of the same size, due to the obvious similarity in shape, we can think of them as a cylinder. In such a case, a sector address can be thought of us having three components such as: Cylinder number. Surface number, Sector number. In the earlier scheme where the address consisted of Surface number, Cylinder number is same as track number used. Track number and Sector number. Therefore, both these schemes are equivalent. We can observe in the figure which describes four platters and therefore, eight surfaces, with 10 sectors per track. The numbering starts with 0 at the outermost cylinder and topmost surface. We will assume that each sector is numbered anticlockwise so that if the disk rotates clockwise, it will encounter sectors 0, 1.2 etc., in that sequence.

We go to the next surface below on the same platter and on the same cylinder, when all the sectors on that surface on that cylinder are numbered. This surface is made analogous or akin to the other side of a coin. We continue with other platters for the same cylinder in the same fashion, as soon as both the surfaces of one platter are over. Once the full cylinder is over the next step is to start with the inner cylinder, and continue from the top surface again. Hence according to this scheme, Sectors 0 to 9 will be on the topmost surface (i.e., surface number = 0) of the outermost cylinder (i.e., cylinder = number = 0). Sector 10 to 19 will be on the next surface (at the back) below (i.e., surface number = 1), but on the same platter and the same cylinder (i.e., cylinder number = 0). Continuing this, with 8 surfaces (i.e., 8 tracks/cylinder), we will have Sectors 0-79 on the outermost cylinder (i.e., Cylinder 0). When the full cylinder is over, we start with the inner cylinder, but from the top surface and repeat procedure. Therefore, the next cylinder (Cylinder = 1) will have sector 80 to 159, and so on.

Likewise, we can view the entire disk as a series of sectors starting from 0 to n as Sector Numbers (SN) as shown.

0	1	2	3…….	N

3.4.2 File support Levels

For an application program the translation from logical to physical level is the responsible of operating System. In those days, an application programmer had to specify Operating System the actual disk address (cylinder, surface, and sector) to access a file. In order to access a specific customer record, one needs to write routines to keep track of where that record resided and then he had to specify this address. The scheme had a lot of problems in terms of security, privacy and complexity, and each application programmer had a lot of work to do.

Eventually, the translation from the logical to the physical level has to be done by someone. The question arises who will do this? Whether the Operating System does it or the Application Program does it? In answering this question, the existing Operating Systems have a great deal of differences. Some (like UNIX) treat a file as a Sequence of bytes. This is the one of the important spectrums where the Operating System provides the minimum support. In such scenario, the Operating System fails to recognize the concept of a record.

Therefore, the record length is not maintained as a file attribute in the file system of such an Operating System like UNIX or Windows 2000. So, an instruction such as fread" in C or "Read… record" in COBOL and many more cannot be understood by such operating system. It only understands an instruction "Read byte numbers X to Y". Therefore, something like the application program or the DBMS uses which has to do the necessary conversion. At a little higher level of support, some others treat files as consisting of records of fixed or variable length, like AOS/VS). In this case, along with the other information about the file Operating System keeps the information about record lengths etc.

3.4.3 Writing/Reading a Record

3.4.3.1 Writing a Record

File system is responsible for translating the address of logical records into physical address. So how this is achieved by taking simple example of a sequential file i.e., customer records. Let us see step by steps.

1. **Arrangement of logical records**: Consider customer record consists of 700 bytes. The application program is responsible for creating these records in HLL such as “WRITE CUST-REC” to achieve this. At the time of execution, this results in a system call to the operating system to write a record. These records may or may not be in any specific sequence (as customer number).

The OS assigns a Relative Record Number (RRN) to each of the record which is starting with 0, as these records are written.

Imagine all the customer records put one after another like carpet shown in following figure 3.8.

PRN=0 BYTES 0-699	PRN = 1 BYTES 700-1399	…… 	PRN = 9 BYTES 6300-6999

Figure 3.8: The PRN and RBN

If 10 customer records are there the 700*10 = 7000 bytes will be written onto the customer for PRN = 0 to 9. The operating system can calculate a Relative byte number (RBN) for every record. This is the starting byte number for each record. RBN is calculated with respect to 0 as the starting byte number and put after other as logical records. Following figure shows the relationship between RRN and RBN for the records shown in above figure.

RRN	**RBN**
0	0
1	700
2	1400
.	…
9	6300

Figure 3.9: The relation between RRN and RBN

2. Arrangement of Blocks: The second step is how o actually write these logical records into different types of blocks. Let us consider a disk with 8 surfaces (0 to 7). Each surface is having 80 tracks (0 to 79) and each track is having 10 sectors (0 to 9). So, the disk has 8*80*10 = 6400 sectors of 512 bytes each. 1 block = 1 sector = 512 bytes. So, the operating system looks at the disk as consisting of 6400 logical blocks (0 to 6399) each of 512 bytes.

Block = 0 Bytes 0 to 511	Block = 1 Bytes 512 to 1023	Block = 6399 Bytes 3276288 to 3276800

Figure 3.10: The operating system view on Disk

3. The allocation of blocks to a file: Operating system is acting kike arbitrator and is responsible for allocating/deallocating blocks to different files. Three ways are there, i.e., Contiguous, Indexed and Chained. If a user knows that he has 500 customers, but will never have more than 730, he must ask for 730*700/512 blocks = 998.05 or approximately 1000 blocks. Let us consider that the operating system has already allocated block numbers 0 to 99 for some other file and that blocks 100 and thereafter are free. The OS has to maintain the list of free blocks and it allocates block numbers 100 to 1099 for the customer file.

3.4.3.2 Reading a Record

How the records are to be read, let us consider by the Operating system on the request of the AP. An unstructured AP for processing all the records sequentially from a file would be shown in following figures 3.11 and 3.12 for C and COBOL. At the time of processing, it executes the "fread" or "READ" instruction respectively; the AP gives a call to the operating system which then reads a logical record on behalf of the AP. The operating system blocks the AP during this time, after which it is woken up.

```
ABC.
        Count = fread (&custrec, sizeof (custrec), 1 FP);
        if (count = = 0) goto EOJ;
        …
        /* calculate the balance, interest, etc. */
        Process_record ( );
        Goto ABC;
EOJ.
```

```
Exit ( );
```

Figure 3.11: A 3GL program to read and process records (C version)

```
ABC.
        READ CUST-REC…… AT END GO TO EOJ.
        …
        PERFORM PROCESS-REC.
                        (calculate the balance, interest etc.)
        GO TO ABC.
EOJ.
        STOP RUN.
```

Figure 3.12: A 3GL program to read and process records (COBOL version)

3.4.4 Relationship between the Operating System and DMS

Operating system can present to the Application Program (AP), records in the same sequence that they have been written. If other AP wanted to process the records in a sequence, say by customer number, it is the work of AP to ensure that they are presented to the operating system in that fashion so that they also retrieved**.**

If some records are to be selectively processed in a sequence, then it is required to maintain some data structure like an index for indicate that where a specific record is written. So, there is another piece of software to maintain and access these types of indexes and it is known as Data Management Systems (DMS).

The data management functions such as maintenance of an index, etc. were parts of operating system, but as these functions started getting more complex, separate DMS were written. The DMS can be either File Management System (FMS) or Database Management System (DBMS).

ORACLE, DB2, INFORMIX are some of the examples of DBMS. DMS is in between the AP and OS and which is responsible for maintaining all the index tables based on keys.

3.4.5 File Directory Entry

Group of files combined is known as directory. A directory contains all information about file, its attributes. The directory can be viewed as a symbol table that translates file names into their directory entries. Directory itself can be organized in many ways.

The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory. File directory entry describes files and directories. It is a primary critical directory entry and must be immediately followed by 1 Stream Extension directory entry and from 1 to 17 File Name directory entries.

For each file created by the operating system, the OS maintains a directory entry which is also known as Volume Table of Contents (VTOC) in IBM. Let us consider following figure shows the possible information that the operating system keeps at a logical level for each file. Physically, it could be kept as one single record or multiple records, where access control information constitutes one small record. The logical records of VTOC or file directory entries are stored on the disk using some hashing algorithm on the file name. Alternatively, an index on the file names can be maintained to allow faster access of this particular information once the file is supplied. But in UNIX, the hashing technique is not used. All entries are created sequentially or in first available empty slot. So consistently the operating system goes through the entire directory starting from the first entry to access the directory entry given a file name. The algorithm is very simple and time consuming for execution.

File Name
File Extension
File size – Current and maximum
File Usage Count
Number of processes having this file opens
File Type (binary, ASCII, etc)
File Record Length (If records are recognized)
File Key length, positions (If ISAM is a part of operating system)
File organization (Sequential, Indexed, Random)
File Owner, Creator (Usually the same)
File Access Control Information
File Dates (Creation, last usage etc.)
Other information
File Address (Block Number) of the first block

The significance for address translation is the file address field for every file. This signifies the address (i.e., block number) of the first block allocated to the file. If the allocation is adjacent, finding out the address of the subsequent blocks is very easy. And if the allocation is chained or indexed, the operating system has to pass through the data structure to access the subsequent blocks of the same file.

3.4.6 Open/Close Operations

File operation are simply those things which user can perform on a file. For example, user can create file, save a file, open a file, read a file and modify a file. OS can provide system call for performing various operations on the file are: create, write, read, delete, re positioning and translating.

Following are same list of operations.

1. Create.
2. Write.
3. Close.
4. Open.
5. Read.
6. Delete.

All these operations require that the directory structure be first search for the target file.

Open ( )

- A file can be opened in one of the two modes, read mode or write mode.
- In read mode, the OS does not allow anyone to alter data; file opened in read mode can be stored among several entities.
- Write mode allows data modifications, file opened in write mode can be read but cannot be shared.

Close ( )

- This is the most important operation from OS point of view.
- When a request to close a file is generated the OS.
- OS removes all the block (in shared mode)
- Saves the data if altered to the secondary storage media.
- Releases all the buffer and file handlers associated with a file.

3.4.7 Disk Space 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 Contiguous Allocation scheme each file occupies a contiguous set of blocks on the disk. Let us understand with an 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.

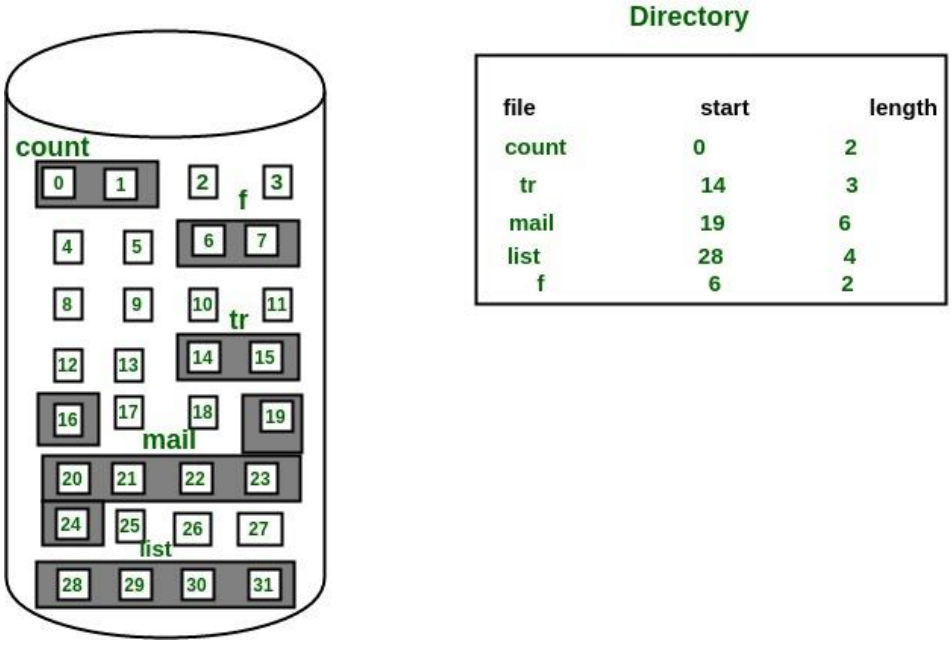


Figure 3.14: Contiguous Allocation

Advantages:

- This scheme supports both the Sequential and Direct Accesses. In case of direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- As the number of seeks are minimal because of contiguous allocation of file blocks, This scheme is extremely fast.

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

This scheme has a linked list of disk blocks for each file which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.

A pointer to the starting and the ending file block is contained in the directory entry. Each block contains a pointer to the next block occupied by the file.

How the blocks are randomly distributed in the file ‘jeep’ is shown in following image. The block (25) which is the last bloc contains -1 which indicating a null pointer and does not point to any other block.

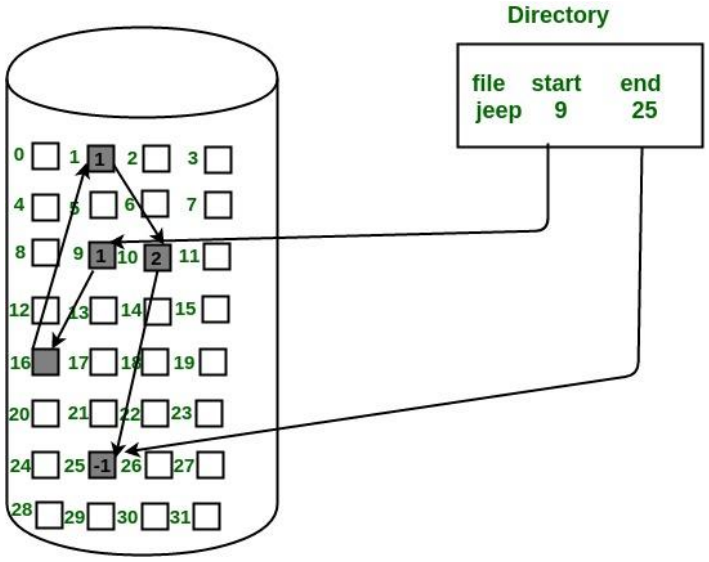


Figure 3.15: Linked List Allocation

Advantages:

- It is very efficient and flexible scheme in terms of file size. As the system does not have to look for a contiguous chunk of memory, file size can be increased easily.
- This method does not have to worry about external fragmentation, which makes it comparatively better in terms of memory utilization.

Disadvantages:

- A large number of seeks are needed to access every block individually, because of the file blocks are distributed randomly on the disk, which makes linked allocation slower.
- Random or direct access is not supported by it. We cannot directly access the blocks of a file. To access any block k can be made possible by traversing k blocks sequentially from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

Indexed Allocation have a special block contains the pointers to all the blocks occupied by a file known as the **Index block**. Each file has its own index block. The disk address of the i^{th} file block

is contained in the i^{th} entry in the index block. The address of the index block is contained by the directory entry as shown in the image:

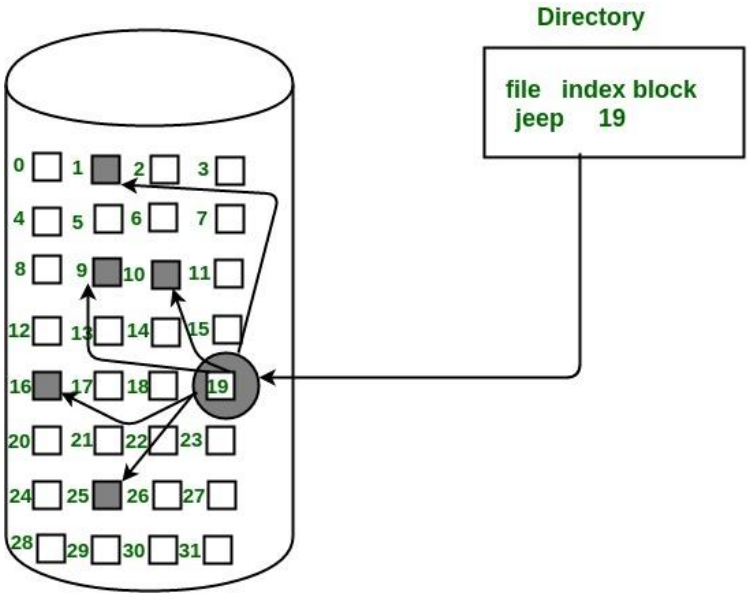


Figure 3.16: Indexed Allocation

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- The problem of external fragmentation is overcome by this scheme.

Disadvantages:

- For indexed allocation the overhead of the pointer is greater than linked allocation.
- The files that expand only 2-3 blocks that is very small files for such files, 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.

Single index block may not be able to hold all the pointers, for files that are very large. Following mechanisms can be used to resolve this:

1. **Linked scheme:** For holding the pointers this scheme links two or more index blocks together. A pointer or the address to the next index block is contained by every index block.
2. **Multilevel index**: In this policy, to point to the second level index blocks a first level index block is used, which in turn points to the disk blocks occupied by the file. Depending on the maximum file size this can be extended to 3 or more levels.
3. **Combined Scheme**: In this scheme, the I node also called as information Node which is considered as special node contains all the information about the file such as the name, size,

authority, etc. and to store the Disk Block addresses which contain the actual file the remaining space of I node is used for that as shown in the image below. The first few of these pointers in I node point to the direct blocks i.e., the addresses of the disk blocks are stored in pointer that contain data of the file. The next few pointers point to indirect blocks. These Indirect blocks may be classified as single indirect, double indirect or triple indirect. Single Indirect block contain the disk address of the blocks that contain the file data but does not contain the file data. Just similar to Single Indirect block, double indirect blocks do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

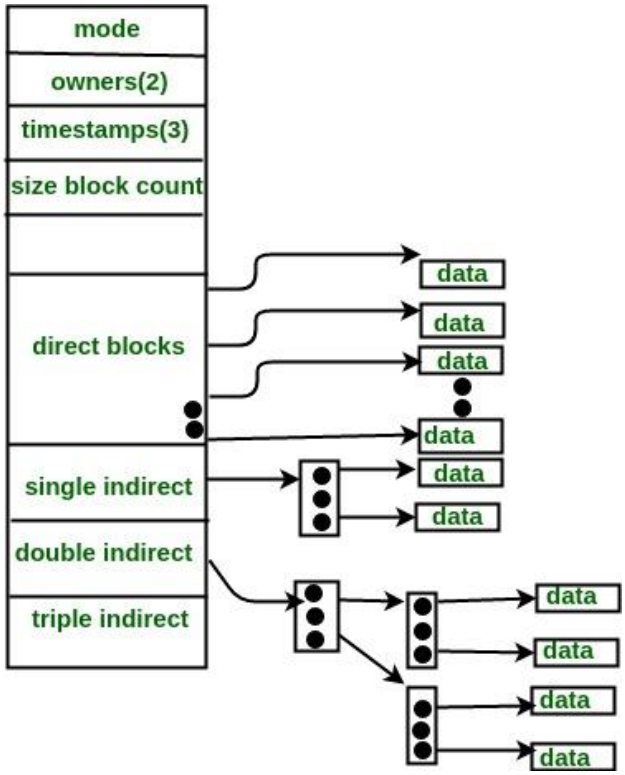


Figure 3.17: Combined Scheme

3.4.8 Directory Structure: User's View

By considering the user' s viewpoint, a directory is a file of files, i.e., a single file containing details about other files belonging to that directory.

The collection of the correlated files on the disk is called as a Directory. In other words, one can say, a container which contains file and folder is called directory. The complete file attributes or some attributes of the file can be stored in a directory. A directory can be comprised of various

files. One can maintain the information related to the files which is possible with the help of the directory.

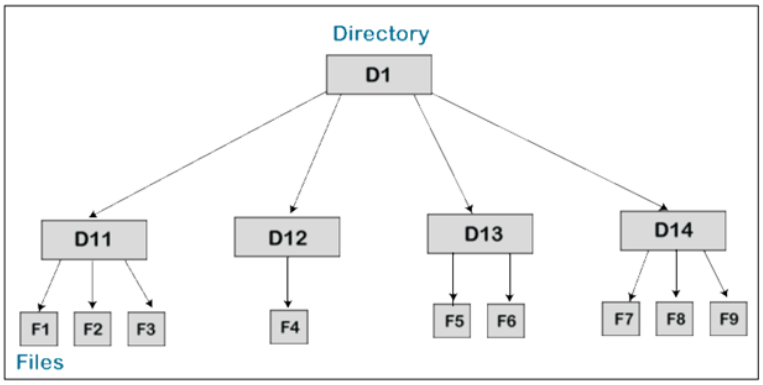


Figure 3.18: Directory Structure

We can divide the hard disk into multiple partitions of different sizes, this can be done to take the advantages of various file systems on the different operating systems. Partitions are known as minidisks or volumes.

Each partition must contain at least one directory. Through it, we can list all the files of the partition. For each file there is a directory entry in the directory which is maintained, and in that directory entry, all the information related to the file is stored.

A directory stores various types of information as mentioned bellow:

1. Name
2. Type
3. Location
4. Size
5. Position
6. Protection
7. Usage
8. Mounting

1. **Name: –** Name is the name of the directory, which is visible to the user.
2. **Type: –** Type of a directory means what type of directory is present such as single-level directory, two-level directory, tree-structured directory, and acyclic graph directory.
3. **Location: –** Location is the location of the device where the header of a file is located.
4. **Size: –** Size means number of words/blocks/bytes in the file.
5. **Position: –** Position means the position of the next-read pointer and the next-write pointer.
6. **Protection: –** Protection means access control on the read/write/delete/execute.

7. **Usage: –** Usage means the time of creation, modification, and access, etc.
8. **Mounting: –** Mounting means if the root of a file system is grafted into the existing tree of other file systems.

Operations on Directory

The various types of operations on the directory are:

1. Creating
2. Deleting
3. Searching
4. List a directory
5. Renaming
6. Link
7. Unlink

1. **Creating: –** In this operation, a directory is created. The name of the directory should be unique.
2. **Deleting: –** If there is a file that we don't need, then we can delete that file from the directory. We can also remove the whole directory if the directory is not required. An empty directory can also be deleted. An empty directory is a directory that only consists of dot and dot-dot.
3. **Searching: –** Searching operation means, for a specific file or another directory, we can search a directory.
4. **List a directory: –** In this operation, we can retrieve the entire files list in the directory. And we can also retrieve the content of the directory entry for every file present in the list.

If in the directory, we want to read the list of all files, then first, it should be opened, and afterwards we read the directory, it is a must to close the directory so that the internal table space can be free up.

Types of Directory Structure

There are various types of directory structure:

1. Single-Level Directory
2. Two-Level Directory
3. Tree-Structured Directory
4. Acyclic Graph Directory
5. General-Graph Directory

1. Single-Level Directory: – Single-Level Directory is the easiest directory structure. There is only one directory in a single-level directory, and that directory is called a root directory. In a

single-level directory, all the files are present in one directory that makes it easy to understand. In this, under the root directory, the user cannot create the subdirectories.

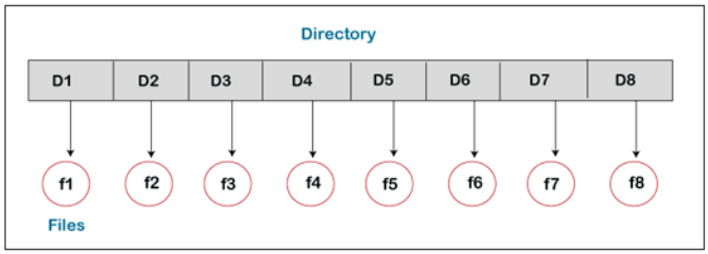


Figure 3.19: Single-Level Directory

2. **Two-Level Directory**: Two-Level Directory is another type of directory structure. In this, it is possible to create an individual directory for each of the users. There is one master node in the two-level directory that includes an individual directory for every user. At the second level of the directory, there is a different directory present for each of the users. Without permission, no user can enter into the other user's directory.

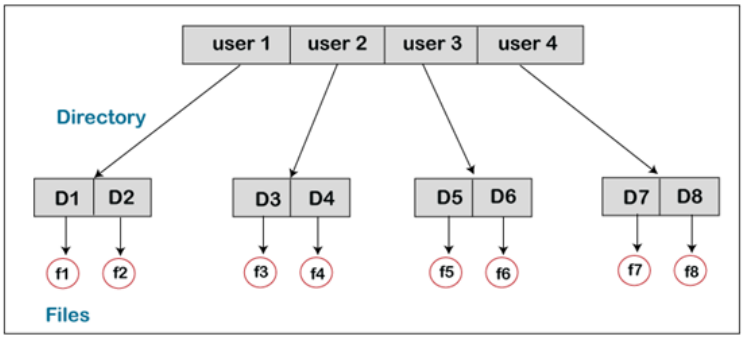


Figure 3.20: Two-Level Directory

3. **Tree-Structured Directory**: A Tree-structured directory is another type of directory structure in which the directory entry may be a sub-directory or a file. The tree-structured directory reduces the limitations of the two-level directory. We can group the same type of files into one directory. In a tree-structured directory, there is an own directory of each user, and any user is not allowed to enter into the directory of another user. Although the user can read the data of root, the user cannot modify or write it. The system administrator only has full access to the root directory. In this, searching is quite effective and we use the current working concept. We can access the file by using two kinds of paths, either absolute or relative.

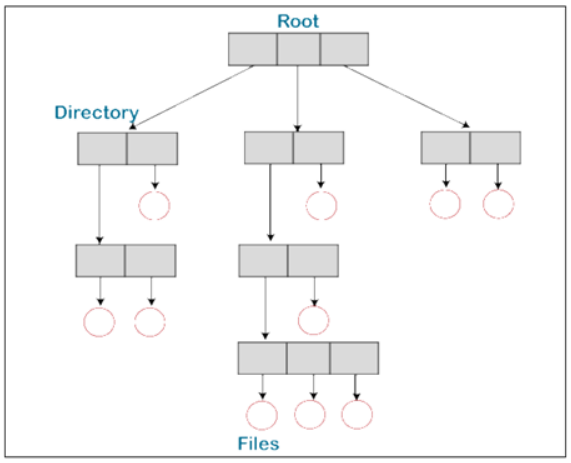


Figure 3.21: Three-Level Directory

4. Acyclic-Graph Directory: In the tree-structure directory, the same files cannot exist in the multiple directories, so sharing the files is the main problem in the tree-structure directory. With the help of the acyclic-graph directory, we can provide the sharing of files. In the acyclic-graph directory, more than one directory can point to a similar file or subdirectory. We can share those files among the two directory entries.

With the help of aliases, and links, we can create this type of directory graph. We may also have a different path for the same file. Links may be of two kinds, which are hard link (physical) and symbolic (logical).

5. General-Graph Directory: The General-Graph directory is another vital type of directory structure. In this type of directory, within a directory we can create cycle of the directory where we can derive the various directories with the help of more than one parent directory.

The main issue in the general-graph directory is to calculate the total space or size, taken by the directories and the files.

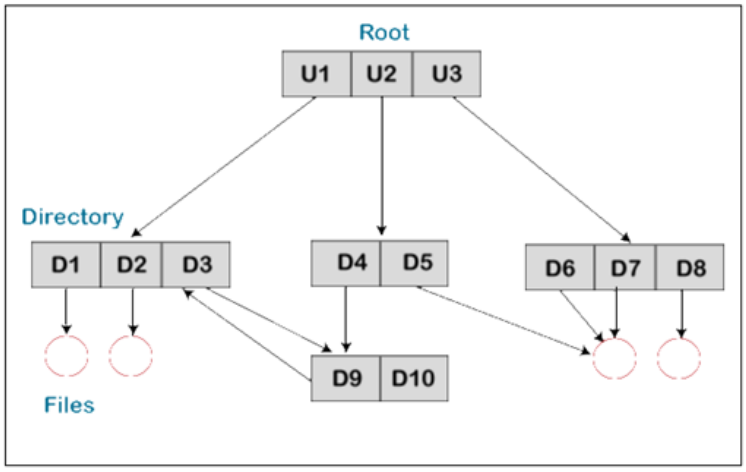


Figure 3.22: General-Graph Directory

3.4.9 Implementation of a Directory System

The selection of directory-allocation and directory-management algorithms significantly affects the efficiency, performance, and reliability of the file system.

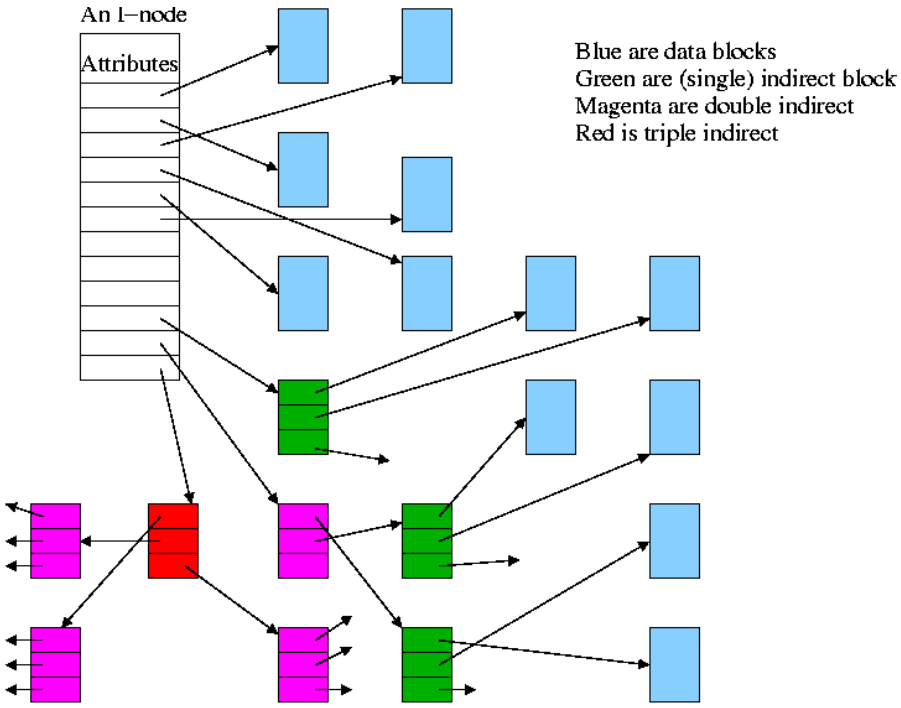


Figure 3.23: Implementation of a Directory System

Linear List

The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file. We must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it.

To reuse the directory entry, we can do one of several things. We can mark the entry as unused (by assigning it a special name, such as an all-blank name, or with a used-unused, bit in each entry), or we can attach it to a list of free directory entries. A third alternative is to copy the last entry in the directory into the freed location and to decrease the length of the directory. A linked list can also be used to decrease the time required to delete a file.

The real disadvantage of a linear list of directory entries is that finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow. In fact, many operating systems implement a software cache to store the most recently used

directory information. A cache hit avoids the need to constantly reread the information from disk. A sorted list allows a binary search and decreases the average search time.

However, the requirement that the list be kept sorted may complicate creating and deleting files, since we may have to move substantial amounts of directory information to maintain a sorted directory. A more sophisticated tree data structure, such as a B-tree, might help here. An advantage of the sorted list is that a sorted directory listing can be produced without a separate sort step.

Hash Table

Another data structure used for a file directory is a hash table. With this method, a linear list stores the directory entries, but a hash data structure is also used. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for collisions—situations in which two file names hash to the same location.

The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size. For example, assume that we make a linear-probing hash table that holds 64 entries. The hash function converts file names into integers from 0 to 63, for instance, by using the remainder of a division by 64. If we later try to create a 65th file, we must enlarge the directory hash table—say, to 128 entries. As a result, we need a new hash function that must map file names to the range 0 to 127, and we must reorganize the existing directory entries to reflect their new hash-function values. Alternatively, a chained-overflow hash table can be used. Each hash entry can be a linked list instead of an individual value, and we can resolve collisions by adding the new entry to the linked list. Lookups may be somewhat slowed, because searching for a name might require stepping through a linked list of colliding table entries. Still, this method is likely to be much faster than a linear search through the entire directory.

3.5 Device Driver

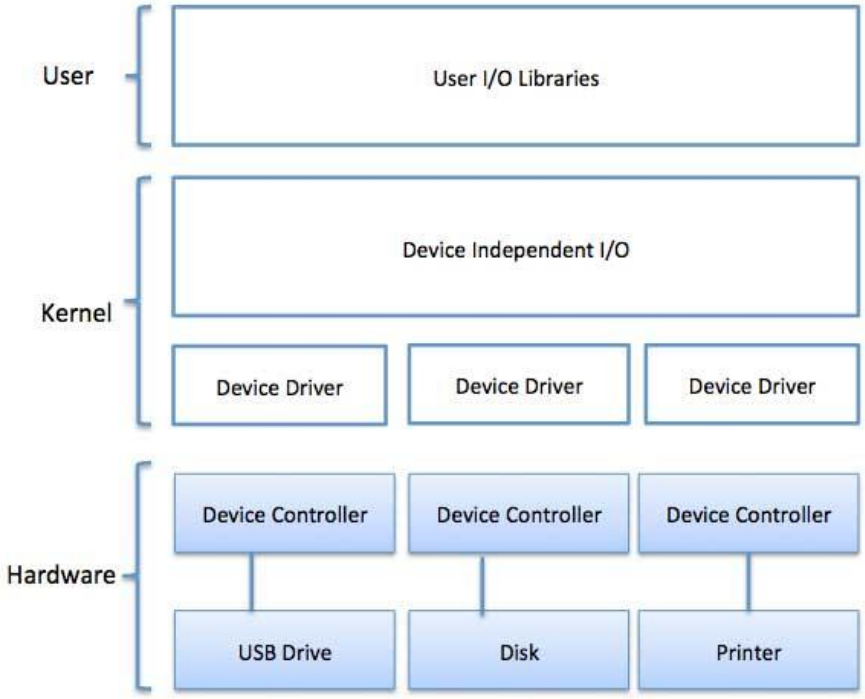


Figure 3.24 : Device Drivers

Device driver which are usually known to as a *driver* is a computer program that controls or operates a particular type of device that is attached to a computer. Without needing to know precise details of the hardware being used drivers enable operating systems and other computer programs to access hardware functions. As well driver provides a software interface to hardware devices. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

To handle a particular device, drivers are software modules can be plugged into an OS. To handle all I/O devices, operating System takes help from device drivers. Device drivers implement a standard interface in such a way that code contains device-specific register reads/writes, and they encapsulate device-dependent code. Device driver is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

3.5.1 Functions of Device Driver

Device drivers simplify programming by acting as translator between a hardware device and the applications or operating systems that use it. Programmers can write the higher-level application code independently of whatever specific hardware the end-user is using.

- It accepts abstract read and write requests from the device independent software above to it.
- It controls the power requirement and log event.
- It checks the status of the device. If device is in idle state, then request is processed and if devices are in busy state, then it queues the request which will process latter.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

3.5.2 Path Management

Different controllers require large memory buffers and they have complicated electronics so are expensive. There is an idea for reducing the cost is to have different types of devices attached to only one controller. At a single time, a controller can control one and only one device and that's why, only one device can be active even if some amount of parallelism is possible due to overlapped seeks. If there, are I/O requests from both the devices attached to the controller, one of them will have to wait. And all pending requests for such any device are queued in a device request queue by the device drivers. And the device driver creates a data structure and has the algorithm to handle the issue of queue. If the response time is very important, these I/O waits have to be reduced, and if one is ready to spend more money, one can have separate controller for each device. The connection will exist between each controller/device pair.

In some of the mainframe computers, the functions of a controller are very complex, and they are split into two units. One is Channel and other is known as Control Unit (CU). Channel sounds like a wire or a bus, but it is actually a very small computer with the capacity of executing only some specific I/O instructions. A controller normally controls devices of the same types, but a channel can handle controllers of different types and there exists multiple paths between the memory and the devices and these paths are symmetrical or asymmetrical.

Because of multiple paths, the complexity of the device driver routine increases, but the response time increases. One controller control two devices, the speed will be slower, if there was only one controller per device. And this is because, there is no issue of path management and waiting period until a controller gets free.

3.5.3 The Sub modules of DD

As the information management divided into two parts, i.e., file system and Device Drivers. Conceptually it is divided in four sub modules:

1. **I/O Procedure**: In order to perform the path management and to create the pending requests, the I/O procedure maintains the data structures by using Channel Control Block (CCB), Control Unit Control Block (CUCB), Device Control Block (DCB), and Input/output Request Block (IORB). The data structure is maintained in the memory and it is uploaded as new request arrives and it is serviced. One can consider a certain area into the memory dedicated to contain the IORB records. This area could have number of slots for the IORB records for all the devices, CUs and channels. The management of free slots, allocation and unlinking these IORBs to proper DCBs, CUCBs and CCBs will require complex algorithms which are main part of I/O procedure.
2. **I/O Scheduler**: Request for I/O operations pending for a device normally originates from different processes with different priorities. It seems that rational solution to the problem of scheduling is by the priorities of requesting processes. But this is generally not followed. The I/O operation is electromechanical and therefore, it is very slow. Hence. The whole emphasis of I/O scheduling is on reducing the disk arm movement, and consequently the seek time. This improves throughput and response time even if it might mean a little extra wait for a process with higher science, let us imagine that there are two processes waiting for a device to get free and there are Two IORBs for that device). Let us also assume that the process With the lower priority wants to read from the same track where the R/W arms currently are, therefore, requiring no seek time
3. **Device Handler**: The device handler basically is a piece of software which prepares an I/O program for the channel or a controller, loads it for them and instructs the hardware to execute the actual I/O. After this happens, the device handler goes to sleep. The hardware performs the actual I/O and on completion, it generates an interrupt. The ISR (Interrupt Service Routines) for this interrupt wakes up the device handler again. It checks for any

errors. If there is no error, the device handler instructs the DMA transfer the data to the memory.

4. **Interrupt Service Routines (ISR):** An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt. ISR examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value. If no further handling is required the ISR notifies the kernel with a return value. An ISR must perform very quickly to avoid slowing down the operation of the device and the operation of all lower-priority ISRs.

3.6 Exercise Questions

- Explain in short concept of Information Management in operating system.
- Write a short note on Disk Basics.
- Explain Direct Memory Access.
- Explain various File System in Operating system.
- Explain the role of Device Drivers in operating system.
- List and explain various Disk Space Allocation Methods**.**
- Explain in short Direct Memory Access Advantages and Disadvantages.
- Explain the sub modules of Device Drivers in operating system.

Unit 4 Process Management

Objectives:

- To study concept of process and its lie cycle.
- To study process properties and its hierarchy.
- To study Process Scheduling.
- To study Scheduling Algorithms.
- To study concepts of thread

4.1 Process

A process is basically a program in execution. A process must progress its execution in a sequential fashion.

An entity which represents the basic unit of work to be implemented in the system is called as process.

To make a process to perform all the tasks mentioned in the program, we write our computer programs in a text file and when we execute this program.

A program can be divided into four sections ─ stack, heap, text and data, When a is loaded into the memory and it becomes a process. A Simplified layout of a process inside main memory can be understood in the figure shown bellow −

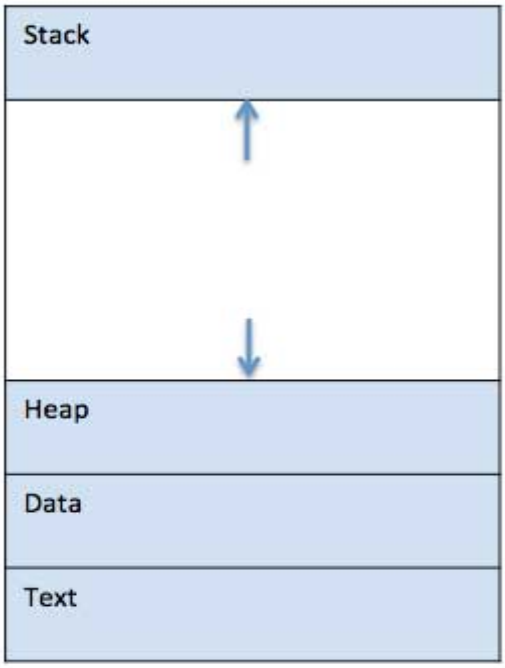


Fig 4.1 Process in Main Memory

Stack: The temporary data such as method/function parameters, return address and local variables etc is contained by the process Stack.

Heap: During run time of any process some memory is get allocated dynamically allocated to a process called as Heap.
Text: It includes the contents of the processor's registers and current activity represented by the value of Program Counter.
Data: The global and static variables are contained in this section.

4.1.1 Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. Let us see and understand a simple program to print message "Hello World" written in C programming language −

```
#include <stdio.h>
int main() {
   printf("Hello, World! \n");
   return 0;
}
```

A collection of instructions written to performs a specific task when computer program is executed by a computer. One can conclude that a process is a dynamic instance of a computer program, when we compare a program with a process.

An algorithm is a part of a computer program that performs a well-defined task. Whereas a collection of various computer programs, related data and libraries are called as a software.

4.1.2 Multi programming

There are usually several concurrent application processes in a modern computing system which want to execute, and to manage all the processes effectively and efficiently it is the responsibility of the Operating System.

To be a multi program is one of the most important aspects of an Operating System. There are multiple processes waiting to be executed and ready to run in a computer system, but they are waiting for CPU to be get allocated to them and when they can begin their execution. These processes are also known as jobs. To accommodate all of these processes or jobs into memory even is not possible, as the main memory is too small for this. Thus, these processes are initially kept in an area called job pool. All those processes awaiting allocation of main memory and CPU are kept in this job pool.

Out of all these waiting jobs, CPU selects one job and brings it from the job pool to main memory and starts executing it. The processor executes one job until it is interrupted by some external factor or it goes for an I/O task.

Non-multi programmed system's working –

- The CPU becomes idle in a non multi programmed system, As soon as one job leaves the CPU and goes for some other tasks (say I/O). Until the job which was executing earlier comes back and resumes its execution with the CPU, the CPU keeps waiting and waiting to other jobs. So, CPU remains free for all this while.
- Now CPU remains idle for a very long period of time which is the drawback off the system. As well CPU makes other jobs waiting for long time, as other jobs are waiting to be executed and might not get a chance to execute because the CPU is still allocated to the earlier job.

- Even though other jobs are ready to execute, CPU is not allocated to them as the CPU is allocated to a job which is not even utilizing it (as it is busy in I/O tasks) this poses a var serious problem.
- It is really a serious issue that one job is using the CPU for say 1 hour while the others have been waiting in the queue for 5 hours. Hence the concept of multi programming came up to avoid situations like this and come up with efficient utilization of CPU.

The main idea of multi programming is to maximize the CPU time.

Multi programmed system's working –

- In a multi-programmed system, the Operating System interrupts the job, chooses another job from the job pool (waiting queue) as soon as one job goes for an I/O task, Then OS gives CPU to this new job and starts its execution. while this new job does CPU bound tasks, the previous job keeps doing its I/O operation. Let us assume if the second job also goes for an I/O task, the CPU chooses a third job and starts executing it. The CPU is allocated to job as soon as a job completes its I/O operation and comes back for CPU tasks.
- In this way, CPU time can be saved by the system.
- Therefore, to keep the CPU busy as long as there are processes ready to execute is the ultimate goal of multi programming. This way, multiple programs can be executed on a single processor by executing a part of a program at one time, a part of another program after this, then a part of another program and so on, hence executing multiple programs. Hence, the CPU never remains idle.

4.2 Context Switching

To switch a process from one state to another to execute its function using CPUs in the system Operating system uses a technique or method called as The Context switching. It stores the old running process's status in the form of registers when switching perform in the system and assigns the CPU to a new process to execute its tasks. The previous process must wait in a ready queue, while a new process is running in the system. The execution of the old process starts at that point where another process stopped it. To perform multiple tasks without the need for additional processors in the system It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU.

4.2.1 Need for Context switching

To store the system's tasks status and to share a single CPU across all processes to complete its execution A context switching helps. The execution of the process starts at the same point where there is conflicting, When the process reloads in the system.

Following are the reasons that describe the need for context switching in the Operating system.

- The switching of one process to another process is not directly in the system. To use the CPU's resource to accomplish its tasks and store its context, A context switching helps the operating system that switches between the multiple processes. We can resume the service of the process at the same point later. The data which is stored may be lost while switching between processes, If we do not store the currently running process's data or context.
- The currently running process will be shut down or stopped by a high priority process to complete its tasks in the system, If a high priority process falls into the ready queue.
- If any running process requires I/O resources in the system, the current process will be switched by another process to use the CPUs. The old process goes into a ready state to wait for its execution in the CPU, when the I/O requirement is met. Context switching stores the state of the process to resume its tasks in an operating system. Otherwise, the process needs to restart its execution from the initials level.
- The process status is saved as registers using context switching, if any interrupts occur while running a process in the operating system. Once the interrupt gets resolved, to resume its execution at the same

point later the process switches from a wait state to a ready state, where the operating system interrupted occurs.

- Without the need for any additional processors a context switching allows a single CPU to handle multiple process requests simultaneously.

4.2.2 Context switching triggers

There are actually three types of contexts switching triggers which are mentioned and explained as follows.

Interrupts

Multitasking

Kernel/User switch

Interrupts: A CPU requests for the data to read from a disk, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts, if there are any interrupts occur.

Multitasking: A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.

Kernel/User Switch: When switching between the user mode, and the kernel/user mode is performed then it is used in the operating systems.

4.2.3 Steps for Context Switching

Context switching of the processes involves several steps in it. let us consider and assume the context switching of two processes, P1 to P2, to understand the concept of context switching, when an I/O needs, interrupt or priority-based process occurs in the ready queue of PCB is demonstrated in the figure mentioned bellow.

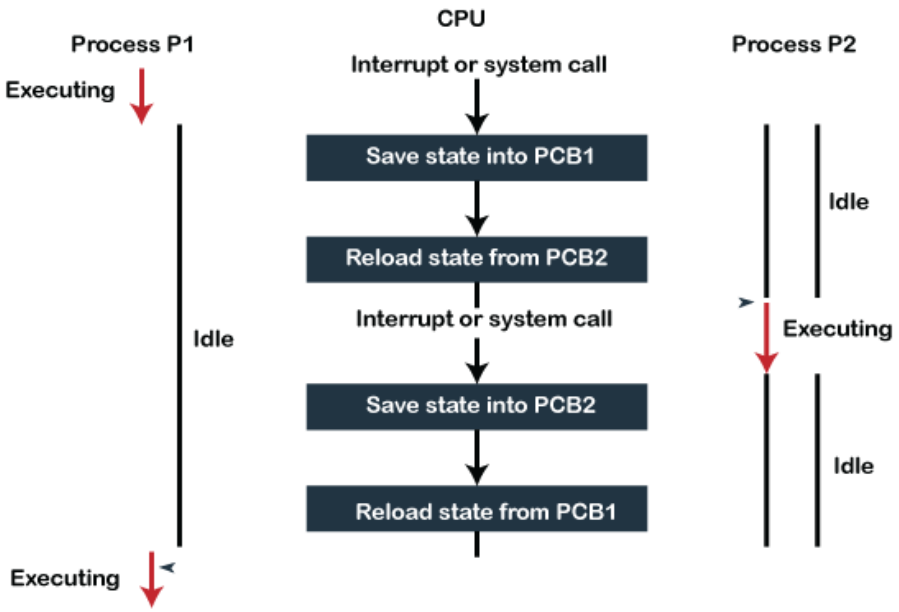


Fig. 4.2 Context Switching Steps

Let us observe the diagram carefully, initial the P1 process is running on the CPU to execute its task, and, another process, P2, is in the ready state at the same time. The P1 process switches its state from running to the waiting state, if an error or interruption has occurred or the process requires input/output. Context switching saves the context of the process P1 in the form of registers and the program counter to the PCB1, before changing the state of the process P1. After that, state of the P2 process is modified from the ready state of the PCB2 to the running state.

4.3 Process Control Block (PCB)

For every process a data structure maintained by the Operating System which is called as A Process Control Block. The PCB is identified by an integer process ID (PID). All the information needed to keep track of a process which is maintained by PCB is as listed below in the table −

Process State: The current state of the process i.e., whether it is ready, running, waiting, or whatever.

Process privileges: This is required to allow/disallow access to system resources.

Process ID: In operating system unique identification for each of the process is maintained called as Process ID.

Pointer: A pointer to parent process.

Program Counter: a pointer to the address of the next instruction to be executed for this process is called as Program Counter.

CPU registers: Various CPU registers where process need to be stored for execution for running state.

CPU Scheduling Information: Process priority and other scheduling information which is required to schedule the process.

Memory management information: This is the information consisting of information about page table, Segment table, memory limits, depending on memory used by the operating system.

Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.

IO status information: This includes a list of I/O devices allocated to the process.

The architecture of a PCB may contain different information in different operating systems and is completely dependent on Operating System. Following figure shows the structure of a PCB –

Fig. 4.3 Structure of Process Control Block

Throughout lifetime of process the PCB is maintained for a process, and is deleted once the process terminates.

4.4 Process Life Cycle

The process is passes through different states, when it executes. In different operating systems, these stages may be different and the names of these states are also not standardized.

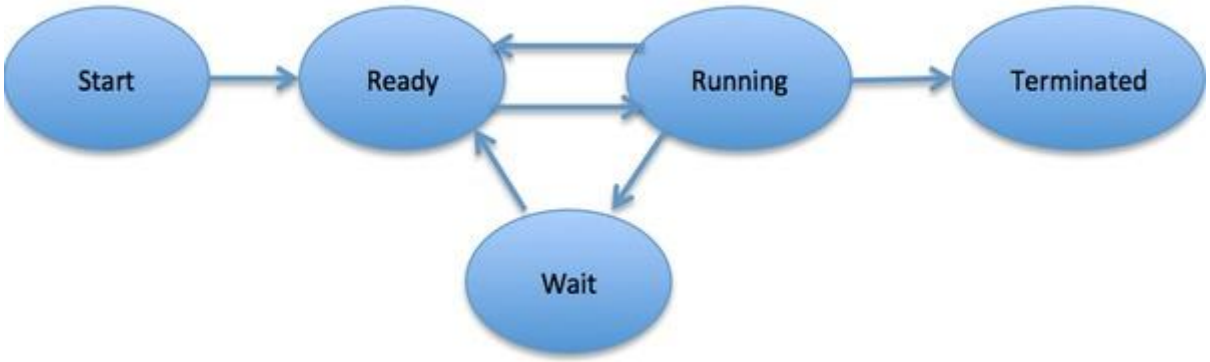


Fig. 4.4 Process Life Cycle

A process can have various state during its life cycle which are explain as follow.

Start: When a process is first started/created, this is the initial state.

Ready: The process which is ready to execute and waiting to be assigned to a processor. Processes those are ready to execute are waiting for the processor to get it 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 but interrupted by the scheduler to assign CPU to some other process.

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.

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.

Terminated or Exit: When the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state. Here it waits to be removed from main memory.

Process Creation: A process can be created for different operations in the system. Some of the events that lead to process creation are:

User requests to create a new process

System initialization

Batch job initialization

Using a system call a running process requires creating another process.

Using the fork() method a process may be created by another process. The process is called as parent process which calls the fork() method or simply, the process creating another process and the created process is called a child process. A parent process can have multiple child processes but a child process can have only one parent. As well the parent and child processes have the same memory, open files, and environment strings, but they have distinct address spaces.

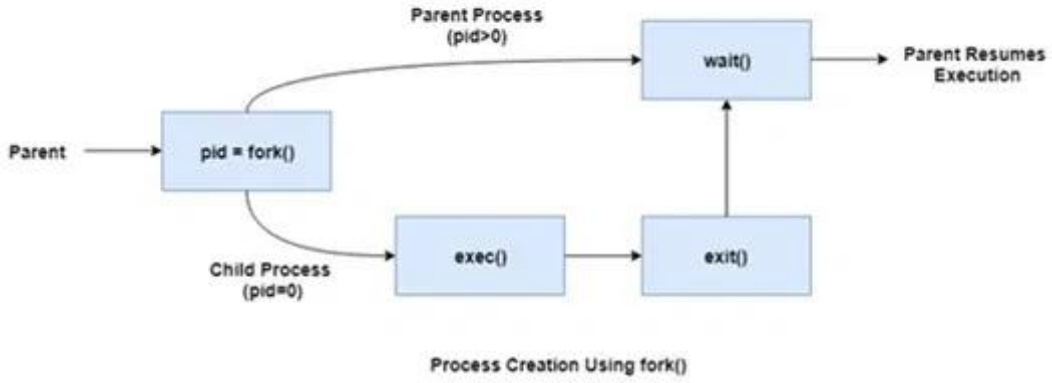


Fig. 4.5 Process Creation

processes may create other processes through appropriate system calls, such as fork or spawn. But still the process which creates other process, is called as the parent of the other process, whereas the sub-process which is newly created called as its child.

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

The process scheduler is termed as sched, and is given PID 0 on a typical UNIX system. At system start-up time the first thing done by it is to launch init, which gives that process PID 1. Further becomes the ultimate parent of all other processes when Init launches all the system daemons and user logins.

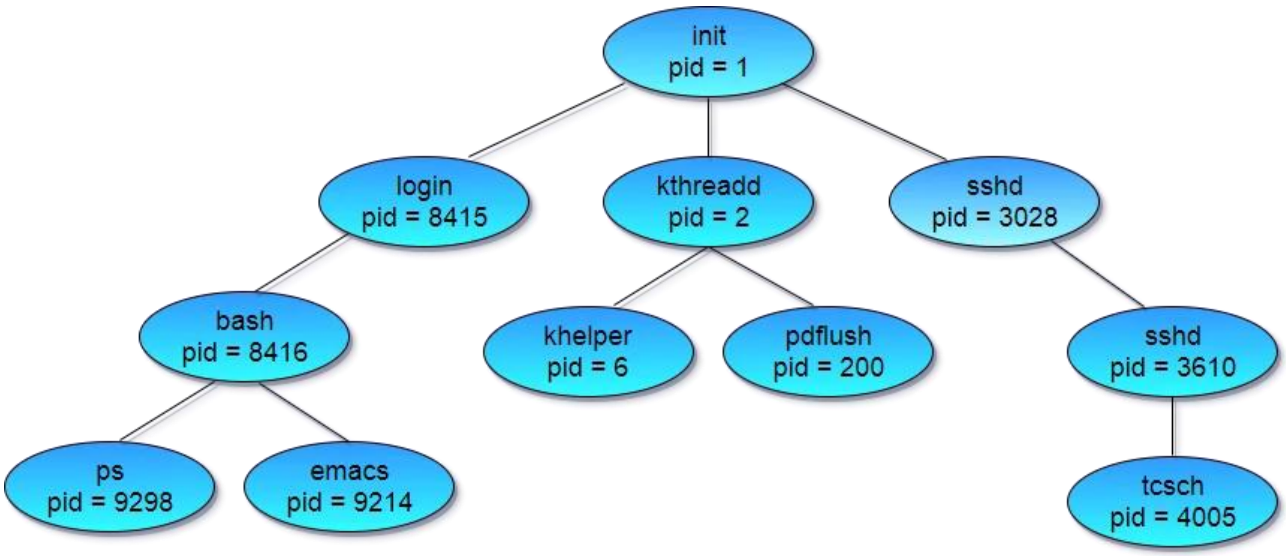


Fig. 4.6 Process Creation Hierarchy

Depending on system implementation a child process may receive some number of shared resources with its parent. Child processes may or may not be limited to a subset of the resources in order to prevent runaway children from consuming all of a certain system resource, which were originally allocated to the parent.

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

- Wait for the child process to terminate before proceeding. Parent process makes a wait() system call, for either a specific child process or for any particular child process, which causes the parent process to block until the wait() returns. UNIX shells normally wait for their children to complete before issuing a new prompt.
- Run concurrently with the child, continuing to process without waiting. When a UNIX shell runs a process as a background task, this is the operation seen. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

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

- The child process is a duplicate of the parent process.
- The child process has a program loaded into it.

To illustrate these different implementations, let us consider the UNIX operating system. In UNIX, each process is identified by its process identifier, which is a unique integer. A new process is created by the fork system call. The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes (the parent and the child) continue execution at the instruction after the fork system call, with one difference: The return code for the fork system call is zero for the new(child) process, whereas the(non zero) process identifier of the child is returned to the parent.

4.5 Process Termination

Process termination occurs when a process is forced to stops its execution by just terminating it. The system function exit() is used to terminate a process. There might be number of reasons which result in

terminating the process. The two different categories are, voluntary and involuntary. Some of the causes of process termination are:

- A process may be terminated naturally after its execution releasing all of its resources.
- A child process may be terminated if its parent process requests for the same.
- A process can be terminated if it tries to use a resource that it is not allowed to.
- Another cause of a process termination can be an I/O failure for a process.
- If a parent process is terminated, then it will be another cause for its child process's termination.
- A process can also be terminated if it requires more memory than what is currently available in the system.

Why do processes terminate?

- Most processes terminate because they have done their work. Processes after completing their work, perform a system call to tell the Operating System that they have completed the work. In UNIX, this system call is 'exit'. In Windows, this goes by 'EndProcess'. The user has a clickable icon to exit the process for the foreground processes. It's usually denoted by the x symbol for windows or the red button in macOS.
- There may be some instances where the processes exit involuntarily. This may be due to some fatal error. For example, if you command the compiler to compile a file called red.c but no such file exists, the compiler will terminate.
- Sometimes the process causes an error due to which it has to terminate. There are various types of errors may occur are mentioned as follow, but are not limited to:
  – Referencing non-existent memory

  – Divide by zero error

  – Executing an illegal instruction

- In some cases, the process signals the operating system that it wishes to handle the error itself using an interrupt.
- There are some processes that instruct the OS to kill other processes. In UNIX, this command is straightforward – 'kill'. Anyway, the killer must have the necessary authorization to kill the other process.

4.6 Process Hierarchy

We require to run many processes at a time in a computer system and while their execution some processes need to create other processes. The parent and the child processes tend to associate with each other in certain ways, when a process creates another process. The child process can also create other processes if required. This parent-child like structure of processes form a hierarchy, called Process Hierarchy.

Unlike sexual reproduction though, the child processes have only one parent. A process can be a parent by itself and that is usually the case.

In UNIX, a process group is formed by process and all of its children and grandchildren. The signal is delivered to all members of the process group currently associated with the keyboard when a user sends a signal from the keyboard. Each process can catch the signal Individually and do as it wish.

There is no hierarchy system in Windows. Each process is treated equally. A parent process gets a handle and that is only the only remote association with the hierarchy that it does. But even then, here some other process may get control from this parent process and thereby invalidating the hierarchy system.

4.6.1 Two-State Process Model

A Two-State Process Model classifies a process in two different types as mentioned bellow :

- A process running on CPU
- A process not running on CPU

When the CPU runs the process, it is in the running state but not when a process is created, at that time it is in the not in running state. Remember the newly created process will be run by the CPU If there is a process in a running state and another process is created with a higher priority. The former process will go in the not running state.

4.6.2 Five-State Process Model

The Five-State Process Model categorizes a process in five states:

- New: When a process is newly created, it is said to be in the new state.
- Ready: All those processes which are loaded into the primary memory and are waiting for the CPU are said to be in ready state.
- Running: All the processes which are running are said to be in the running state.
- Waiting: All the processes, which leave the CPU because of any reason (I/O or for any high priority process) and wait for their execution, are in waiting state.
- Terminated: A process that exits or terminates from the CPU and the primary memory is said to be in the terminated state.

4.7 Process Pre-emption

In pre-emption an interrupt mechanism is used that suspends the process which is executing currently and the next process to execute is determined by the short-term scheduler. All processes must get some CPU time for execution is the responsibility taken by Pre-emption.

A diagram that demonstrates process pre-emption is as follows −

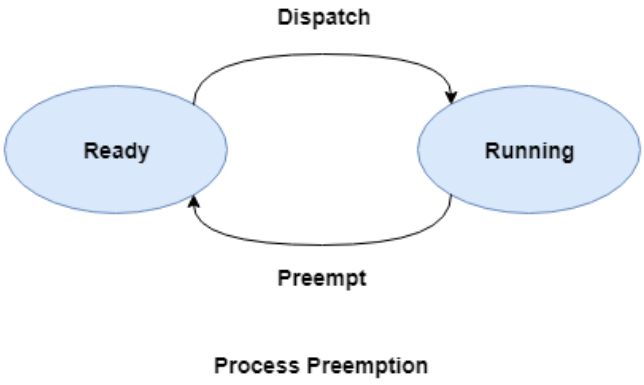


Fig. 4.7 Process Pre-emption

4.7 Process Blocking

If the process is waiting for some event to occur then the process is got blocked. As the I/O events are executed in the main memory and don't require the processor and hence this event may be I/O. After the event is complete, the process again goes to the ready state.

Let us understand the concept with following diagram that demonstrates process blocking is as follows −

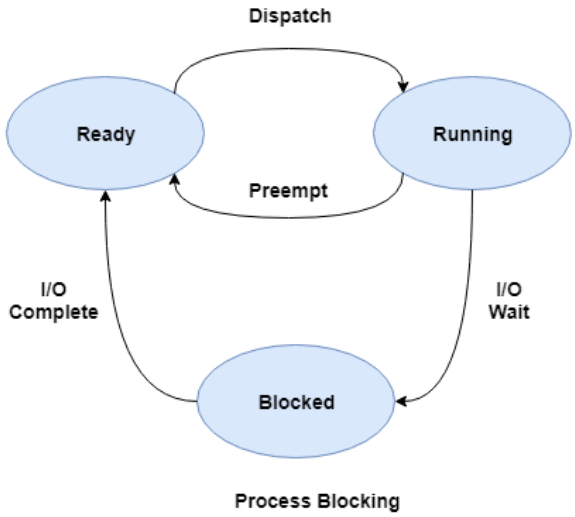


Fig. 4.8 Process Blocking

4.8 Process Priorities

As you likely know, UNIX and Linux operating systems are multiuser environments. You will also have a large number of processes running on the system, If you have a large number of users on the system. there are also many processes needed to keep the system running smoothly, and to provide system services to users, in addition to all of the user processes running on a system.

If these system processes do not receive adequate CPU time, the result could be major system problems. To ensure that these key processes get the CPU time they need, their assigned execution priority numbers are higher than those of the user processes.

A process's execution priority number is used by the operating system to schedule CPU time for the process. As implied in the previous paragraph, processes with higher execution priority numbers are given more CPU time than those with lower execution priority numbers. The operating system (kernel) automatically and continually computes and updates each process's execution priority number based on several different factors.

Although the operating system ultimately controls a process's execution priority number, there is a way for a process owner to reduce the priority of his or her processes. This may be accomplished by changing the nice number of their process.

A process's nice number is the process's priority number with respect to other processes. A process with a higher nice number, or higher level of niceness, will yield CPU time to other processes on the system. A process with a lower nice number will have a lower level of niceness relative to other processes.

The nice number of a process can be increased by the owner of the process, and increased or decreased by the superuser/root. The nice command is used when a command is started to alter the command's nice number from its default.

4.9 Process Scheduling

Process Scheduling is an OS task that schedules processes of different states like ready, waiting, and running.

Process scheduling allows OS to allocate a time interval of CPU execution for each process. Another important reason for using a process scheduling system is that it keeps the CPU busy all the time. This allows you to get the minimum response time for programs.

4.9.1 Process Scheduling Queues

Process Scheduling Queues help you to maintain a distinct queue for each and every process states and PCBs. All the process of the same execution state are placed in the same queue. Therefore, whenever the state of a process is modified, its PCB needs to be unlinked from its existing queue, which moves back to the new state queue.

Three types of operating system queues are:

Job queue – It helps you to store all the processes in the system.

Ready queue – This type of queue helps you to set every process residing in the main memory, which is ready and waiting to execute.

Device queues – It is a process that is blocked because of the absence of an I/O device.

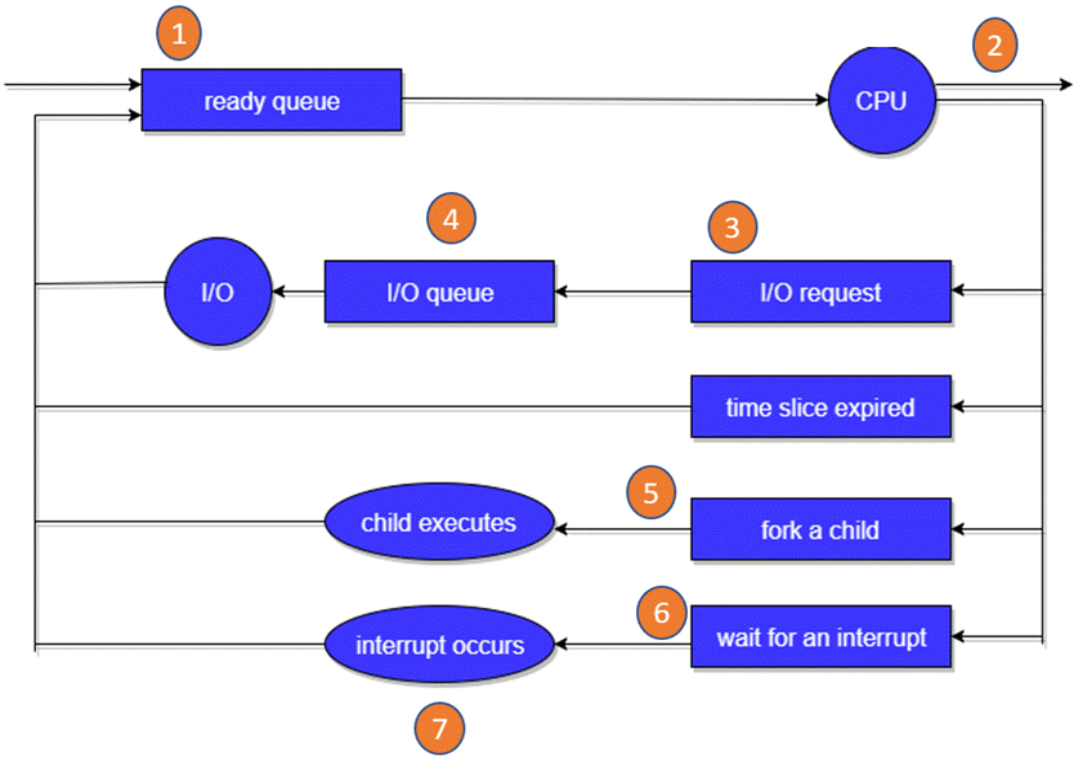


Fig. 4.9 Process Scheduling Queues

In the above-given Diagram,

- Rectangle represents a queue.
- Circle denotes the resource
- Arrow indicates the flow of the process.

Every new process first put in the Ready queue .It waits in the ready queue until it is finally processed for execution. Here, the new process is put in the ready queue and wait until it is selected for execution or it is dispatched.

- One of the processes is allocated the CPU and it is executing
- The process should issue an I/O request
- Then, it should be placed in the I/O queue.
- The process should create a new subprocess
- The process should be waiting for its termination.
- It should remove forcefully from the CPU, as a result interrupt. Once interrupt is completed, it should be sent back to ready queue.

Two State Process Model

Two-state process models are:

- Running State
- Not Running State

Running: In the Operating system, whenever a new process is built, it is entered into the system, which should be running.

Not Running: The process that are not running are kept in a queue, which is waiting for their turn to execute. Each entry in the queue is a point to a specific process.

Scheduling Objectives

Here, are important objectives of Process scheduling

- Maximize the number of interactive users within acceptable response times.
- Achieve a balance between response and utilization.
- Avoid indefinite postponement and enforce priorities.
- It also should give reference to the processes holding the key resources.

4.9.1 Type of Process Schedulers

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

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

4.9.1.1 Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long-term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes, then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long-term scheduler is very critical and may affect the system for a very long time.

4.9.1.2 Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short-term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short-term scheduler makes some mistakes while selecting the job.

4.9.1.3 Medium term scheduler

Medium term scheduler takes care of the swapped-out processes. If the running state processes needs some IO time for the completion, then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped-out processes and this procedure is called swapping. The medium-term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

Comparison for different types of schedulers

Table 4.1 Comparison for different types of schedulers

S.No.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1.	A job scheduler	A CPU scheduler	A process swapping scheduler
2.	Slowest speed	Fastest Speed	Speed is between the other two
3.	Controls the degree of multiprogramming	Provides less control over the degree of multiprogramming	Reduces the degree of multiprogramming

4.	Absent or minimal in the time-sharing OS	Minimal in time-sharing OS	Part of time-sharing OS
5.	Selects a process from pool and loads it into memory for execution	Selects a process that is ready for execution	Re-introduces processes into memory for continued execution

4.10 Various Times related to the Process

1. Arrival Time: The time at which the process enters into the ready queue is called the arrival time.

2. Burst Time: The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

3. Completion Time: The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

4. Turnaround time: The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

5. Waiting Time: The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

6. Response Time: The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

4.11 CPU Scheduling

In the uni-programming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

In Multiprogramming systems, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called CPU scheduling. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short-term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short-term scheduler saves the current context of the process (also called PCB) and changes its state from running to waiting. During the time, process is in waiting state; the Short-term scheduler picks another process from the ready queue and assigns the CPU to this process. This procedure is called context switching.

What is saved in the Process Control Block?

The Operating system maintains a process control block during the lifetime of the process. The Process control block is deleted when the process is terminated or killed. There is the following information which is saved in the process control block and is changing with the state of the process.

Why do we need Scheduling?

In Multiprogramming, if the long-term scheduler picks more I/O bound processes then most of the time, the CPU remains idol. The task of Operating system is to optimize the utilization of resources.

If most of the running processes change their state from running to waiting then there may always be a possibility of deadlock in the system. Hence to reduce this overhead, the OS needs to schedule the jobs to get the optimal utilization of CPU and to avoid the possibility to deadlock.

4.11 Time Slice

CPUs kernel doesn't simply distribute the entirety of our PCs' resources to single process or service. CPU is continuously running many processes that are essential for it to operate, so our kernel needs to manage these processes without moment's delay.

When program needs to run, process must be created for it. This process needs to have important resources like RAM and CPU. The kernel schedules time periods for CPU to perform commands and instructions in process. Be that as it may, there's just single CPU and numerous processes.

How does CPU outstand to execute different processes without moment's delay? It does it by executing processes one by one, individually by time slice. A time slice is short time frame that gets assigned to process for CPU execution.
It is timeframe for which process is allotted to run in pre-emptive multitasking CPU. The scheduler runs each process every single time-slice. The period of each time slice can be very significant and crucial to balance CPUs performance and responsiveness.

If time slice is quite short, scheduler will take more processing time. In contrast, if the time slice is too long, scheduler will again take more processing time.

When process is allotted to CPU, clock timer is set corresponding to time slice.

- If the process finishes its burst before time slice, CPU simply swaps it out like conventional FCFS calculation.
- If the time slice goes off first, CPU shifts it out to back of ongoing queue.

The ongoing queue is managed like circular queue, so, after all processes are executed once, scheduler executes first process again and then second and so forth.

Advantages :

- Fair allocation of CPU resources.
- It deals all process with equal priority.
- Easily implementable on the system.
- Context switching method used to save states of pre-empted processes
- gives best performance in terms of average processing time.

Disadvantages :

- If the slicing time is short, processor output will be delayed.

- It spends time on context switching.
- Performance depends heavily on time quantum.
- Priorities can’t be fixed for processes.
- No priority to more important tasks.
- Finding an appropriate time quantum is quite difficult.

4.12 Scheduling Algorithms

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

- Maximum CPU utilization
- Fare allocation of CPU
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time

There are the following algorithms which can be used to schedule the jobs.

1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-pre-emptive type of scheduling.

2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a pre-emptive type of scheduling.

3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-pre-emptive type of scheduling.

4. Shortest remaining time first

It is the pre-emptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

4.12.1 FCFS Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serve

Disadvantages of FCFS

- The scheduling method is non-pre-emptive, the process will run to the completion.
- Due to the non-pre-emptive nature of the algorithm, the problem of starvation may occur.

Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID P0, P1, P2, P3 and P4. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

Table 4.2 Processes With Arrival time and waiting time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	10	6
3	3	9	21	18	9
4	6	12	33	29	17

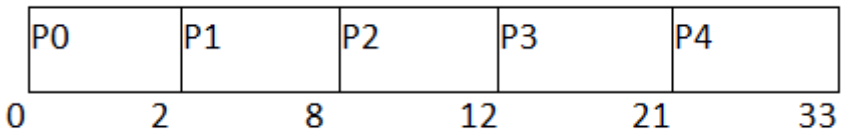


Fig. 4.10 Gantt Chart of FCFS

4.12.2 Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

- Maximum throughput
- Minimum average waiting and turnaround time

Disadvantages of SJF

- May suffer with the problem of starvation
- It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined.

Example: In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

Table 4.3 Processes with Arrival time and waiting time

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

Since, No Process arrives at time 0 hence; there will be an empty slot in the Gantt chart from time 0 to 1 (the time at which the first process arrives).

According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.
Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

So that's how the procedure will go on in shortest job first (SJF) scheduling algorithm.

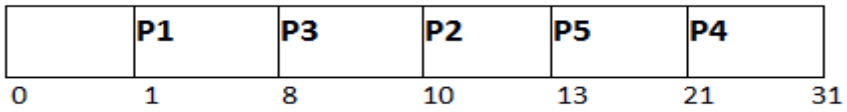


Fig. 4.11 Gantt Chart of SJF

4.12.3 Shortest Remaining Time First (SRTF) Scheduling Algorithm

This Algorithm is the pre-emptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short-term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the ready queue, No pre-emption will be done and the algorithm will work as SJF scheduling. The context of the process is saved in the Process Control Block when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the next execution of this process.

Example: In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table.

Table 4.4 Processes and their arrival time and burst time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0
2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5

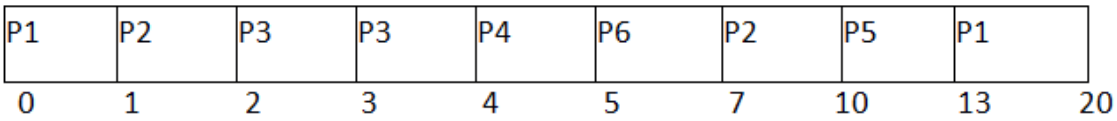


Fig. 4.12

Gantt Chart of SRTF

Avg Waiting Time = 24/6

The Gantt chart is prepared according to the arrival and burst time given in the table.

Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.

The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a pre-emptive one, the current execution is stopped and the scheduler checks for the process with the least burst time.
Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.

The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.

The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively.

P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again.

The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.

The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

Once all the processes arrive, No pre-emption is done and the algorithm will work as SJF.

4.12.4 Round Robin Scheduling Algorithm

Round Robin scheduling algorithm is one of the most popular scheduling algorithms which can actually be implemented in most of the operating systems. This is the pre-emptive version of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time, then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

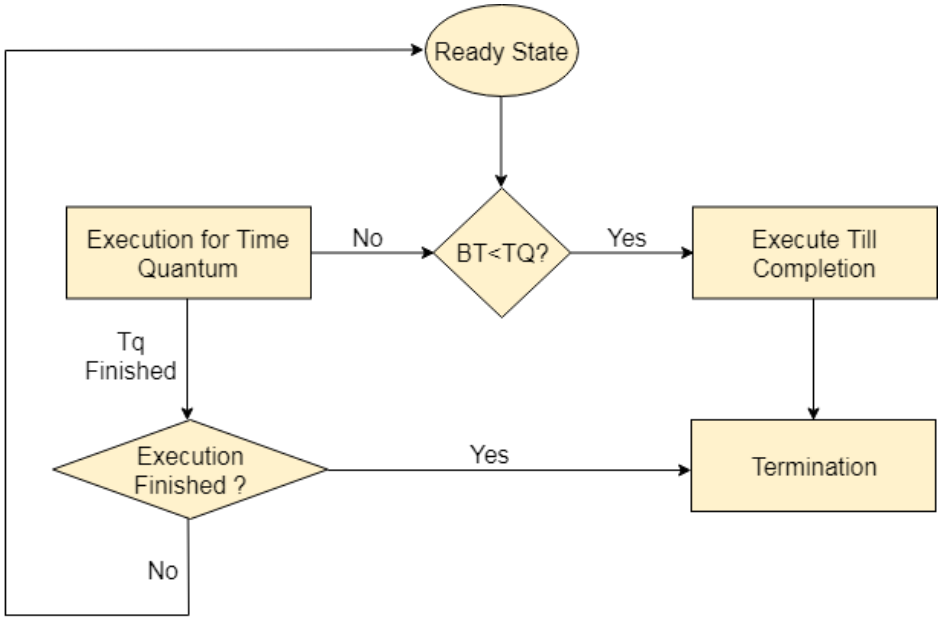


Fig. 4.13 Round Robin Scheduling

Advantages

- It can be actually implementable in the system because it is not depending on the burst time.
- It doesn't suffer from the problem of starvation or convoy effect.
- All the jobs get a fare allocation of CPU.

Disadvantages

- The higher the time quantum, the higher the response time in the system.
- The lower the time quantum, the higher the context switching overhead in the system.
- Deciding a perfect time quantum is really a very difficult task in the system.

In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

Process ID	Arrival Time	Burst Time
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4

According to the algorithm, we have to maintain the ready queue and the Gantt chart. The structure of both the data structures will be changed after every scheduling.

4.12.5 Priority Scheduling

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One is Pre-emptive priority scheduling while the other is Non-Pre-emptive Priority scheduling.

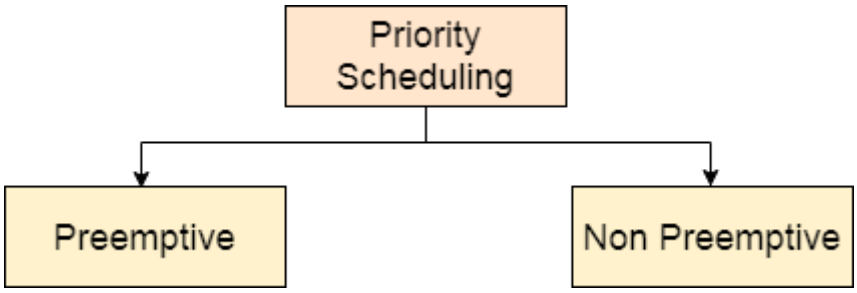


Fig. 4.14 Priority Scheduling Types-1

The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called static priority, while if it keeps changing itself at the regular intervals, it is called dynamic priority.

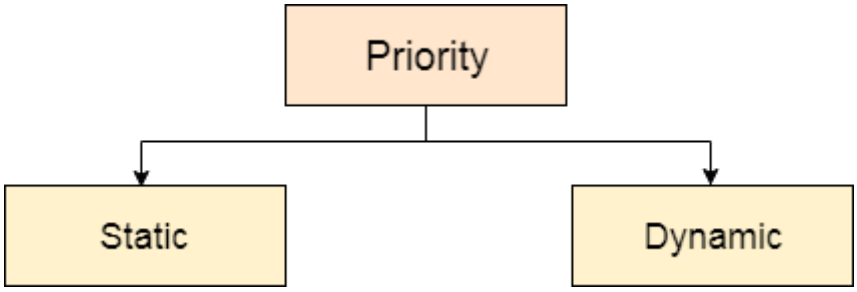


Fig. 4.15 Priority Scheduling Types-2

4.13 What is 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 few information like code segment, data segment and open files. When one thread alters a code segment memory item, 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 have 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.

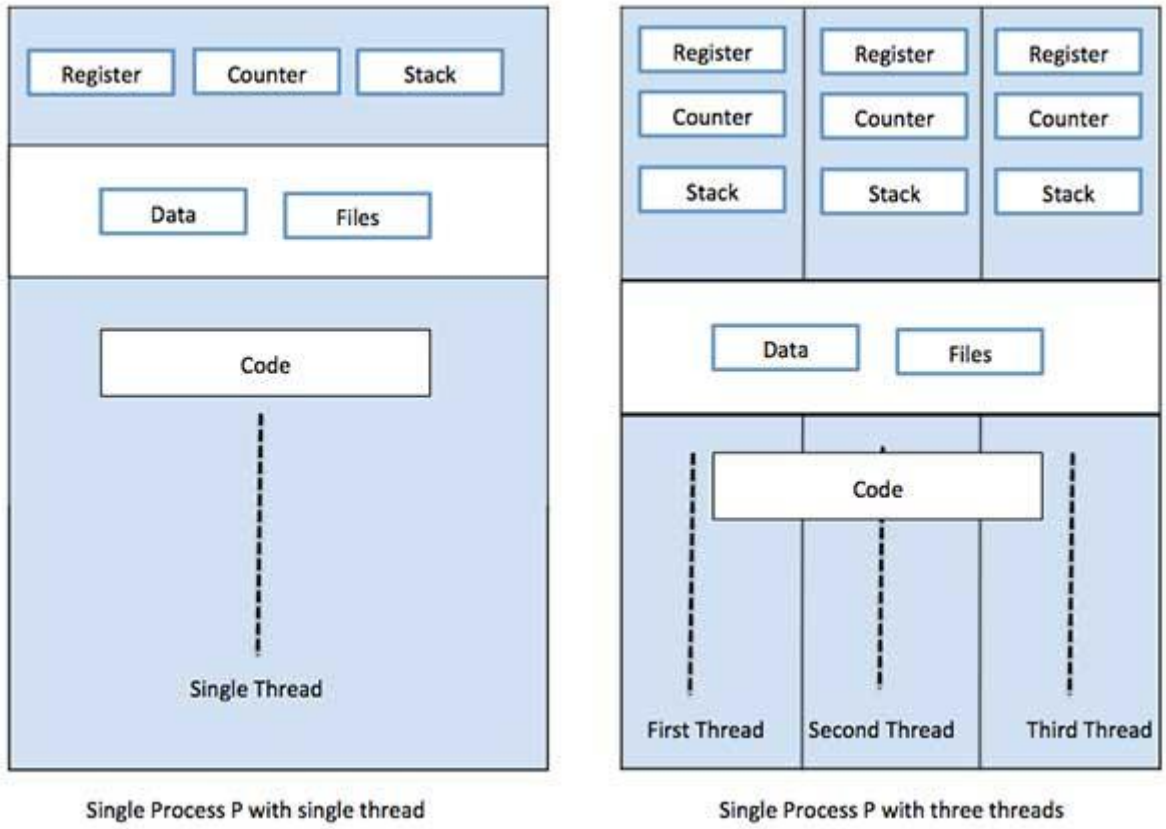


Fig. 4.16 Multithreading

4.13.1 Difference between Process and Thread

Table 4.5 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.

4.13.2 Types of Thread

Threads are 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.

4.13.2.1 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.

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.

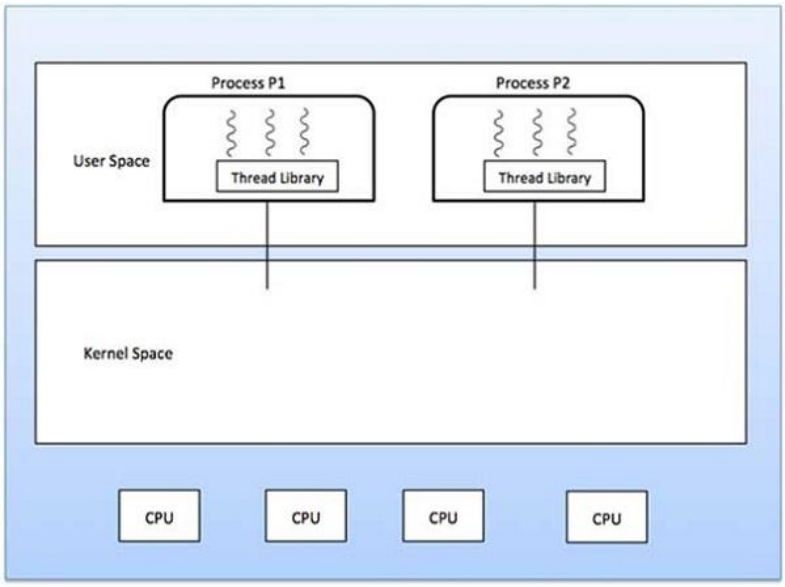


Fig. 4.17 User Level Threads

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

4.13.2.2 Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are 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 individuals' threads within the process. Scheduling by the Kernel is done on a thread basis. 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.

4.13.3 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 relationships.
- Many to one relationship.
- One to one relationship.

4.13.3.1 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.

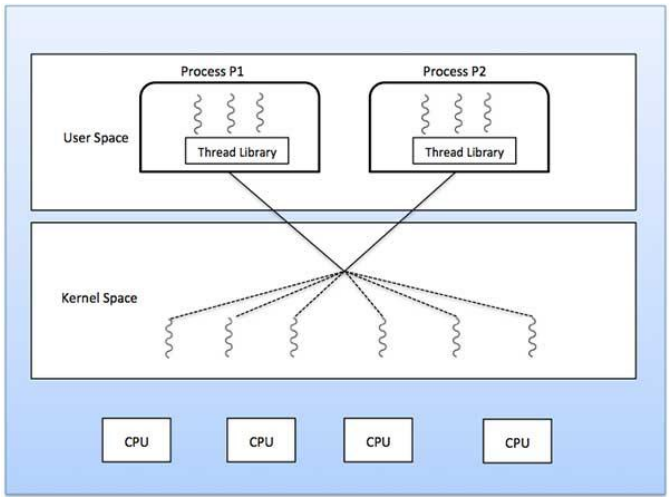


Fig. 4.18 Many to many model

4.13.3.2 Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. 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 are 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.

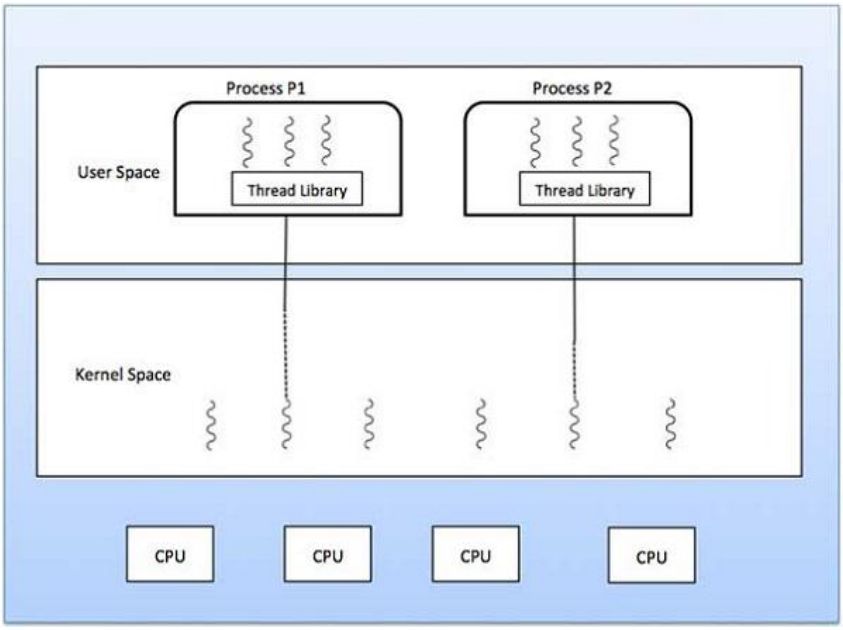


Fig. 4.19 Many to one model

4.13.3.3 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.

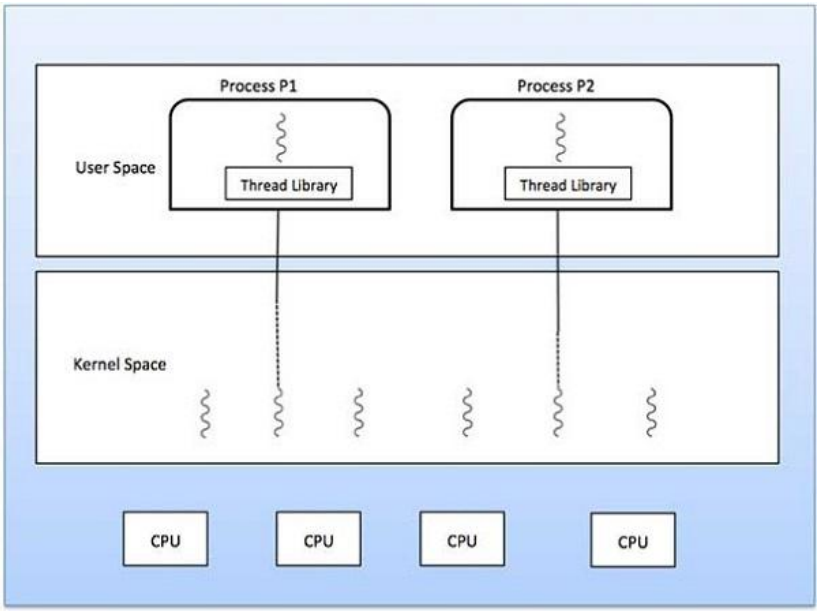


Fig. 4.20 One to one model

4.13.4 Difference between User-Level & Kernel-Level Thread

Table 4.6 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.

Exercise Questions

1. What is Process? Explain in short.
2. Describe the concept of context switching.
3. Describe the Steps for Context Switching.
4. Write short note on multi programming.
5. Write detailed note on Process Control Block
6. Explain with neat diagram Process life cycle.
7. Explain various Multithreading Models.
8. Differentiate between User-Level & Kernel-Level Thread
9. Differentiate between Process and Thread.
10. Explain Shortest Job First (SJF) Scheduling.
11. Write a note on scheduling algorithms.
12. Explain concept of CPU scheduling in detail.
13. Write a short note on thread.
14. Describe Process Priorities in detail.

References:

Books:

- Operating System Design Principles, William Stallings, Prentice Hall of India
- Operating System, Achyut Godbole, Tata McGraw Hill

Web resources

- https://www.javatpoint.com
- https://www.tutorialspoint.com
- https://geeksforgeeks.org
- https://www.guru99.com
- https://www.studytonight.com
- https://technobyte.org

Unit 5 Inter Process Communication

Objectives:

- To study the Producer-Consumer Problems.
- To study the concept of mutual exclusion.

5.1 The Producer-Consumer Problems

A classic problem which is used for multi-process synchronization i.e., synchronization between more than one processes is the Producer-Consumer problem. In this problem there is one Consumer that is consuming the products produced by the Producer as well there is one Producer that is producing something. The same memory buffer that is of fixed-size is shared by the producers and consumers.

The Producer is responsible to generate the data, which is then placed into the buffer, and again start generating data, while the Consumer is responsible to consume the data from the buffer.

5.1.1 What's the problem here?

Some of the problems that might occur in the Producer-Consumer are as mentioned below:

- When the buffer is not full, only then the producer should produce data. The producer shouldn't be allowed to put any data into the buffer when the buffer is full.
- When the buffer is not empty, the consumer is not allowed to consume data. The consumer shouldn't be allowed to take any data from the buffer, if the buffer is empty.
- The buffer should not be allowed to access at the same time by the producer and consumer.

5.1.2 What's the solution?

The semaphores can be used to solve the above mentioned three problems. To solve this let us take three semaphore variables in the producer-consumer problem:

1. Semaphore S: To achieve mutual exclusion between processes, this semaphore variable is used. This variable will allow either Producer or Consumer to use or access the shared buffer at a particular time. This variable is set to 1 initially.
2. Semaphore E: To define the empty space in the buffer, this semaphore variable is used. Initially, it is set to the whole space of the buffer i.e., "n" because the buffer is initially empty.
3. Semaphore F: To define the space that is filled by the producer, this semaphore variable is used. As initially there is no space filled by the producer, so initially it is set to "0".

The above problem can be solved by using the above three semaphore variables and by using the *wait()* and *signal()* function. The *signal()* function increases the semaphore variable by 1 whereas the *wait()* function decreases the semaphore variable by 1. So, let's see how.

The following is the pseudo-code for the producer:

```
void producer() {
    while(T) {
        produce()
        wait(E)
        wait(S)
        append()
        signal(S)
        signal(F)
    }
}
```

The above code can be summarized as:

- *while()* is used to produce data, again and again, if it wishes to produce, again and again.
- *produce()*to produce data by the producer this function is called.
- *wait(E)* When the producer produces something then there is a decrease in the value of the empty space in the buffer so will reduce the value of the semaphore variable "E" by one. The program will stop its execution and no production will be done, when the buffer is full i.e., the value of the semaphore variable "E" is "0".
- *wait(S)* Not to allow other process can enter into the critical section, this method is used to set the semaphore variable "S" to "0".
- *append(): T*o append the newly produced data in the buffer, this function is used.
- *signal(s)* To allow to other processes to come into the critical section this method is used to set the semaphore variable "S" to "1". Which is because the production is done and the append operation is also done.
- *signal(F)* As after adding the data into the buffer, one space is filled in the buffer and the variable "F" must be updated, so this method is used to increase the semaphore variable "F" by one.

This is what the solution for the produce part of the producer-consumer problem. Now, let's see the consumer solution. The following is the code for the consumer:

```
void consumer() {
    while(T) {
        wait(F)
        wait(S)
        take()
        signal(S)
        signal(E)
        use()
    }
}
```

The above code can be summarized as:

- *while()* is used to consume data, again and again, if it wishes to consume, again and again.
- *wait(F)* if some data is consumed by the consumer, then the variable "F" must be decreased by one, then this method is used to decrease the semaphore variable "F" by one.
- *wait(S)* : No other process can enter into the critical section, when the semaphore variable is set to "S" to "0" using this function.
- *take()* function is used to take the data from the buffer by the consumer.
- *signal(S)* is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the consumption is done and the take operation is also done.
- *signal(E)* As after taking the data from the buffer, one space is freed from the buffer and the variable "E" must be increased so this method is used to increase the semaphore variable "E" by one.
- use() : To use the data taken from the buffer by the process to do some operation, for which this function is used.

5.1.3 Race condition

In General, a common storage area, such as memory or files on a hard disk may be shared by different processes of operating systems, where they are allowed to read and write. To coordinate the desired operations in the right way between processes that use common areas is the important responsibility of Operating systems. These processes need to communicate with each other, have a lot of business, in order to ensure that the action of one process will not affect the normal action of another process, and then lead to the process cannot get the desired results after running. To represent the communication between multiple processes the term IPC (Inter Process Communication) is usually used in the concept of operating system.

Let us introduce a simple example to illustrate and to explain what race condition is: A file contains a number n, process A and process B want to read the number of this file, and add this number to 1, save back to the file. Assuming that the initial value of n is 0, in our ideal case, after process A and process B run, the value of N in the file should be 2, but in fact, it may occur that the value of n is 1. We can consider what steps each process needs to take to do this:

- Read the value of N in the file
- Let n = n + 1
- Save the new n value back to the file.

It is necessary to review several knowledge points in the concept of operating system, before further explaining race conditions:

- Even when there is only one CPU), processes can run concurrently.
- A rescheduling of the process can be cause, Clock interruption in the operating system.
- A rescheduling of process runs can also cause by interrupts from other devices, in addition to clock interrupts.

Assuming that a clock interrupt occurs when process A runs steps 1 and 2, but it has not started to run step 3, the operating system makes process B start to run by scheduling., It finds that the value of n is 0, so it runs steps 2 and 3, and finally saves n = 1 to a file, it happens when process B runs step 1. As it does not know that process B has changed the value in the file before it runs step 3, so process A also writes n = 1 back to the file, so after that, process A continues to run. That's the problem. There are other processes to manipulate the data it operates on, in the process of running process A.

The only way to make n = 2 is to expect process A and process B In order Complete all steps separately.

A definition of race conditions can be given as follow.

> When the final result depends on the exact timing of the process and two or more processes read and write some shared data, called race conditions.

To discuss race conditions, we use processes as objects as mentioned in the above text, which is also applicable to threads, where threads include but are not limited to kernel threads and user threads. As, to run programs processes actually rely on threads in an operating system. The concept of race conditions can also be useful in the Java language for thread security.

5.2 Mutual Exclusion and Critical Zone

The race condition can be avoided, by ensuring that when a process uses a shared variable or file, other processes cannot do the same thing. In simple word you can say, we need to "mutually exclusive".

Let understand the concept with an example, the program fragment in step 1-3 has to define as a critical area, which means that the area is sensitive, because once the process runs to this area, it means that the common data area or file will be operated on, and it means that other processes may also be running to the critical area. The race conditions can be avoided If the two processes cannot be in the critical zone at the same time in an appropriate way.

Which means that, one has to think about how to "mutually exclusive".

Mutual Exclusive Scheme

To prevent multiple processes from entering the critical region at the same time, this what the essence of mutual exclusion is.

Shielding interruption

In the example discussed previously, as process A was interrupted in the critical zone, process B was able to enter the critical zone. If to shield all interrupts is allowed to process A immediately after entering the critical area and respond to interrupts only after leaving the critical area, then even if interrupts occur, the CPU will not switch to other processes, so process A can safely modify the file content at this time, without worrying that other processes will interfere with its work.

Though, the idea is good, but in fact it is not feasible. The process A cannot shield interruption to other CPUs, when there are more than one CPU, it can only shield the CPU that is dispatching it, so the process dispatched by other CPUs can still enter the critical area; second, on the issue of power, can the power of shielding interruption be given to user processes? If no longer responds to the interrupt and if the process masks the interrupt, then a process may hang up the entire operating system.

Lock variables

Possibly, when a process wants to enter the critical zone by setting a lock flag bit and setting its initial value to 0, it first checks whether the lock value is 0, if 0, then it is set to 1, then it enters the critical zone, and after exiting the critical zone, it changes the lock value to 0; if the lock value is already 1 when checking, it means that other processes are already in the critical zone, so the process follows. The loop is responsible to waits and to constantly detects the value of the lock until it becomes zero.

Though in this way, the race conditions occur, because when a process reads out a lock value of 0, before it sets its value to 1, another process is scheduled, and it reads the lock value of 0, so that both processes are in the critical zone at the same time.

Strict Alternation

The action of changing the lock variable from 0 to 1 is performed by the process that wants to acquire the lock, this is what the problem with lock variables is. There is no race condition, if we change this action to a process that has acquired a lock to execute.

To represent who is currently allowed to get the lock, the set a variable turned. Let us assume that there are two processes and let us understand the logic of process A is as follows:

```
While (turn!= 0) {// If it's not your turn to acquire the lock, enter the loop and wait
        }
        Do_critical_region();//Procedures for executing critical zones
        Turn = 1; // The lock variable is modified to other values by the party that acquires the lock, allowing other processes to acquire the lock
        Do_non_critical_region();//Procedures to execute non-critical zones
```

The logic of process B is as follows:

```
When (turn!= 1) {// If it's not your turn to acquire the lock, enter the loop and wait
        }
        Do_critical_region();//Procedures for executing critical zones
        Turn = 0; // / The lock variable is modified to other values by the party that acquires the lock, allowing other processes to acquire the lock
```

```
        Do_non_critical_region();//Procedures to execute non-critical
zones
```

But here we need to consider one thing. Let us consider that for process A the do_non_critical_region() required to be executed for a long time, that is, the logic of the non-critical zone of process A needs to be executed for a long time, and the logic of the non-critical zone of process B is executed quickly. Obviously, the process A who is entering the critical zone will have a frequency that will be a little less than that of process B. Ideally, process B should be more approaching. Boundary area several times. But before executing the logic of non-critical region the process B sets turn to 0, it is found and observed that the value of turn is not always 1when it quickly finishes executing the logic of non-critical region and checks the value of turn. While process A is carrying on a long logic code of non-critical region at this time, the value of turn needs process A to set it to 1, sot that process B cannot enter. Enter the critical zone.

This prove that when one process is much slower than another, strict rotation is not a good method.

5.2.1 Peterson algorithm

The strict rotation method problem lies in words, which is, multiple processes enter the critical region in turn. The process of obtaining locks depends on the modification of lock variables by other processes this is what the fundamental reason of problem, while other processes need to be executed by non-critical region logic before they can modify lock variables.

The turn variable in the strict rotation method is really popular good method for two reasons. It indicates who is currently in the turn to acquire the lock, but also used to prevents other processes from entering the critical zone before its value is changed. It is sure that after passing through the logic of the non-critical zone a process always changes the turn value.

Hence to overcome these two variables to express can be used, one to indicate that the current process has left the critical zone and other to indicate who should get the lock at present. This method is actually called Peterson's algorithm, which was proposed by T. Dekker, a Dutch mathematician.

```
    static final int N = 2;
    int turn = 0;
    boolean[] interested = new boolean[N];

    void enter_region(int process) {
        int other = 1 - process;
        interested[process] = true;
        turn = process;
        while(turn == process && !interested[other]) {
        }
    }
```

```
void exit_region(int process) {
    interested[process] = false;
}
```

a process needs to calls enter_region first when it wants to enter the critical region, and after leaving the critical region, it calls exit_region. Just immediately after leaving the critical area Peterson algorithm makes the process eliminate its “interest” in the critical area, so other processes can judge whether they can legally enter the critical area according to the value of turn.

TSL Directive

The method Looking back at the “lock variable” has one of its fatal drawbacks is that when state variables are changed a race condition occurs. The state variable changes from 0 to 1 or from 1 to 0, they can be interrupted, so there are

This can conclude on the basis of locking variables that the process that wants to get into the critical region is not doing the modification of locking variables, but the modification is done by the process which is entered in critical region and that wants to leave the critical region, the race condition can be avoided, and then the strict rotation method and the improved Peterson algorithm based on the strict rotation method are introduced. In the way of software these methods are all considered. Actually, the change of lock variables can be guaranteed without interruption which is with the support of hardware CPU, it makes lock variables a good solution to process mutual exclusion.

The TSL instructions which are called Test and Set Lock at present are supported by most computer CPUs. It stores a non-zero value on the memory address when it reads a variable of memory (word) into the register RX. From the hardware level reading and writing operations are guaranteed to be non-interruptible, that is to say, atomic. While executing TSL instructions and prohibiting other CPUs from accessing memory before the end of TSL instructions, it uses the method of locking the memory bus. Which is also called as CAS (Compare and Set).

When it is necessary to change the lock variable from 0 to 1, first copy the value of memory to the register, and set the value of memory to 1, then the value of the register has to be checked if it is 0, if it is 0, then the operation is successful, if not 0, then repeat detection, knowing that the value of the register has changed to 0, if the value of the register has changed to 0, it means that another process has left the critical area. It is necessary to set the value of the variable in memory to 0, when a process leaves the critical region.

Be busy waiting

The Peterson algorithm and TSL method have one similar or common characteristic: they are busy waiting when they are waiting to enter the critical region, which we often call spin. It wastes CPU time slices and can lead to priority inversion Questions is a disadvantage of it.

Consider a computer with two processes, H priority is higher and L priority is lower. Scheduling rules specify that H can run only till it is in a ready state. At some point, H is in the ready state, ready to run and L is in the critical region. While L cannot be scheduled because of its low priority but H needs to be busy waiting, so it cannot leave the critical area, so H will always be busy waiting, while L cannot leave the critical area. This situation is called priority inversion problem priority inversion problem）

Blocking and awakening of process

Some primitives provided by operating system, sleep and wakeup.

The primitive is not allowed to interrupt during execution and the process or function provided by the kernel to an out-of-core call becomes a primitive.

Until another process calls the wakeup method, sleep is a system call that calls a blocked process, which takes the blocked process as a parameter and wakes it up. When a process is blocked, the CPU will not allocate time slices to it, while busy waiting is always idle, consuming time slices of the CPU this is what the biggest difference between blocking and busy waiting.

Semaphores

Initially, one need to understand the semaphore is used for what purpose, because the blockage and wake-up of a process are caused by different processes, such as process A calling sleep () will enter the blockage, and process B calling wakeup (A) will wake up process A. Because it is carried out in different processes, there is also the problem of interruption. Procedure A needs to call sleep () into blocking state according to logic. Though, process B starts running because of clock interruption, just before it calls sleep method. According to logic, it calls wakeup () method to wake up process A, but because process A has not entered blocking state, the wakeup signal is lost, waiting for process A to interrupt before. There may be no process to wake it up, when the location starts to continue running and gets blocked.

Hence, by introducing an additional variable the blocking and waking of a process should be recorded, which records the number of wakes-up times, and the value of the variable is added to 1 for each wake-up. With this variable the wakeup operation is recorded in the variable even if the wakeup operation precedes the sleep operation. It is considered that the process does not need to be blocked, when the process sleeps, because other processes have been awakened.

This variable is called semaphore in terms of operating system concept, a method proposed by Dijkstra in 1965. The different semaphores operations are, down and up.

Down operation which first detects whether the semaphore value is greater than 0. This operation corresponds to sleep operation. if semaphore value is greater than 0, it decreases by 1. The process does not need to block at this time, which is equivalent to consuming a wakeup; if the semaphore is 0, the process will enter a blocking state.

The up operation corresponds to wake up. The system will choose one of the processes to wake it up if a process is blocked on the semaphore. The value of the semaphore at this time does not need to change, but the blocked process is one less. It will add the value of the semaphore to 1, if no process is blocked on the semaphore during up operation.

These down and up operations are also called as PV operations because in Dijkstra's paper, P and V are used to represent down and up, respectively.

The semaphores operations which are down and up are primitives supported by the operating system. They are atomic operations without interruption.

Mutex is actually a special case of semaphores. Its value is only 0 and 1. One can use mutex when we do not need the counting ability of semaphores. Actually, semaphores allow multiple processes to enter the critical area at the same time, whereas the critical value allows only one process to enter at the same time.

A semaphore is a variable and this variable is generally used to achieve the process synchronization. It is also used to indicate the number of resources that are available in a system at a particular time. It is generally denoted by "S". one can use any variable name of own choice.

A semaphore uses two functions i.e., *wait()* and *signal()*. To change the value of the semaphore these two functions are used but the value can be changed by only one process at a particular time and no other process should change the value simultaneously.

if the value of the semaphore variable is positive the *wait()* function is used to decrement the value of the semaphore variable "S" by one. No operation will be performed if the value of the semaphore variable is 0.

```
wait(S) {
    while (S == 0); //there is ";" sign here
    S--;
}
```

The *signal()* function is used to increment the value of the semaphore variable by one.

```
signal(S) {
    S++;
}
```

There are two types of semaphores:

- Binary Semaphores: In Binary semaphores, the value of the semaphore variable will be 0 or 1. Initially, the value of semaphore variable is set to 1. The *wait()* function can be called and the value of the semaphore is changed to 0 from 1, if some process wants to use some resource. The process then uses the resource and when it releases the resource then the *signal()* function is called and the value of the semaphore variable is increased to 1. Let us consider at some particular instant of time, some other process wants to use the same resource and the value of the semaphore variable is 0 then it has to wait for the release of the resource by the previous process. In this way, process synchronization can be achieved.

- Counting Semaphores: In Counting semaphores, firstly, the semaphore variable is initialized with the number of resources available. After that, whenever a process needs some resource, then the *wait()* function is called and the value of the semaphore variable is decreased by one. The process then uses the resource and after using the resource, the *signal()* function is called and the value of the semaphore variable is increased by one. When all the resources are taken by the process and the value of the semaphore variable goes to 0 and there is no resource left to be used, the process has to wait for its turn if in case it wants to use resources. the process synchronization is greatly achieved by this way.

Advantages of semaphore

- When you use semaphores because semaphores allow only one process to enter into the critical section then the mutual exclusion principle is followed.
- Here, you need not verify that a process should be allowed to enter into the critical section or not. So, the waste of processor time is avoided here.

Disadvantages of semaphore

- While using semaphore, no other higher priority process allowed to get into the critical section if a low priority process is in the critical section. The process with higher priority has to wait for the complete execution of the lower priority process.
- The implementation of a semaphore is quite difficult, because the *wait()* and *signal()* functions need to be implemented in the correct order.

Inter-process Communication

- Processes within a system may be independent or cooperating
- Other processes including sharing data can affect Cooperating process .
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need inter-process communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models

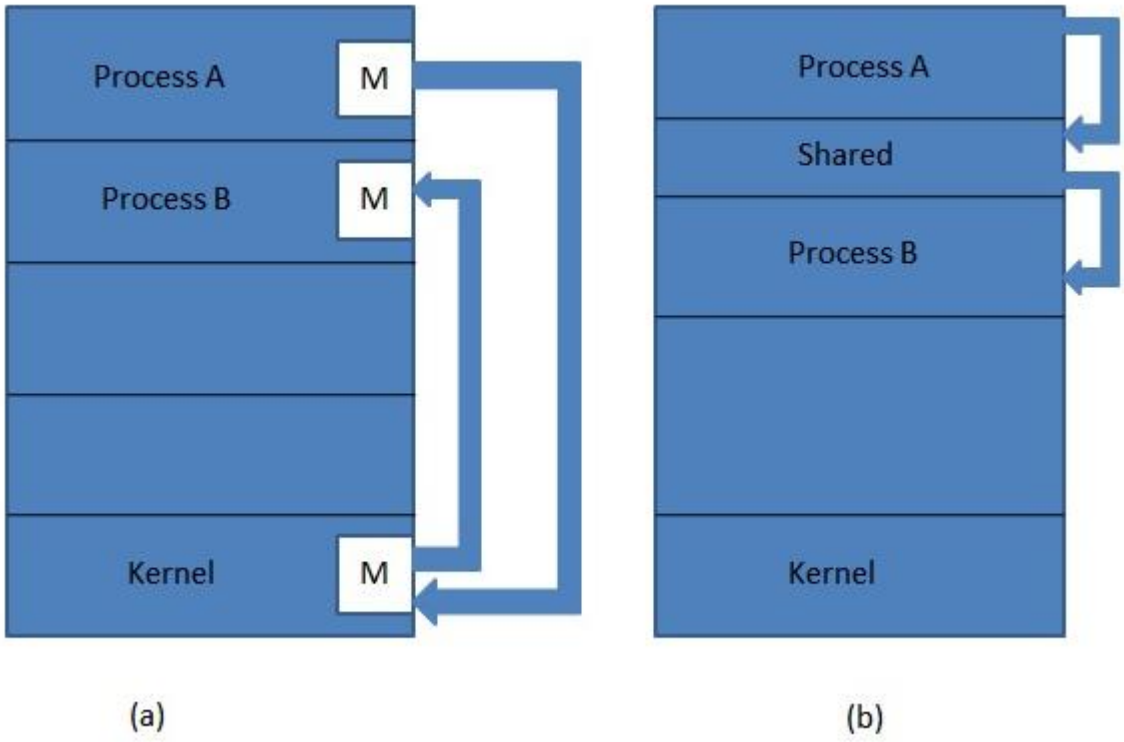


Figure 5.1 Communications Models

Cooperating Processes

- the execution of another process does not affect independent process.
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Advantages of process cooperation
  - Computation speed-up
  - Modularity
  - Convenience

a variety of synchronization methods are used to widely discussed and analysed the problems illustrated by operating systems.

. Let us examine four of the better-known problems as mention follow.

i) Shared Memory Method

Ex: Producer-Consumer problem

There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. Buffer is common space or memory location shared by two processes where the item produced by Producer is stored and from which the Consumer consumes the item, if needed. This problem may be divided in to two versions: the first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on the size of the buffer, the bounded buffer problem is known as a second version in which Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. The producer will start producing items, after Producer and the Consumer will share some common memory. Producer will wait to get it consumed by the Consumer If the total produced item is equal to the size of buffer. Similarly, the consumer will first check for the availability of the item. Consumer will wait for Producer to produce it If no item is available. Consumer will consume the item, if there are items available. The pseudo code to implement the concept is as demonstrated bellow:

Shared Data between the two Processes

```
#define buff_max 25
#define mod %

        struct item{

                // different member of the produced data
                // or consumed data
                ---------
        }

        // An array is needed for holding the items.
        // This is the shared place which will be
        // access by both process
        // item shared_buff [ buff_max ];

        // Two variables which will keep track of
        // the indexes of the items produced by producer
        // and consumer The free index points to
        // the next free index. The full index points to
        // the first full index.
        int free_index = 0;
        int full_index = 0;
```

Producer Process Code

```
item nextProduced;

        while(1){

                // check if there is no space
                // for production.
```

```
            // if so keep waiting.
            while((free_index+1) mod buff_max == full_index);

            shared_buff[free_index] = nextProduced;
            free_index = (free_index + 1) mod buff_max;
        }
```

Consumer Process Code

```
item nextConsumed;

        while(1){

            // check if there is an available
            // item for consumption.
            // if not keep on waiting for
            // get them produced.
            while((free_index == full_index);

            nextConsumed = shared_buff[full_index];
            full_index = (full_index + 1) mod buff_max;
        }
```

In the above code, the Producer will start producing again when the (free_index+1) mod buff max will be free because if it not free, this implies that there is no need to produce more, as there are still items that can be consumed by the Consumer. Similarly, if free index and full index point to the same index, this implies that there are no items to consume.

ii) Messaging Passing Method

Now, let us discuss how the communication between processes via message passing is takes place. In this method, without using any kind of shared memory processes communicate with each other. Two processes p1 and p2 if they wish to communicate with each other, they proceed as follows:

- Establish a communication link if it is not existed if a link already exists, no need to establish it again.
- Using basic primitives start exchanging messages.

  We need at least two primitives:

  – send(message, destination) or send(message)

– receive(message, host) or receive(message)

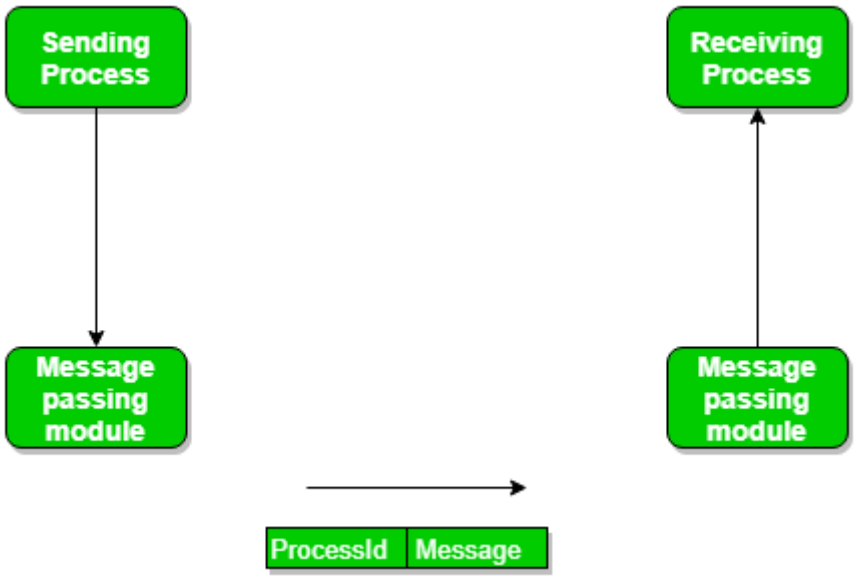


Figure 5.2: **Messaging Passing Method**

The message size can be of fixed size or of variable size. It is easy for an OS designer but complicated for a programmer If it is of fixed size and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message is actually divided into two parts: header and body.

To store carious information such as message type, destination id, source id, message length, and control information the header part is used. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Usually, FIFO style is used to send messages

Message Passing through Communication Link.

Direct and Indirect Communication link

Now, let us discuss different methods of implementing communication link. There are some questions which need to be kept in mind while implementing the link like :

1. How are links established?
2. Can more than two processes be associated with a single link?
3. Between every pair of communicating processes how many links can be there?
4. What is the capacity of a link? Is the size of a message that the link can accommodate fixed or variable?
5. Is a link unidirectional or bi-directional?

A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity. In zero capacity, the receiver informs the sender that it has received the message until then the sender waits. In non-zero capacity cases, after the send

operation a process does not know whether a message has been received or not. For this, the explicate communicate with the receiver must be done by the sender explicitly. Depending on the situation implementation of the link is done, it can be either an in-directed communication link or a direct communication link.

when the processes use a specific process identifier for the communication then direct Communication links are implemented, but it is hard to identify the sender ahead of time.

For example: the print server.

In-direct Communication consists of a queue of messages and it is done via a shared mailbox (port). The sender places the message in mailbox and the receiver picks them up.

Message Passing through Exchanging the Messages.

Synchronous and Asynchronous Message Passing:

A process that is waiting for some event is blocked, the event for which it may be blocked such as a resource becoming available or the completion of an I/O operation. IPC can take place not only between the processes on same computer but as well on the processes running on different computer i.e., in networked/distributed system. In both cases, while sending a message or attempting to receive a message the process may or may not be blocked so message passing may be blocking or non-blocking. Blocking is also known as synchronous and in synchronous communication the sender will be blocked until the message is received by receiver which is called as blocking send. Correspondingly, the receiver block until a message is available is called blocking receive. Non-blocking is also known as asynchronous communication and in which the sender sends the message and continue which is called as non-blocking send. Correspondingly, the receiver receives a valid message or null which is known as non-blocking receive. After a careful analysis, one can conclude 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. Though, in case the send fails the sender expects acknowledgement from the receiver. 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. At the same time, if the message send keep on failing, the receiver will have to wait indefinitely. That is why we also consider the other possibility of message passing. There are basically three preferred combinations:

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

In case of Direct message passing, The process must explicitly name the recipient or sender of communication when it wants to communicate.

e.g., send(p1, message) means send the message to p1.

similarly, receive(p2, message) means receive the message from p2.

In this method of communication, the communication link can be either unidirectional or bidirectional and it gets established automatically, but one link can be used between one pair of

the sender and receiver and more than one pair of links cannot be possess by one pair of sender and receiver. Between sending and receiving Symmetry and asymmetry can also be implemented i.e., either for sending and receiving the messages both processes will name each other or only the sender will name receiver for sending the message and there is no need for receiver for naming the sender for receiving the message. In this method if the name of one process changes, this method will not work, this is the big problem with this method of communication.

In Indirect message passing, for sending and receiving messages processes use mailboxes which is also called as ports. Each mailbox has a unique id and if they share a mailbox then only processes can communicate. A single link can be associated with many processes but link established only if processes share a common mailbox. Several communication links can be shared by each pair of processes and these links may be unidirectional or bi-directional.

Suppose, the operations that are required if two processes want to communicate though Indirect message passing are: create a mail box, use this mail box for sending and receiving messages, then destroy the mail box. The standard primitives used are: send(A, message) which means send the message to mailbox A. In the same way the primitive for the receiving the message works e.g., received (A, message). In this mailbox implementation there is a problem. Suppose the same mail box is shared by more than two processes and suppose the process p1 sends a message to the mailbox, which process will be the receiver? This can be solved by either enforcing that only two processes can share a single mailbox or enforcing that only one process is allowed to execute the receive at a given time or select any process randomly and notify the sender about the receiver.

A mailbox can also be shared between multiple sender/receiver pairs and can be made private to a single sender/receiver pair. A mailbox implemented with multiple sender and single receiver is called as port, which is used in client/server applications. The port is created by OS on the request of the receiver process and owned by the receiving process and can be destroyed either on request of the same receiver process or when the receiver terminates itself. Enforcing that only one process is allowed to execute the receive can be done using the concept of mutual exclusion. The n processes share the Mutex mailbox. Sender is non-blocking and sends the message. All other processes will be blocking and will wait when the first process which executes the receive and will enter in the critical section.

Now, Let us use message passing concept to understand and discuss the Producer-Consumer problem. The consumer can consume an item when producer places items inside messages in the mailbox and when at least one message present in the mailbox. The code is given below:

Producer Code

```
void Producer(void){

        int item;
        Message m;

        while(1){
```

```
                receive(Consumer, &m);
                item = produce();
                build_message(&m , item ) ;
                send(Consumer, &m);
        }
    }
```

Consumer Code

```
void Consumer(void){

        int item;
        Message m;

        while(1){

                receive(Producer, &m);
                item = extracted_item();
                send(Producer, &m);
                consume_item(item);
        }
    }
```

Examples of IPC systems

1. Posix : uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:

There is various mechanism:

- Pipe
- Socket
- Remote Procedural calls (RPCs)

Some of the traditional problem mentioned bellow are representing flaws of process synchronization in systems where cooperating processes are present .

We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem.
3. The Readers Writers Problem

Bounded Buffer Problem

- A finite buffer pool is used to exchange messages between producer and consumer processes, this problem is actually generalised in terms of the Producer Consumer problem.
- This problem is often called the Bounded buffer problem Because the buffer pool has a maximum size.
- Solution to this problem is, to keep track of the current number of full and empty buffers respectively create two counting semaphores "full" and "empty".

The classic problem of synchronization is Bounded buffer problem, which is also called producer consumer problem. Let us try to understand the problem.

What is the Problem Statement?

There is a buffer of n slots and each slot is capable of storing one unit of data. Consider two processes running and operating on the buffer, namely, producer and consumer.

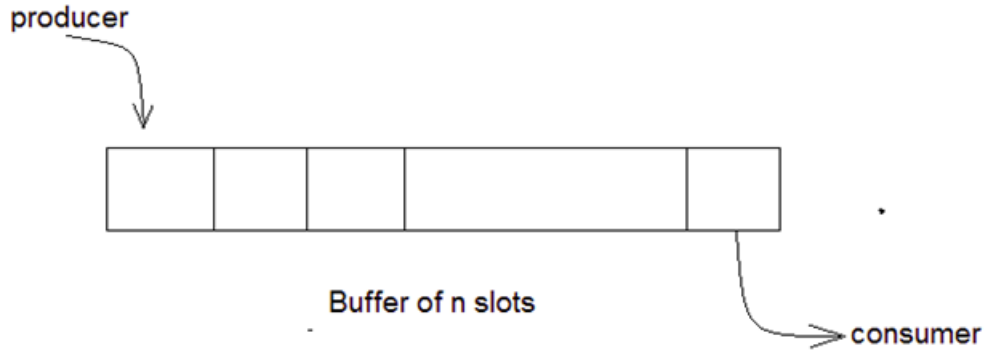


Figure 5.3 Bounded Buffer Problem

Into an empty slot of the buffer a producer tries to insert data. Whereas from a filled slot in the buffer consumer tries to remove data. Hence, if they are being executed concurrently, those two processes won't produce the expected output.

To make the producer and consumer work in an independent manner there needs a way.

Here's a Solution: The use of semaphores is one solution of this problem. The semaphores which will be used here are:

- To acquire and release the lock `m`, a binary semaphore is used.
- Since, initially all slots are empty and hence, `empty`, a counting semaphore are used whose initial value is the number of slots in the buffer.
- `full`, a counting semaphore whose initial value is `0`.

At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

The Producer Operation

The pseudocode of the producer function looks like this:

```
do
{
   // wait until empty > 0 and then decrement 'empty'
   wait(empty);
   // acquire lock
   wait(mutex);

   /* perform the insert operation in a slot */

   // release lock
   signal(mutex);
   // increment 'full'
   signal(full);
}
while(TRUE)
```

- Here is the code for a producer as mentioned above, we can see that a producer first waits until there is at least one empty slot.
- As there will now be one less empty slot it decrements the empty semaphore because, since the producer is going to insert data in one of those slots.
- To restrict the consumer to access the buffer until producer completes its operation, it acquires lock on the buffer.
- The lock is released after performing the insert operation, and the value of full is incremented because the producer has just filled a slot in the buffer.

The Consumer Operation

The pseudocode for the consumer function looks like this:

```
do
{
   // wait until full > 0 and then decrement 'full'
   wait(full);
   // acquire the lock
   wait(mutex);

   /* perform the remove operation in a slot */

   // release the lock
   signal(mutex);
   // increment 'empty'
   signal(empty);
}
while(TRUE);
```

- The consumer waits until there is at least one full slot in the buffer.
- Then it decrements the full semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.
- After that, the consumer acquires lock on the buffer.
- Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.
- Then, the consumer releases the lock.
- Finally, the empty semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.

Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes is the dining philosophers' problem is.

What is the Problem Statement?

Consider around a circular dining table there are five philosophers sitting. In the middle of dining table, it has five chopsticks and a bowl of rice as shown in the below figure.

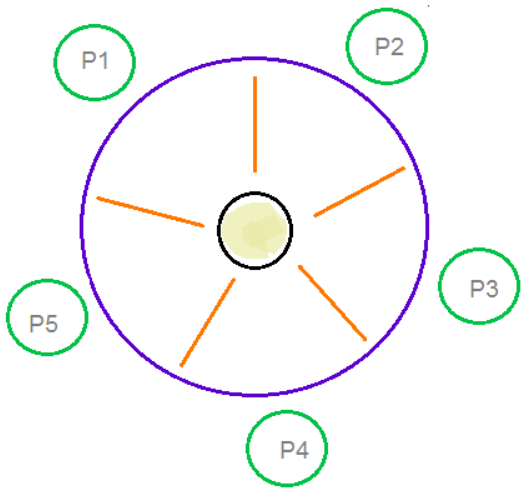


Figure 5.4 Dining Philosophers Problem

At any instant, a philosopher is either eating or thinking. When a philosopher uses two chopsticks - one from their left and one from their right when he wants to eat. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Here's the Solution

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, stick[5], for each of the five chopsticks.

The code for each philosopher looks like:

```
while(TRUE)
{
   wait(stick[i]);
   /*
      mod is used because if i=5, next
      chopstick is 1 (dining table is circular)
   */
   wait(stick[(i+1) % 5]);

   /* eat */
   signal(stick[i]);

   signal(stick[(i+1) % 5]);
   /* think */
}
```

When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever. The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.
- Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

The Readers Writers Problem

- In this problem there are some processes(called readers) that only read the shared data, and never change it, and there are other processes(called writers) who may change the data in addition to reading, or instead of reading it.
- There is various type of readers-writers problem, most centred on relative priorities of readers and writers.

  Reader's writer problem is another example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

The Problem Statement

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non-zero number of readers accessing the resource at that time.

The Solution

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.
Here, we use one mutex `m` and a semaphore `w`. An integer variable `read_count` is used to maintain the number of readers currently accessing the resource. The variable `read_count` is initialized to `0`. A value of `1` is given initially to `m` and `w`. Instead of having the process to acquire lock on the shared resource, we use the mutex `m` to make the process to acquire and release lock whenever it is updating the `read_count` variable.

The code for the writer process looks like this:

```
while(TRUE)
{
   wait(w);

  /* perform the write operation */

  signal(w);
}
```

And, the code for the reader process looks like this:

```
while(TRUE)
{
```

```
    //acquire lock
    wait(m);
    read_count++;
    if(read_count == 1)
        wait(w);

    //release lock
    signal(m);

    /* perform the reading operation */

    // acquire lock
    wait(m);
    read_count--;
    if(read_count == 0)
        signal(w);

    // release lock
    signal(m);
}
```

Here is the Code uncoded:

- As seen above in the code for the writer, the writer just waits on the w semaphore until it gets a chance to write to the resource.
- After performing the write operation, it increments w so that the next writer can access the resource.
- On the other hand, in the code for the reader, the lock is acquired whenever the read_count is updated by a process.
- When a reader wants to access the resource, first it increments the read_count value, then accesses the resource and then decrements the read_count value.
- The semaphore w is used by the first reader which enters the critical section and the last reader which exits the critical section.
- The reason for this is, when the first readers enter the critical section, the writer is blocked from the resource. Only new readers can access the resource now.
- Similarly, when the last reader exits the critical section, it signals the writer using the w semaphore because there are zero readers now and a writer can have the chance to access the resource.

Peterson's Algorithm in Process Synchronization

Problem: The producer consumer problem (or bounded buffer problem) describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue. Producer produce an item and put it into buffer. If buffer is already full then producer will have to wait for an empty block in buffer. Consumer consume an item from buffer. If buffer is already empty then consumer will have to wait for an item in buffer. Implement Peterson's Algorithm for the two processes using shared memory such that there is mutual exclusion between them. The solution should have free from synchronization problems.

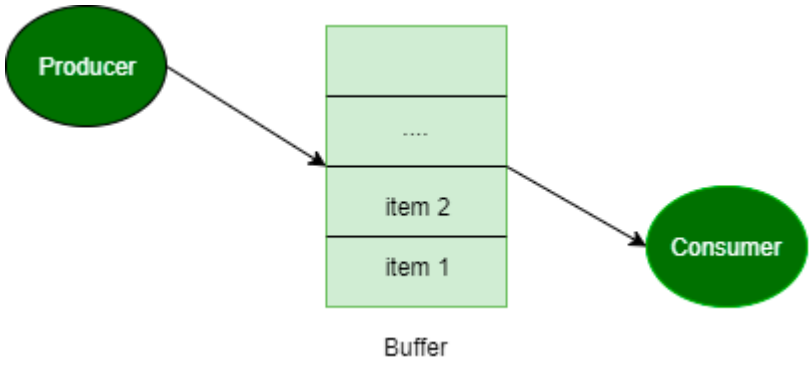


Figure 5.5: Peterson's Algorithm in Process Synchronization

Peterson's algorithm –

```
// code for producer (j)

// producer j is ready
// to produce an item
flag[j] = true;

// but consumer (i) can consume an item
turn = i;

// if consumer is ready to consume an item
// and if its consumer's turn
while (flag[i] == true && turn == i)

        { // then producer will wait }

        // otherwise, producer will produce
        // an item and put it into buffer (critical Section)

        // Now, producer is out of critical section
        flag[j] = false;
        // end of code for producer

        //-------------------------------------------------------
```

```
        // code for consumer i

        // consumer i is ready
        // to consume an item
        flag[i] = true;

        // but producer (j) can produce an item
        turn = j;

        // if producer is ready to produce an item
        // and if its producer's turn
        while (flag[j] == true && turn == j)

                { // then consumer will wait }

                // otherwise consumer will consume
                // an item from buffer (critical Section)

                // Now, consumer is out of critical section
                flag[i] = false;
// end of code for consumer
```

Explanation of Peterson's algorithm

Peterson's Algorithm is used to synchronize two processes. It uses two variables, a bool array flag of size 2 and an int variable turn to accomplish it.

In the solution i represents the Consumer and j represents the Producer. Initially the flags are false. When a process wants to execute its critical section, it sets its flag to true and turn as the index of the other process. This means that the process wants to execute but it will allow the other process to run first. The process performs busy waiting until the other process has finished its own critical section.
After this the current process enters its critical section and adds or removes a random number from the shared buffer. After completing the critical section, it sets its own flag to false, indication it does not wish to execute anymore.

The program runs for a fixed amount of time before exiting. This time can be changed by changing value of the macro-RT.

```
// C program to implement Peterson's Algorithm
// for producer-consumer problem.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
```

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdbool.h>
#define _BSD_SOURCE
#include <sys/time.h>
#include <stdio.h>

#define BSIZE 8 // Buffer size
#define PWT 2 // Producer wait time limit
#define CWT 10 // Consumer wait time limit
#define RT 10 // Program run-time in seconds

int shmid1, shmid2, shmid3, shmid4;
key_t k1 = 5491, k2 = 5812, k3 = 4327, k4 = 3213;
bool* SHM1;
int* SHM2;
int* SHM3;

int myrand(int n) // Returns a random number between 1 and n
{
	time_t t;
	srand((unsigned)time(&t));
	return (rand() % n + 1);
}

int main()
{
	shmid1 = shmget(k1, sizeof(bool) * 2, IPC_CREAT | 0660); // flag
	shmid2 = shmget(k2, sizeof(int) * 1, IPC_CREAT | 0660); // turn
	shmid3 = shmget(k3, sizeof(int) * BSIZE, IPC_CREAT | 0660); // buffer
	shmid4 = shmget(k4, sizeof(int) * 1, IPC_CREAT | 0660); // time stamp

	if (shmid1 < 0 || shmid2 < 0 || shmid3 < 0 || shmid4 < 0) {
		perror("Main shmget error: ");
		exit(1);
	}
	SHM3 = (int*)shmat(shmid3, NULL, 0);
	int ix = 0;
	while (ix < BSIZE) // Initializing buffer
		SHM3[ix++] = 0;

	struct timeval t;
	time_t t1, t2;
	gettimeofday(&t, NULL);
	t1 = t.tv_sec;

	int* state = (int*)shmat(shmid4, NULL, 0);
	*state = 1;
	int wait_time;
```

```
int i = 0; // Consumer
int j = 1; // Producer

if (fork() == 0) // Producer code
{
	SHM1 = (bool*)shmat(shmid1, NULL, 0);
	SHM2 = (int*)shmat(shmid2, NULL, 0);
	SHM3 = (int*)shmat(shmid3, NULL, 0);
	if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
		perror("Producer shmat error: ");
		exit(1);
	}

	bool* flag = SHM1;
	int* turn = SHM2;
	int* buf = SHM3;
	int index = 0;

	while (*state == 1) {
		flag[j] = true;
		printf("Producer is ready now.\n\n");
		*turn = i;
		while (flag[i] == true && *turn == i)
			;

		// Critical Section Begin
		index = 0;
		while (index < BSIZE) {
			if (buf[index] == 0) {
				int tempo = myrand(BSIZE * 3);
				printf("Job %d has been produced\n", tempo);
				buf[index] = tempo;
				break;
			}
			index++;
		}
		if (index == BSIZE)
			printf("Buffer is full, nothing can be produced!!!\n");
		printf("Buffer: ");
		index = 0;
		while (index < BSIZE)
			printf("%d ", buf[index++]);
		printf("\n");
		// Critical Section End

		flag[j] = false;
		if (*state == 0)
			break;
		wait_time = myrand(PWT);
		printf("Producer will wait for %d seconds\n\n", wait_time);
```

```
                sleep(wait_time);
        }
        exit(0);
}

if (fork() == 0) // Consumer code
{
        SHM1 = (bool*)shmat(shmid1, NULL, 0);
        SHM2 = (int*)shmat(shmid2, NULL, 0);
        SHM3 = (int*)shmat(shmid3, NULL, 0);
        if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
                perror("Consumer shmat error:");
                exit(1);
        }

        bool* flag = SHM1;
        int* turn = SHM2;
        int* buf = SHM3;
        int index = 0;
        flag[i] = false;
        sleep(5);
        while (*state == 1) {
                flag[i] = true;
                printf("Consumer is ready now.\n\n");
                *turn = j;
                while (flag[j] == true && *turn == j)
                        ;

                // Critical Section Begin
                if (buf[0] != 0) {
                        printf("Job %d has been consumed\n", buf[0]);
                        buf[0] = 0;
                        index = 1;
                        while (index < BSIZE) // Shifting remaining jobs forward
                        {
                                buf[index - 1] = buf[index];
                                index++;
                        }
                        buf[index - 1] = 0;
                } else
                        printf("Buffer is empty, nothing can be consumed!!!\n");
                printf("Buffer: ");
                index = 0;
                while (index < BSIZE)
                        printf("%d ", buf[index++]);
                printf("\n");
                // Critical Section End

                flag[i] = false;
                if (*state == 0)
```

```
                                break;
                        wait_time = myrand(CWT);
                        printf("Consumer will sleep for %d seconds\n\n", wait_time);
                        sleep(wait_time);
                }
                exit(0);
        }
        // Parent process will now for RT seconds before causing child to terminate
        while (1) {
                gettimeofday(&t, NULL);
                t2 = t.tv_sec;
                if (t2 - t1 > RT) // Program will exit after RT seconds
                {
                        *state = 0;
                        break;
                }
        }
        // Waiting for both processes to exit
        wait();
        wait();
        printf("The clock ran out.\n");
        return 0;
}
```

Output:

```
Producer is ready now.

Job 9 has been produced
Buffer: 9 0 0 0 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 8 has been produced
Buffer: 9 8 0 0 0 0 0 0
Producer will wait for 2 seconds

Producer is ready now.

Job 13 has been produced
Buffer: 9 8 13 0 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 23 has been produced
Buffer: 9 8 13 23 0 0 0 0
Producer will wait for 1 seconds

Consumer is ready now.
```

Job 9 has been consumed
Buffer: 8 13 23 0 0 0 0 0
Consumer will sleep for 9 seconds

Producer is ready now.

Job 15 has been produced
Buffer: 8 13 23 15 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 13 has been produced
Buffer: 8 13 23 15 13 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 11 has been produced
Buffer: 8 13 23 15 13 11 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 22 has been produced
Buffer: 8 13 23 15 13 11 22 0
Producer will wait for 2 seconds

Producer is ready now.

Job 23 has been produced
Buffer: 8 13 23 15 13 11 22 23
Producer will wait for 1 seconds

The clock ran out.

Exercise Questions

1. State and explain Producer-Consumer Problem with solution.
2. Write a short note on mutual exclusion.
3. What is IPC? Explain the various methods of IPC.
4. Explain in short Dining Philosophers Problem.
5. What is Bounded Buffer problem? Explain in short.
6. What is Blocking and awakening of process?
7. Explain in detail the concept of semaphore?
8. With suitable example define and explain race condition.

References:

1. Operating System Concepts, 8th Edition, by Galvin et al, 2008, Wiley Publications.
2. Lecture notes and ppt of Ariel J. Frank, Bar-Ilan University.
3. Operating Systems | Internals and Design Principles | by William Stallings, Ninth Edition | By Pearson Publications
4. Web Portal: https://www.geeksforgeeks.org

Unit 6 *I/O Management And Deadlock*

Objectives:

- To study the concepts of IO management
- To study device drivers
- Device-Independent I/O Software
- Synchronous vs asynchronous I/O
- TO study Kernel I/O Subsystem
- To study Physical Organization of CD-ROM
- To study Structure of a hard disk
- To study concept of Deadlocks

6. 1 Introduction

I/O software is generally organized in the following form of layers −

- **User Level Libraries** − This provides simple interface to the user program to perform input and output. For example, **stdio** is a internal library provided by C as well as C++ programming languages.
- **Kernel Level Modules** − This provides device driver to interface with the device controller and device independent I/O modules which can be used by the device drivers.
- **Hardware** − This layer has actual hardware and hardware controller which basically interact with the device drivers and it makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

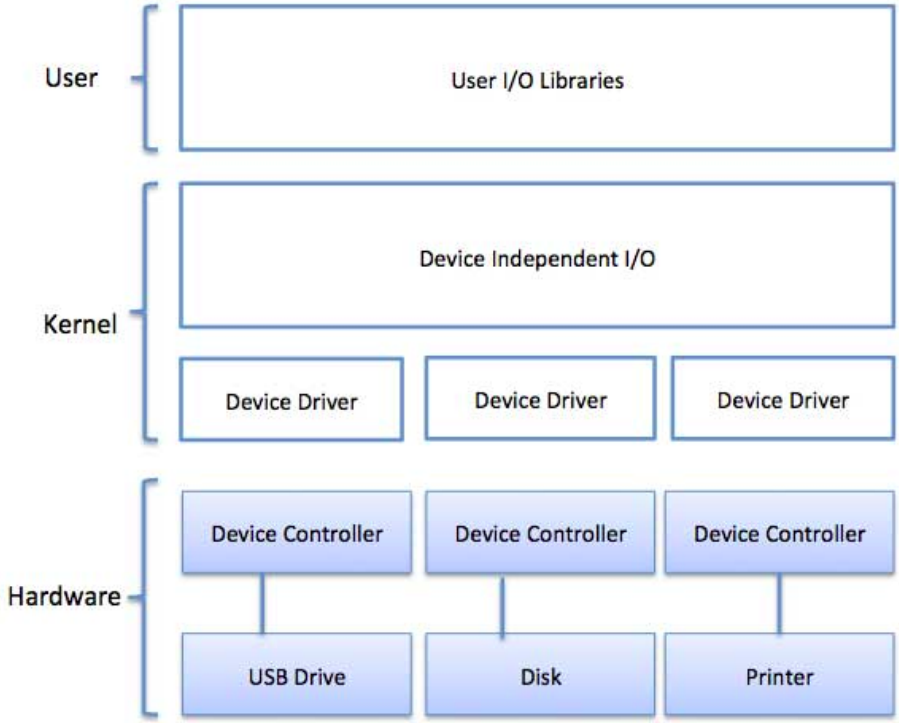


Fig 6.1 Operating System interface

6.2 Device Drivers

Device drivers are software modules which can be connected into an OS to handle any particular device. Operating System gets help from device drivers to handle all kind of I/O devices. Device drivers generally encapsulate device-dependent code and it implement a standard interface in a way that code contains device-specific register for reads and writes. Device driver, is often written by the device's manufacturer and delivered along with the device on a CD-Drive.

A device driver performs the following jobs −

- To accept request from the device independent software above to it.
- Interaction with the device controller to take and give I/O and perform required error handling.
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is not active at the time of request arrival, it starts carrying out the request directly. Otherwise, if the driver is already busy with some other kind of request, it places the new request in the sequence of pending requests.

6.3 **Interrupt handlers:**

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt is there, the interrupt procedure does whatever it has to in order to handle the associated interrupt, updates the data structures and wakes up process that was waiting for interrupts to get it executed.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

6.4 **Device-Independent I/O Software**

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is very hard to write complete device-independent software but we can write some modules which are common among all of the given devices. Following is a list of functions of device-independent I/O Software −

1. Uniform interfacing for device drivers
2. Device naming - Mnemonic names mapped to Major and Minor device numbers
3. Device protection
4. Providing a device-independent block size
5. Buffering because data coming off a device cannot be stored in final destination.
6. Storage allocation on block devices
7. Allocation and releasing dedicated devices
8. Error Reporting

6.5 **User-Space I/O Software**

These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Many of the user-level I/O software consists of library procedures with some kinds of exceptions such as spooling system which is a way to deal with dedicated I/O devices in a multiprogramming software system.

The programming input-output Libraries (e.g., stdio, stdlib, string) are in user-space to provide an interface to the OS resident device-independent input output software. For example puts(), gets(), printf() and scanf() are example of user level input output library stdio available in C programming language.

6.6 **Kernel I/O Subsystem**

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

- **Scheduling** − The kernel schedules a set of I/O requests in order to determine a good order in which to execute all of them. When an application issues a blocking input/output system call, the request is placed on the queue for that device. The Kernel I/O scheduler then rearranges the order of the queue to improve the complete system efficiency and the average response time experienced by the applications in operating systems.
- **Buffering** − The Kernel input-output Subsystem maintains a memory area called as **buffer** which stores data while they are transferred between two devices or between a device with an application operation. The buffering is done to cope up with a speed mismatch between the producer and consumer of a data operation stream or to adapt between devices that have different sizes of data transfer.
- **Caching** − The kernel maintains cache memory which is the region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original copy.
- **Spooling and Device Reservation** − A spool is basically a buffer that holds output for a device, like a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In many operating systems, spooling is generally managed by a system daemon processes. In other operating systems, it is basically handled by an kernel thread.
- **Error Handling** − The operating system that uses protected memory can guard against many kinds of hardware as well as application errors.

6.7 Terminal I/O(Terminal Hardware, Terminal):

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. Input-output devices can be broadly divided into two categories −

- **Block devices** − It is one with which the driver communicates by sending the entire blocks of data. For example, USB cameras, Hard disks, Disk-On-Key etc.
- **Character devices** − This is one with which generally the driver communicates by sending and receiving single characters (such as bytes). For example, serial ports, parallel ports, sounds cards etc.

6.7.1 Device Controllers

Device drivers are software buldles that can be plugged into an operating system to handle a particular device. Operating System gets help from device drivers to handle all kinds of I/O devices.
Here, the device controller works like an interface between a device and a device driver. Input output units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is generally called the device controller.

There is always a device controller and a device driver available for each device for facilitating the communication with the Operating Systems. The device controller may be able to handle multiple devices as a time. As an interface, main task of device controllers is to convert serial bit stream to block of bytes stream, and perform error correction as per necessity.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.

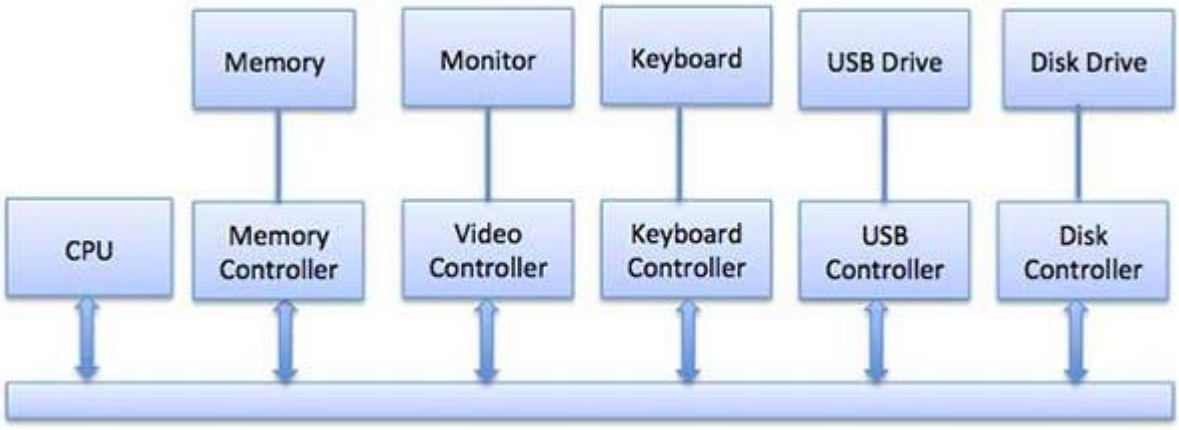


Fig. 6.2 Device Controllers

6.8 Synchronous vs asynchronous I/O

- **Synchronous I/O** − In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** − I/O proceeds concurrently with CPU execution

6.8.1 **Communication to I/O Devices**

The Central Processing Unit must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

6.8.2 **Special Instruction I/O**

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

6.8.3 **Memory-mapped I/O**

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.

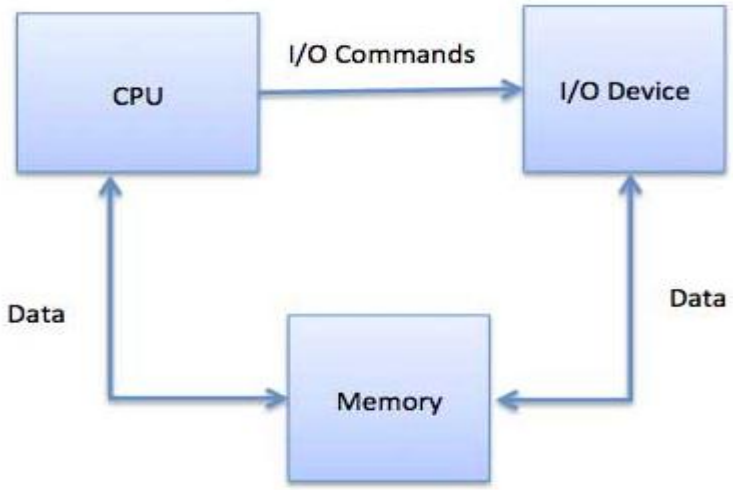


Fig. 6.3 Memory Mapped IO

While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

6.8.4 **Direct Memory Access (DMA)**

Slow devices such as keyboards will generate an interrupt to the main CPU after each byte is transmitted. For any fast device like a disk generated an interrupt for each byte, the operating system would spend most of its time handling of these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead on the system.

Direct Memory Access (DMA) means CPU generally grants input output authority to read from or write to memory without any direct of indirect involvement. DMA module itself controls exchange of data between main memory and the input and output device. CPU is only involved at the start and end of the data transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware devicce called as DMA controller (DMAC) that is used to manage the data transfers and arbitrates access to the system bus. The controllers are programmed with the source and destination pointers (where to read or write the data), counters to keep the track of number of transferred bytes, and its settings, which includes input output and memory types, interrupts and states for the CPU cycles.

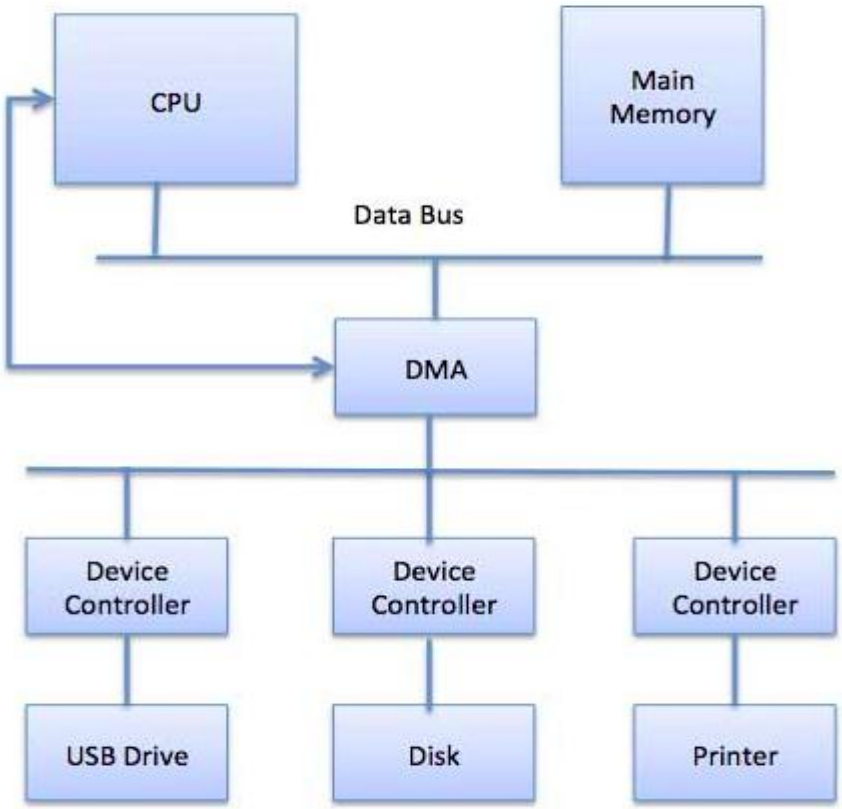


Fig. 6.4 Direct Memory Access

The operating system basically uses the DMA hardware as follows –

Table 6.1 DMA hardware Description used in Operating System

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

6.8.5 **Polling vs Interrupts I/O**

A computer system must have a way to detect the arrival of any type of input sugnal. There are two ways present that this can happen, known as **polling** and **interrupts**. Both of these kinds of techniques allow the processor to deal with events which can happen at any time and that are not related to the process it is currently running.

Polling I/O

It is the simplest way for an I/O device used to communicate with the processor. The system of process of periodically checking and finding status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information into the Status register of processor, and the processor must come and get the information from there.

Most of the time, devices will not require the attention seeking and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary types of polls.

We can compare this kind of method to a teacher continously asking every student in a class, one after another, if they need any kind of help. Definitely, the more efficient method would be for a student to inform the teacher whenever they require to have the assistance.

Interrupts I/O

This is an alternative scheme for dealing with input-output is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires the attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

6.9 Physical Organization of CD-ROM

- Compact Disk – read only memory (write once)
- Data is encoded and read optically with a laser
- Can store around 600MB data

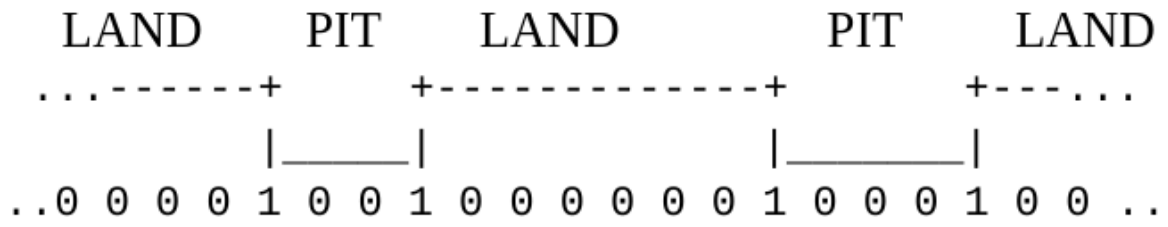
Digital data is represented as a series ofPits and Lands:

- Pit= a little depression, forming a lower level in thetrack
- Land= the flat part between pits, or the upper levelsin the track

Organization of data

Reading a CD is done by shining a laser at the discand detecting changing reflections patterns. 1 = change in height (land to pit or pit to land)

0 = a "fixed" amount of time between 1's



Note : we cannot have two 1's in a row!=> uses Eight to Fourteen Modulation (EFM) encoding table.

The basic unit of data stored on the CD-ROM is the *frame.* Each frame contains 24 bytes of input data. One byte of the source data occupies 17 bits on the disk: 14 bits - the code of the byte of the source data and 3 bits of the merge. For error correction, 180 bits are used. Thus, one frame on the disk occupies 17 • 24 + 180 = 588 bits. The frames are combined into *sectors.* The sector contains 2352 data bytes (98 frames) and 882 bytes for error correction and control. The use of error correction algorithms allows providing a qualitative reading of information with an error probability of 10-10 bits.

There are over 17,000 tracks on the working surface of the CD. Each track contains 32 *sectors* with ordinal numbers (addresses) from 0 to 31. Each sector stores user data and service information part with a total of 9996 bits. The sector is divided into 49 *segments* by 204 bits each. Segments are numbered from 0 to 48, with:

• in the zeroth segment is the sector index, which determines its beginning;
• The second segment contains the data providing the clock adjustment for playback (reading the disc);
• The third segment stores the service information: disk type and side, sector number on the track, etc.
• Segments 4 through 48 are used for data.

6.9.1 General principles of writing/reading

The processes of recording information on a CD (and reading from it) are based on geometric phenomena (reflection, absorption, transmission and refraction of light) and wave (interference, diffraction and polarization of light) optics. The CD contains a recording (working) layer, on which a signalogram is plotted with the help of a laser beam in the form of certain alternations of its states corresponding to a logical zero and a logical unit. In this case, the properties and optical characteristics of the recording layer material change. During the reading process, the laser beam captures these changes and converts them into a digital signal. The most widely used method of recording, leading to a change in the reflection coefficient of the material.

Optical recording achieves a high density of data on the disk. Its limit is due to diffraction of light and is determined by the minimum size of the label. The smaller the wavelength of the light flux emitted by the laser, the higher the recording density. Optical recording (and reading) of data is carried out through a transparent substrate, which is also used to protect the disk from damage. The following recording methods are known:

- ablation (ablation), a method of recording once, at which by heating the material of the recording layer by a laser beam its individual regions are removed;
- bubble a write-once-recording method in which the recording layer is expanded. When reading, the ray reflected from the expanded section (bubble) is scattered and therefore has a lower intensity compared to the ray reflected from the flat surface of the layer;
- The method of multiple recording, in which the phase state or the color of the recording layer portion for the data bit changes with the help of a laser beam. The refractive index or the absorption coefficient depends on the phase state of the recording layer material.

The data on the CD are located along the track in the form of a *waveform,* which can be represented as a set of depressions and areas. The shape of the signalogram, or the configuration of the labels, depends on the way the logical levels are coded. As an example, Fig. 6.11 shows the signalograms for two ways of encoding the logical levels with the same source data code 110101:

6.10 Structure of a hard disk

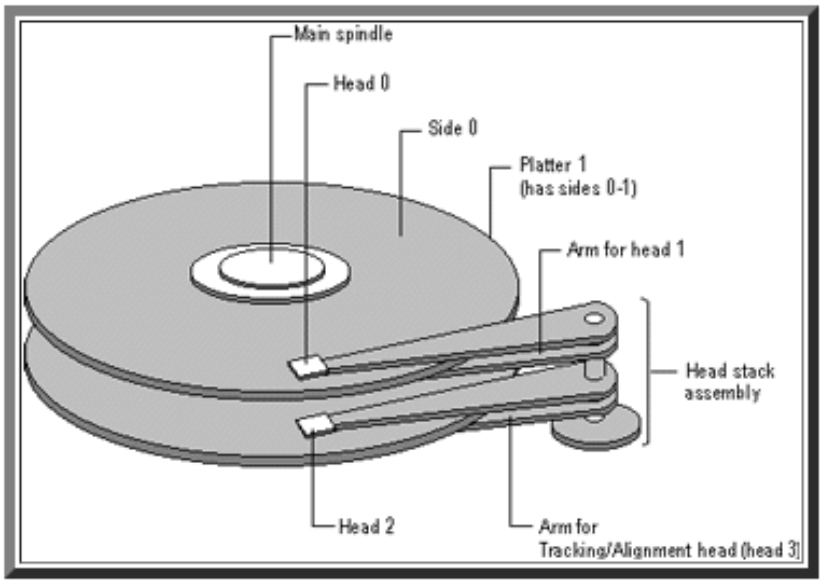


Fig. 6.5 Structure of Hard Drive

- Hard disk consists of a number of platters
- The platters rotate at a very high speed (5400 RPM to 10,000 RPM)
- Disk (read/write) heads move over the platter surface to read and write (magnetize) data bits
- The disk head can read or write data only when the desired disk surface area is under the disk head.

Data access time of data on disk consists of:

1.Seek time (get to the right track)

2. Latency time (wait time for the right sector to rotate under the disk head)
3.Transfer time (actual time needed for reading of the data)

Each platter is logically divided into a number of tracks

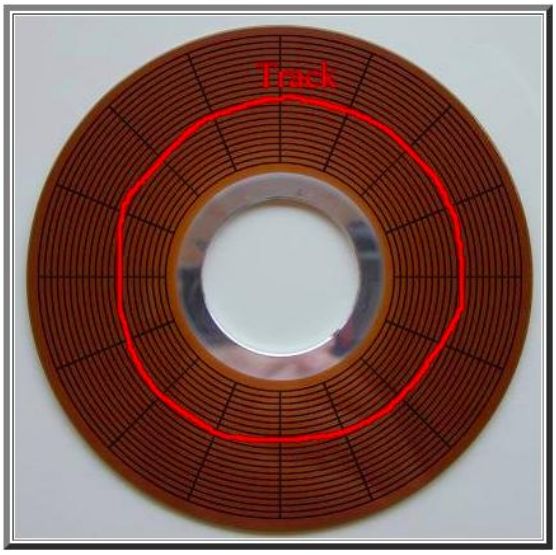


Fig. 6.6 Structure of CD ROM

Each track is logically divided into a number of sectors:

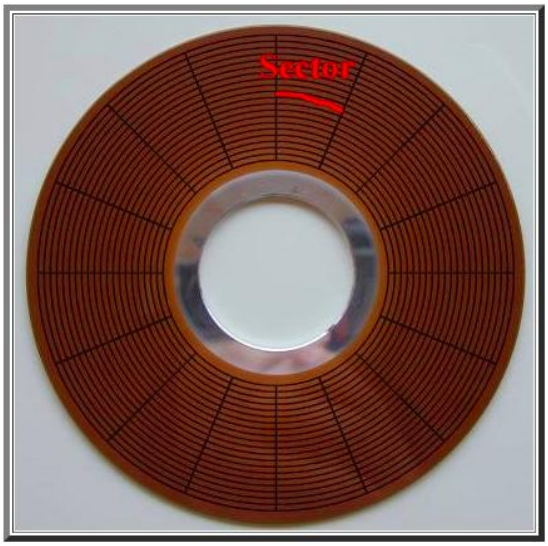


Fig. 6.7 Structure of CD ROM with sectors

Each sector is uniquely identified by its sector number

Disk sectors are usually 512 bytes in size.

To facilitate data access (larger chunks) a number of sectors are **logically** groups into a **disk block**

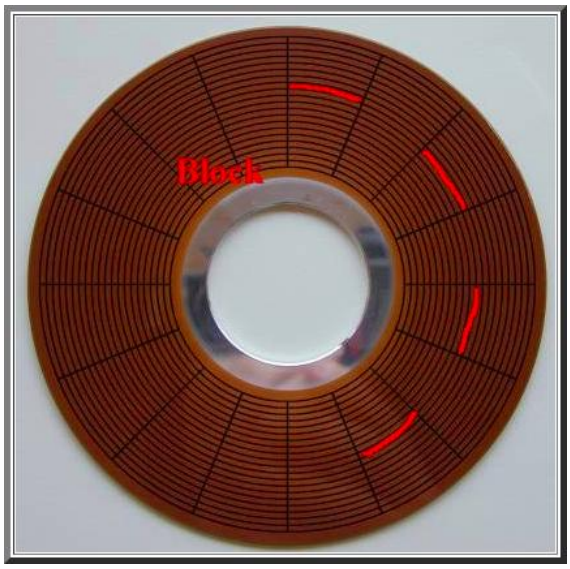


Fig. 6.8 Structure of CD ROM with Blocks

- Each disk block contains a number of sections.
- Each disk block is uniquely identified by its block number.
- There is a mapping defined between disk block numbers and sector numbers.

```
If 1 disk block contains N sectors, then

        disk block k = sector number  k*N. k*N+1, ..., k*N + (N-1)
```

Disk blocks are usually 4K or 8 K bytes in size.

Sectors of a disk block need not be located "contiguously" on the platter (for speedier access, sectors are usually interspersed) E.g., the sectors (4, 5, 6, 7) of this block is **interspersed**:

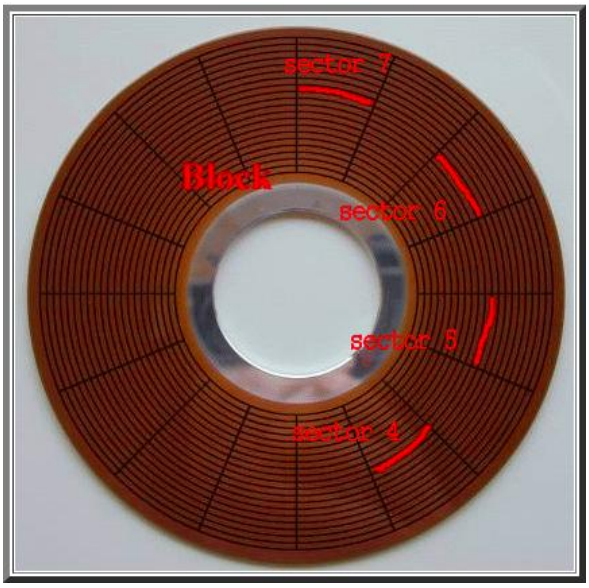


Fig. 6.9 Structure of CD ROM in details

The reason to do so is:

- The sectors of a block will be read consecutively (i.e. one after another)
- Furthermore, the hard disk drive will perform integrity checks on the data (This takes up some time; in the mean time, the disk drive will keep rotating !!!).
- If the sectors are located contiguously, then the disk head may have *passed* the start of the next sector when the drive finishes checking....
  In that case, the drive will have to wait one whole revolution of the platter to read the next sector.
- By interspersing the sectors, the disk drive can achive better access time for the consecutive sectors read/write

6.10.1 Drives and operating system

The drive must be assigned *a drive* letter. That is a task for the operating system, which must be able to recognize the CD-ROM drive. That is usually no problem in Windows 95/98. However, the alphabet can be quite messy, if there are many different drives attached.

Each drive must have its own letter. They are assigned on a first come first-serve-basis. The CD-ROM drive usually gets the first vacant letter after other existing drives, typically D, E, or F. But the letter can be changed in Windows. If you hit Win+Pause, the System box opens. Find your CD-ROM drives like here (The box is Danish, but you'll find it) :

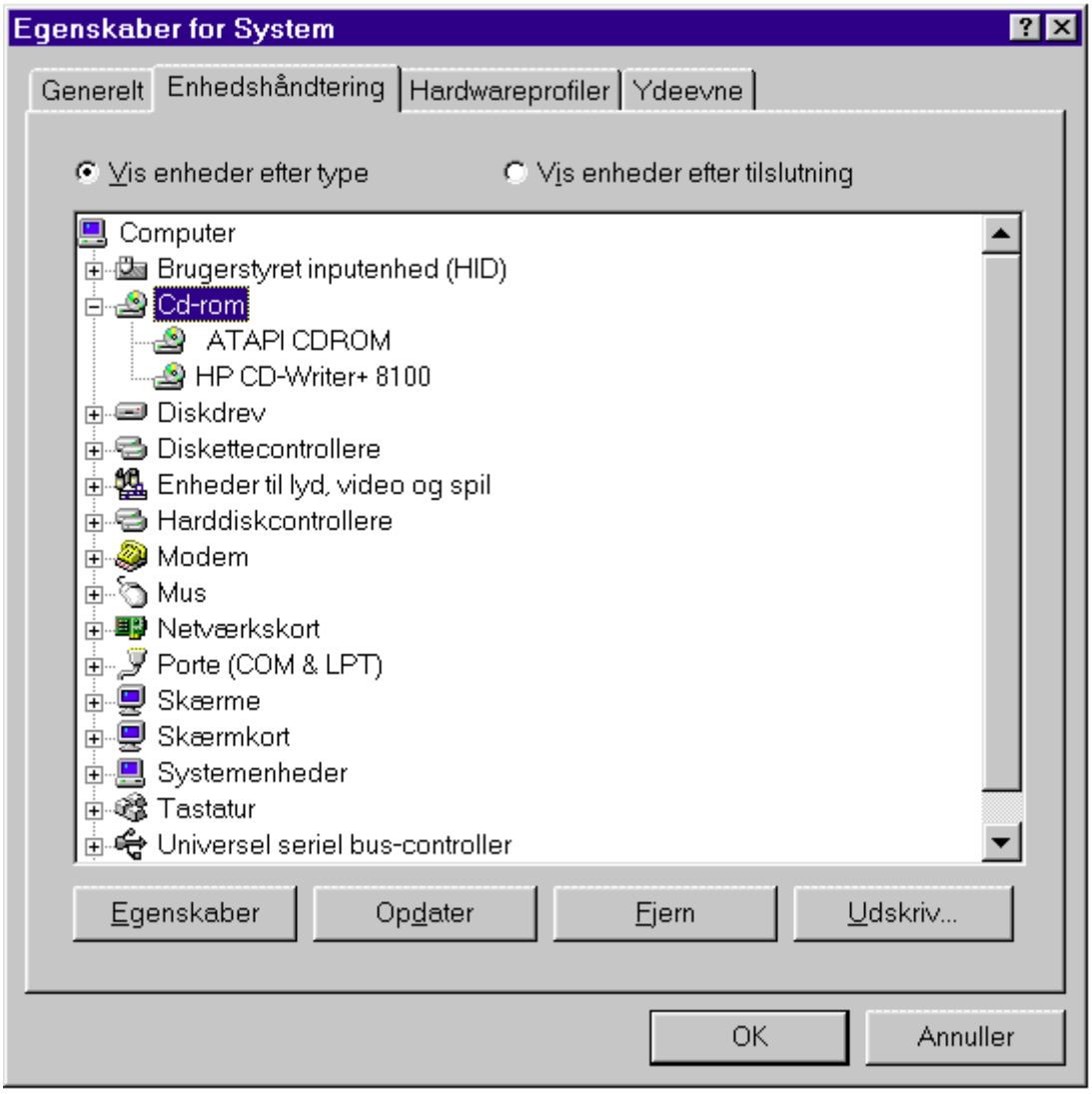


Fig. 6.11 The Hardware devices menu in Operating System

Highlight the drive and choose Properties. Then you can arrange the drive letters:

I like to reserve the letters I: and J: for CD drives. Once the CD-ROM spins and the operating system (DOS or Windows) has "found" the CD-ROM drive, data can be read for processing. Now the CD-ROM works like any other drive. Only, it is Read Only Memory! The CD-ROM holds its own file system called ISO 9660. It is not using FAT!

6.11 An introduction to the DVD:

The DVD is a high-capacity optic media. The DVD standard was developed in the mid 1990s by leading companies like Philips and Sony. DVD stands for Digital Versatile Disk.

The DVD is an all-round disk, which probably will replace CD-ROM and laser disks. Over a few years DVD should replace VHS tapes for videos as well.

Some DVD drives can both read and write the disks. The drives are sold in many versions and with many incompatible sub-standards.

A CD-like disk

The DVD is a flat disk of the same size as a CD. It holds a diameter of 4.7 inches (12 cm) and is .05 inches (1.2 mm) thick. Data are stored in a small indentation in a spiral track, just like in the CD, only the tracks are narrower.

DVD disks are read by a laser beam of shorter wave-length than used by the CD-ROM drives. This allows for smaller indentations and increased storage capacity.

The data layer is only half as thick as in the CD-ROM. This opens the possibility to write data in two layers. The outer gold layer is semi transparent, to allow reading of the underlying silver layer. The laser beam is set to two different intensities, strongest for reading the underlying silver layer. Here you see a common type DVD ROM drive:

Fig. 6.12 The DVD ROM Drive

The DVD drives come in EIDE and SCSI editions and in 5X, etc. versions, like do the CD-ROMs.

Fig. 6.13 The DVD Internals

The DVD drives are often bundled with a MPEG-2 decoder. This is required if you want to replay DVD video disks at optimal quality. Some graphics cards like Matrox-G400 MAX come with a Cine master-based software decoder. This works together with the graphics accelerator chip and gives reasonable DVD replay quality.

The DVD drives will not replace the magnetic hard disks. The hard disks are being improved as rapidly as DVD, and they definitely offer the fastest seek time and transmission rate (currently 20-30 MB/second). No optic media can keep up with this nor with the speedy seeks we get from the hard disks.

But the DVD will undoubtedly gain a place as the successor to the CD-ROM. New drives will read both CD-ROMs and DVDs.

6.11.1 DVD RAM

Three writable technologies are present at the market: Pioneer has a DVD-Recordable technology placing 3.95 GB per disk. DVD-RAM is a RW-disk from Hitachi and Matsushiti.

The 1. Generation disks hold 3.6 GB, while the 2. Generation hold 4.7 GB. The disks are hold in a special cartridge. The so-called DVD+RW, supported by HP, Sony, Philips, Yamaha, Ricoh and Mitsubishi holds up to 4.7 GB per disk.

None of the three products are compatible. However, the companies behind DVD+RW control 75% of the market, so I think this will become the new standard. It appears that the DVD-RAM disks are extremely sensitive to greasy fingers and other contaminants. Therefore they must be handled in special cassettes, which do not fit into ordinary DVD players.

6.12 Deadlocks

A deadlock happens in operating system when two or more processes in the operating system require some resource to complete their execution within system that is held by the other process.

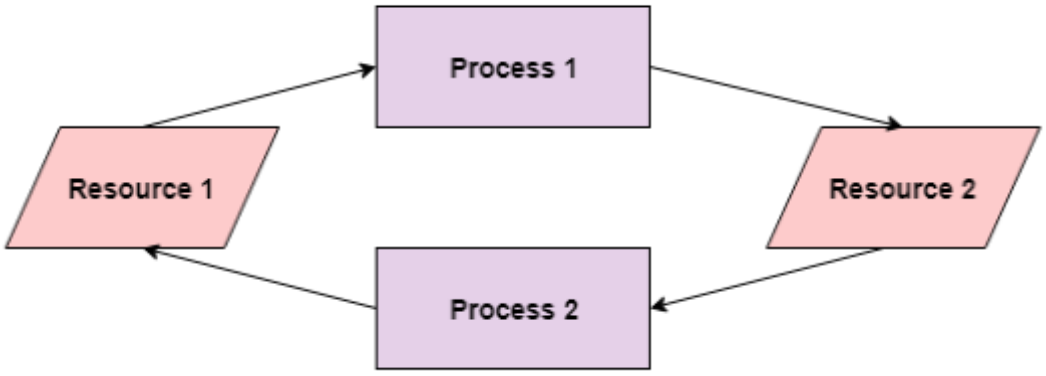


Fig. 6.14 The deadlock in OS

In the above diagram, the process 1 has resource 1 and requires acquiring resource 2. Similar way process 2 has resource 2 and requires to acquire resource 1. Process 1 and process 2 are present in

deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

6.12.1 Coffman Conditions

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows −

6.12.2 Mutual Exclusion

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.

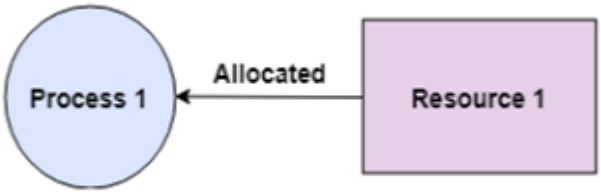


Fig. 6.15 Mutual Exclusion

6.12.3 Hold and Wait

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

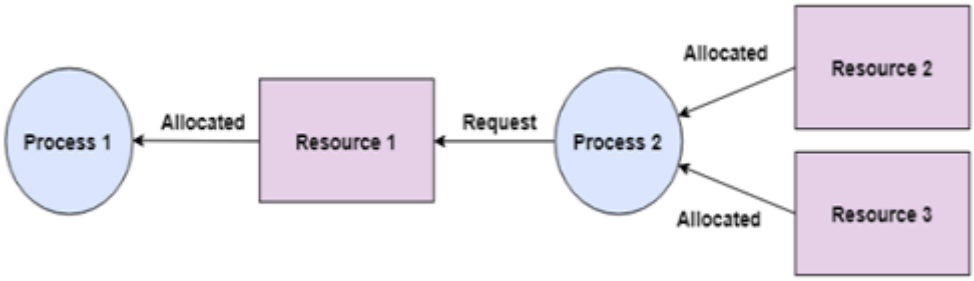


Fig. 6.16 Hold and Wait

6.12.4 No Preemption

A resource cannot be pre-empted forcefully from a process. A process can only release a resource voluntarily by its own. In the diagram below, Process 2 cannot preempt Resource-1 from Process-1. It will only be released when Process-1 releases it by own after its execution is completed.

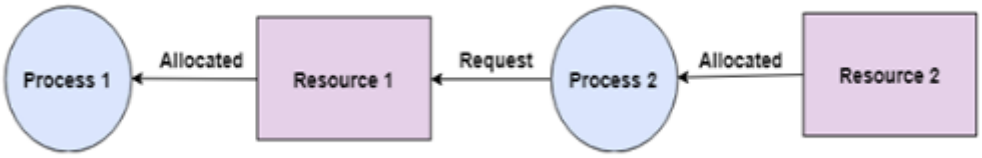


Fig. 6.17 No preemption

6.12.5 Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a system of circular chain. For example: Process1 is allocated Resource2 and it is requesting Resource1. In the same way, Process2 is allocated Resource1 and it is requesting Resource2. This forms circular wait loop system.

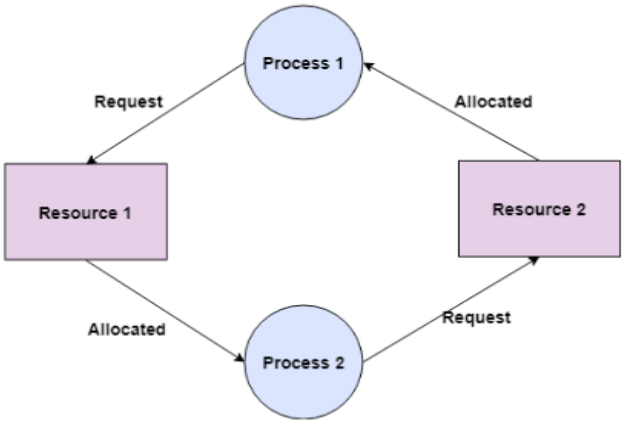


Fig. 6.18 Circular Wait

6.12. 6 Deadlock Detection

A deadlock can be detected by a resource scheduler as it always keeps track of all the resources that are allocated to different kinds of processes. After a deadlock is detection, it can be resolved using the following ways −

- All the processes that are involved in the deadlock are killed. This is not a good approach as all the progress made by the processes is deleted from memory.
- Resources can be preempted from some processes and given to others till the deadlock is resolved.

6.12.7 Deadlock Prevention

It is very essential to prevent the deadlock before it occurs. So, the operating system checks each transaction before it is executed to make sure it will not lead to deadlock. If there is even a small chance that a transaction may lead to deadlock in the future, it will never be allowed to execute.

6.12.8 **Deadlock Avoidance**

It is better to avoid a deadlock rather than taking any kinds of measures after the deadlock has occurred. The wait for graph can be used for avoiding the deadlock. However, this is only useful for smaller databases as it can get quite complex in the larger database.

6.12.9 Problem statements on deadlock:

Deadlock is a problem that can only arise in a system with many different active asynchronous processes. It is important that we learn the three basic approaches to deadlock: prevention, avoidance, and detection (although the terms prevention and avoidance are easy to misidentify).

It can be useful to position a deadlock problem in human terms and ask why human systems never gets a deadlock. Can the students transfer this knowing of human systems to computer systems?

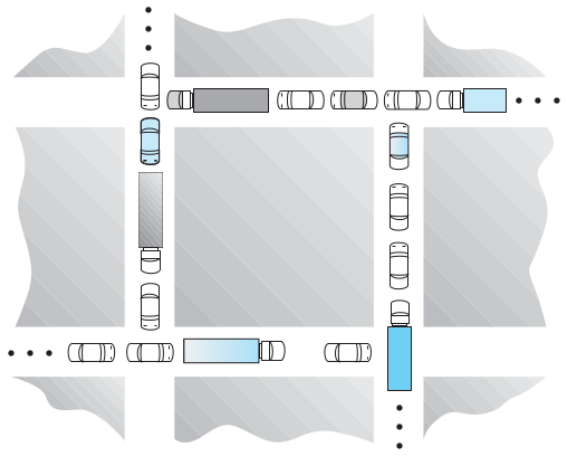


Fig. 6.19 Depiction of Deadlock

Projects can involve simulation: create a list of jobs consisting of requests and releases of resources (single type or multiple types). Ask the students to allocate the resources to prevent deadlock. This basically involves programming the Banker's Algorithm.

1) Consider the traffic deadlock depicted in Figure 6.19.

a. Show that the four necessary conditions for deadlock so hold in this example.

b. State a simple regulation for avoiding deadlocks in this system.

Answer:

a. The four necessary conditions for a deadlock are

(1) mutual exclusion; (2) hold-and-wait; (3) no preemption; and (4) circular wait.

The mutual exclusion condition holds as only one car can occupy a space in that roadway. Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly,

there is indeed a circular wait as each car is waiting for a resultant car to advance. The circular wait condition is also easily observed from the given graphic.

b. A simple rule that would avoid this traffic deadlock is that a car may not progress into an intersection if it is clear they will not be capable to instantly clear the intersection.

2) Consider the deadlock situation that could occur in the dining-philosophers problem when the philosophers obtain the forks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be debared by destroying any one of the four conditions.

Answer:

Deadlock is possible because the 4 necessary conditions hold in the following fashion:

1) Mutual exclusion is required for forks,

2) The philosophers tend to hold onto the fork in hand while they wait for the other fork,

3) There is no preemption of fork in the sense that a fork allocated to a philosopher cannot be taken away by force, and

4) There is a possibility of existence of circular wait.

Deadlocks could be avoided by overcoming the conditions in the following manner:

1) Allow synchronous and simultaneous sharing of forks,

2) Have the philosophers release the first fork if they are unable to obtain the other fork,

3) Allow for forks to be forcibly taken away if a philosopher has had a fork for a long period of time, and

4) Enforce a numbering of the forks and always obtain the lower numbered fork before obtaining the higher numbered one.

3) A possible solution for preventing deadlocks is to have a single, higher-order resource that must be requested before any other resource. For example, if multiple threads attempt to access the synchronization objects A · · · E, deadlock is possible. (Such synchronization objects may include mutexes, semaphores, condition variables, etc.) We can prevent the deadlock by adding a sixth object F . Whenever a thread wants to ac- quire the synchronization lock for any object A· · · E, it must first acquire the lock for object F . This solution is known as containment: The locks for objects A · · · E are contained within the lock for object F . Compare this scheme with the circular-wait scheme of Section.

Answer:

This is probably not a good solution because it yields too large a scope. It is better to define a locking policy with as narrow a scope as possible.

4) Comparison of the circular-wait scheme with the deadlock-avoidance schemes (like the banker's algorithm) with respect to the following issues.

a. The Runtime overheads

b. System throughput available

Answer:

A deadlock-avoidance scheme tends to gain the run-time overheads because of the cost of keep track of the current resource allocation. However, a deadlock-avoidance scheme allows for more concurrent use of resources than schemes that statically avoid the formation of deadlock. In that sense, a deadlock-avoidance scheme could improve the system throughput.

5) In a real computer system, neither the resources available nor the need of processes for resources are consistent over a very long period. Resources break or are replaced, new processes come and go, new resources are bought, then added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without having the possibility of deadlock), and under what circumstances?

a. Increase Available (new resources added).

b. Decrease Available (resource permanently removed from system)

c. Increase Max for one process (the process needs more resources than allowed, it may want more)

d. Decrease Max for one process (the process decides it does not need that many resources)

e. Increase the number of processes.

f. Decrease the number of processes.

Answer:

a. Increase Available (new resources added) - This could safely be changed without any problems.

b. Decrease Available (resource permanently removed from system)- This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

c. Increase Max for one process (the process needs more resources than allowed, it may want more) - This could have an effect on the system and introduce the possibility of deadlock.

d. Decrease Max for one process (the process decides it does not need that many resources) - This could safely be changed without any problems.

e. Increase the number of processes - This could be allowed assuming that resources were allocated to the new process (es) such that the system does not enter an unsafe state.

f. Decrease the number of processes - This could safely be changed without any problems.

6) Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.

Answer:

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

7) Consider a system consisting of m resources of the same type, being shared by n processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and m resources

b. The sum of all maximum needs is less than m + n

Answer:

Using the terminology of Section 6 we have:

a. $\sum_{i=1}^{n} Max_i < m + n$

b. $Max_i \geq 1$ for all i
Proof: $Need_i = Max_i - Allocation_i$
If there exists a deadlock state then:

c. $\sum_{i=1}^{n} Allocation_i = m$

Use a. to get: $\sum Need_i + \sum Allocation_i = \sum Max_i < m + n$
Use c. to get: $\sum Need_i + m < m + n$
Rewrite to get: $\sum_{i=1}^{n} Need_i < n$

This implies that there exists a process P i such that Need i = 0. Since Max i ≥ 1 it follows that P i has at least one resource that it can release.

Hence the system cannot be in a deadlock state.

8) Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Answer:

The following rule prevents deadlock: when a philosopher makes a request for the first chopstick, do not satisfy the request only if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

9) Consider the same setting as the previous problem. Assume now that each philosopher requires three chopsticks to eat and that resource requests are still issued separately. Describe some simple rules for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Answer:

When a philosopher makes a request for a chopstick, allocate the request if: 1) the phi losopher has two chopsticks and there is at least one chopstick remaining, 2) the philos opher has one chopstick and there is at least two chopsticks remaining, 3) there is at l east one chopstick remaining, and there is at least one philosopher with three chopsticks ,4) the philosopher has no chopsticks, there are two chopsticks remaining, and there is at least one other philosopher with two chopsticks assigned.

10) We can obtain the banker's algorithm for a single resource type from the general banker's algorithm simply by reducing the dimensionality of the various arrays by 1. Show through an example that the multiple- resource-type banker's scheme cannot be implemented by individual application of the single-resource-type scheme to each resource type.

Answer:

Consider a system with resources A, B, and C and processes P 0 , P 1 , P 2 , P 3 , and P 4 with the following values of Allocation:

Table 6.2 System with resources and processes allocation

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

And the following value of Need:

Table 6.3 System with resources and processes needs

Need			
	A	B	C
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

If the value of Available is (2 3 0), we can see that a request from process P 0 for (0 2 0) cannot be satisfied as these lowers Available to (2 1 0) and no process could safely finish.

However, if we were to treat the three resources as three single-resource types of the banker's algorithm, we get the following: For resource A (which we have 2 available),

Table 6.4 System with resources (A, B & C), process's needs and allocation

Table 6.4 (a)

	Allocated	Need
P0	0	7
P1	3	0
P2	3	6
P3	2	0
P4	0	4

Processes could safely finish in the order of P 1 , P 3 , P 4 , P 2 , P 0 . For resource B (which we now have 1 available as 2 were assumed assigned to process P 0),

Table 6.4 (b)

	Allocated	Need
P0	3	2
P1	0	2
P2	0	0
P3	1	1
P4	0	3

Processes could safely finish in the order of P 2 , P 3 , P 1 , P 0 , P 4 . And finally, for resource C (which we have 0 available),

Table 6.4 (c)

	Allocated	Need
P0	0	3
P1	2	0
P2	2	0
P3	1	1
P4	2	1

Processes could safely finish in the order of P 1 , P 2 , P 0 , P 3 , P 4 .

As we can see, if we use the banker's algorithm for multiple resource types, the reque st for resources (0 2 0) from process P 0 is denied as it leaves the system in an uns afe state. However, if we consider the banker's algorithm for the three separate resource

s where we use a single resource type, the request is granted. Therefore, if we have m ultiple resource types, we must use the banker's algorithm for multiple resource types.

Exercise Questions

- With neat diagram explain various layers in Operating System Interface.
- Write a short note on device drivers.
- Describe in details Kernel I/O Subsystem
- Write short note on Direct Memory Access (DMA)
- Describe Device-Independent I/O Software
- Differentiate between Synchronous vs asynchronous I/O
- Explain Kernel I/O Subsystem in short
- Explain in short Physical Organization of CD-ROM
- With neat diagram explain Structure of a hard disk.
- Describe in details the concept of Deadlocks.
- Describe deadlock detection and deadlock prevention.

Reference:

- Operating System Concepts, 8th Edition, by Galvin et al, 2008, Wiley Publications.
- Lecture notes and ppt of Ariel J. Frank, Bar-Ilan University.
- Operating Systems | Internals and Design Principles | by William Stallings, Ninth Edition | By Pearson Publications
- http://www.mathcs.emory.edu
- https://www.geeksforgeeks.org
- https://www.tutorialspoint.com

Unit 7 Memory Management

Objectives:

- To study Fixed-Sized and variable-Sized Partition.
- To study Demand Paging
- To study Dynamic Partitioning.
- To study the concept of Buddy System.
- To study Combined Systems.
- To study Virtual Memory**,** its working and needs.
- To study various types of Page Replacement Methods.

7.1 Introduction

The main memory is divided into two parts in case of a uni-programming system: one part for the program currently being executed and one part for the operating system which include resident monitor, kernel. The "user" part of memory in case of a multiprogramming system must be further subdivided to accommodate multiple processes. The operating system carries out the task of subdivision dynamically which is known as memory management.

In a multiprogramming system effective memory management is vital. One can guess how much of the time all of the processes will be waiting for I/O and the processor will be idle, If only a few processes are in memory. Therefore, to ensure a reasonable supply of ready processes to consume available processor time memory needs to be allocated.

7.2 Contiguous Memory Allocation

A memory allocation method that allocates a si**ngle contiguous section of memory to** a process or a file is called **Contiguous memory allocation**. This method estimates the maximum size, up to what the file or process can grow? And also takes into account the size of the file or a process and also

The operating system allocates sufficient contiguous memory blocks to that file, by taking into account the future growth of the file and its request for memory. The operating system will allocate those many contiguous blocks of memory to that file by considering this future expansion and the file's request for memory.

7.2.1 Memory Allocation

The o**perating system a**nd use**r space** both has to accommodate by the main memory**. N**ow, here various user processes have to accommodate by the user space. Here one should expect that at the same time these various user processes must reside in the main memory.

Now, the problem arises, the available memory space how to allocate to the user processes that are waiting in a ready queue?

The solution is in Contiguous memory allocation, the contiguous memory blocks are allocated to the process according to its requirement, when the process arrives from the ready queue to the main memory for execution. And the memory can be divided either in the fixed-sized partition or in the variable-sized partition to allocate the co**ntiguous sp**ace to user processes.

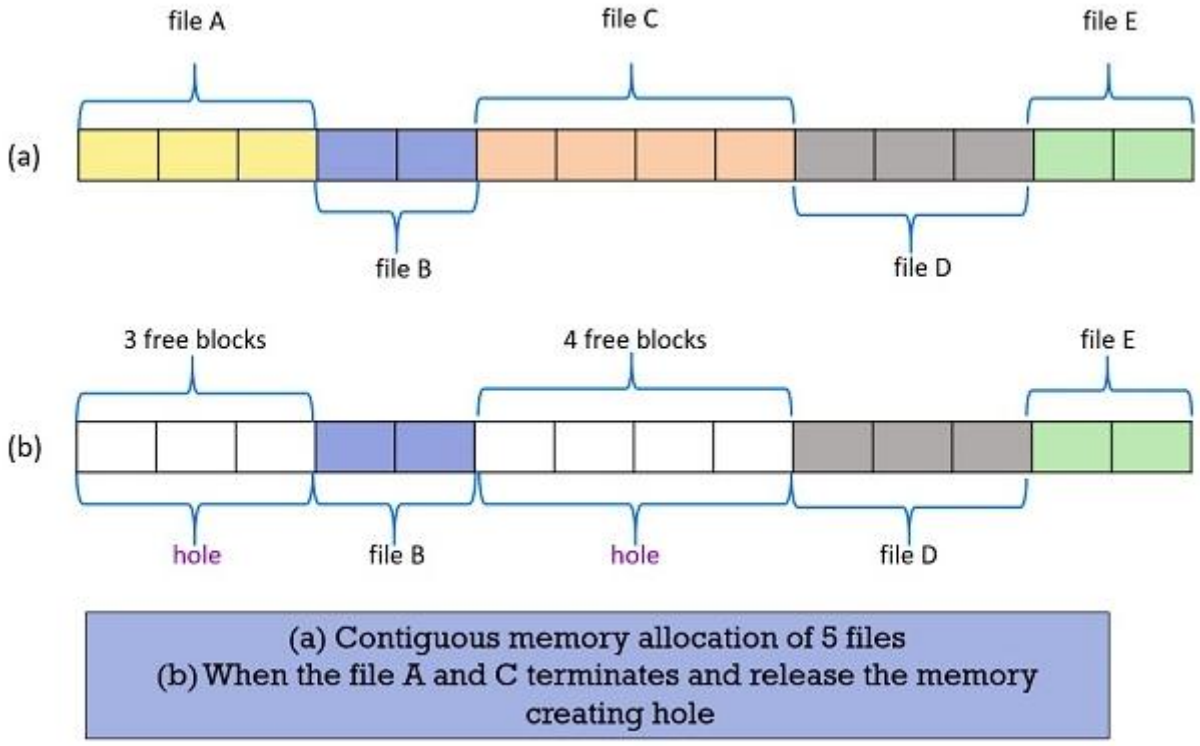


Figure 7.1 Memory allocation

Fixed-Sized Partition: In the fixed-sized partition, the memory is divided into fixed-sized blocks and each block contains exactly one process. But, as the number of the partition will decide the number of so the fixed-sized partition will limit the degree of multiprogramming.

Variable-Size Partition: In the variable size partition method, a table is maintained by operating system, this table containing the information about all memory parts that are still avail**able for t**he processes and all memory parts that are occ**upied by** the processes.

How holes are created in the memory?

At start, for the user processes as a large block, a **hole** the whole memory space is available**.** Eventually, when the processes arrive in the memory, executes, terminates and leaves the memory you will see the set of holes of variable sizes.

Observe carefully in the figure mentioned above, It creates the holes in the memory of variable size when file A and file C release the memory allocated to them.

In case of v**ariable size partition m**ethod, the operating system sees whether it has a memory block of the required size and analyses the memory requirement of the process. It allocates that

memory block to the process, If it finds the match. If not, then it searches the ready queue for the process that has a smaller memory requirement.

Until it cannot satisfy the memory requirement of the next process in the ready queue the operating system allocates the memory to the process. if it does not have a memory block (**hole**) that is large enough to hold that process it stops allocating memory to the process.

for the process it gets spi**lt int**o two parts if the memory block (**hole**) is too l**arge**. One part is returned to the set of holes and other One part of the memory block is allocated to the arrived process. A process terminates and releases the memory allocated to it; the released memory is then placed back to the set of holes. The two h**oles that are adjacent to ea**ch other, in the set of holes, are merged to form one large h**ole.**

At this instance, whether this newly formed free large hole is able to satisfy the memory requirement of any process waiting in the ready queue is checked by the operating system. And the process goes on.

7.2.2 Memory Management

Let us see and discuss how the operating system selects a free hole from the set of holes?

The operating system uses either the bit map to select the hole from the set of holes or the block allocation list.

Block Allocation List

Block allocation list maintains two tables. The entries of the blocks that are allocated to the various files are stored in one table. Whereas the entries of the holes that are free and can be allocated to the process in the waiting queue are contained in another table.

To decide which entries of free blocks among available blocks must be chosen either of these strategies can be used: first-fit, best-fit, worst-fit strategies.

1. First-fit

Here, the searching starts either from where the previous first-fit search has ended or at the beginning of the table. While searching, the first hole that is found to be large enough for a process to accommodate is selected.

2. Best-fit

This method required the sorted list of holes that is the list of free holes to be sorted according to their size. The hole is selected from the list of free holes with the smallest hole which is large enough for the process to accommodate. As it does not allocate a hole of larger size which leaves some amount of memory even after the process accommodates the space this strategy reduces the wastage of memory.

3. Worst-fit

The entire list of free holes is required to be sorted in this method. Here, also the hole which is largest among the free holes is selected. The leftover hole which may be useful for the other process and which is largest one is leaves by this strategy.

4. Bit Map

The bit map method is responsible to preserves or maintain track of the free or allocated block. One block is represented by one bit, bit 0 resembles the free block and bit 1 resembles that the block is allocated to a file or a process.

Figure 7.2 Bit Map

The files or processes to which the specific blocks are allocated are not recorder or stored by this method. Usually, the number of consecutive zeros/free blocks required by a file of process will be searched by implementing the first fit. Having found that much of consecutive zeros it allocates a file or process to those blocks.

As the table of free blocks sorted according to the hole size has to be maintained, hence implementing best-fit or worse-fit will be expensive. Comparatively it is easy to implement the bit map method.

7.2.3 Fragmentation

Fragmentation is classified in two types which are external fragmentation or internal fragmentation. When the free memory blocks available in memory are too small and even non-contiguous is called E**xternal fragmentation**. However, when the process does not fully utilize the memory allocated to it, the int**ernal fragmentation occ**urs.

The problem of external fragmentation has the solution called as m**emory compaction.** Here, all the memory contents are shuffled to the one area of memory thereby creating a large block of memory which can be allocated to one or more new processes. Let us observe the diagram carefully to make large hole the last process C, B, D are moved downwards as shown in following diagram.

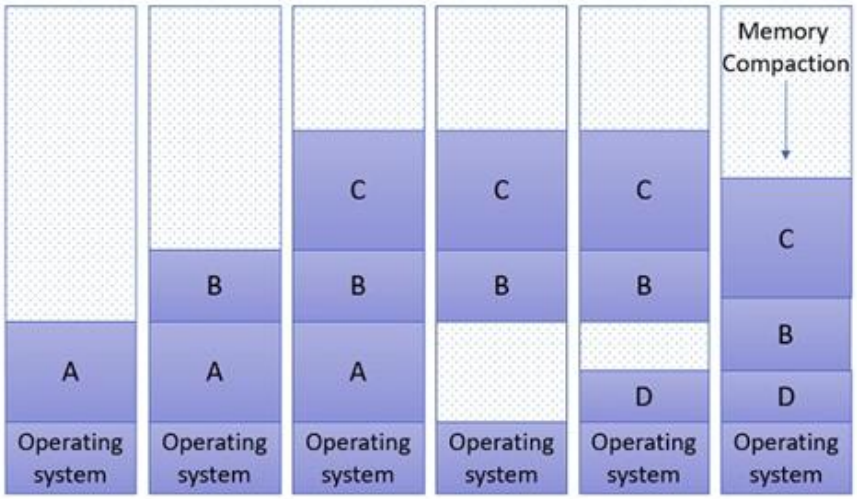


Figure 7.3 Memory compaction

Disadvantages

The mem**ory wastage and** infle**xibility are the** main d**isadvantage o**f contiguous memory allocation**.** Keeping in mind that it will grow during the run **as** the memory is allocated to a file or a process. But many blocks allocated to it remains unutilized until a process or a file grows. And they cannot be allocated to the other process leading to wastage of memory.

It will abort with the message "No disk space" leading to inflexibility if in case, the process or the file grows beyond the expectation i.e., beyond the allocated memory block.

Advantages

The contiguous memory allocation is used to increases the processing speed; this is the main advantage of this. Operating system reads the process memory blocks consecutively and uses the buffered I/O it reduces the head movements and also speed ups the processing.

Key Takeaways

- To perform Memory allocation two schemes can be used, it can be done by variable-sized partition scheme or either by a fixed-sized partition scheme.
- To select a hole from the list of free holes Block allocation list has three methods first-fit, best-fit and worse-fit.
- Bit map has one bit for one memory block and it keeps track of free blocks in memory, bit 0 shows that the block is free and bit 1 show the block is allocated to some file or a process.
- Contiguous memory allocation leads to fragmentation, which is classified as either external or internal.
- Contiguous memory allocation leads to memory wastage and inflexibility. Contiguous memory allocation can enhance processing speed if the operating system uses buffered I/O during processing.

7.3 Fixed Partitioned Memory Management

The function responsible for allocating and managing computer's main memory in operating systems is called Memory Management. To keep track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory are the important Memory Management function.

The Memory Management Techniques are of two types: Contiguous, and Non-Contiguous. In Contiguous Technique, executing process must be loaded entirely in main-memory. Contiguous Technique can be divided into:

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

Fixed Partitioning:

To put more than one processes in the main memory this is the oldest and simplest technique used. In this partitioning, size of each partition may or may not be same but number of partitions (non-overlapping) in RAM is fixed. No spanning is allowed as it is contiguous allocation. Here partition is made before execution or during system configure.

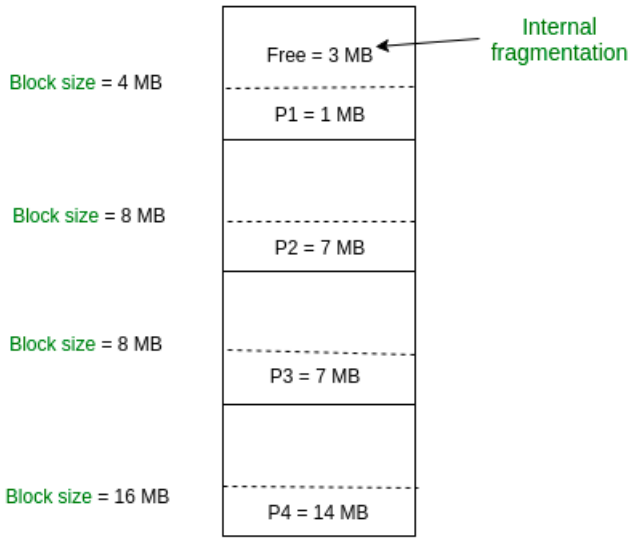


Figure 7.4 Fixed Sized Partitioning

Let us observe in the figure first process is only consuming 1MB out of 4MB in the main memory. Hence, Internal Fragmentation in first block is (4-1) = 3MB. Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

Suppose process P5 of size 7MB comes. Because of contiguous allocation and as spanning is not allowed this process cannot be accommodated in spite of available free space. Hence, 7MB becomes part of External Fragmentation.

Let us discuss the advantages and disadvantages of fixed partitioning.

Advantages of Fixed Partitioning –

1. **Easy to implement:**
   Algorithms needed to implement Fixed Partitioning are easy to implement. Without focussing on the emergence of Internal and External Fragmentation it simply requires putting a process into certain partition.
2. **Little OS overhead:**
   Fixed Partitioning Processing requires indirect computational power and lesser excess.

Disadvantages of Fixed Partitioning

1. **Internal Fragmentation:**

   Main memory use is inefficient. An entire partition is get occupies by any program, no matter how small, which can cause internal fragmentation.

2. **External Fragmentation:**

   To load the processes even though there is space available but not in the contiguous form the total unused space of various partitions cannot be used.

3. **Limit process size:**

   Main Memory cannot be accommodated the process of size greater than size of partition. According to the size of incoming process's size Partition size cannot be varied. Hence, process size of 32MB in above stated example is invalid.

4. **Limitation on Degree of Multiprogramming:**

   In Main Memory partition are made during system configure or before execution. Into the fixed number of partition Main Memory is divided. Suppose if there are n1 partitions in RAM and n2 are the number of processes, then n2<=n1 condition must be fulfilled. In Fixed Partitioning is not allowed and is invalid if Number of processes greater than number of partitions in RAM.

7.4 Dynamic Partitioning

To overcome some of the difficulties with fixed partitioning, an approach known as dynamic partitioning was developed. By more sophisticated memory management techniques this approach has been supplanted. IBM's mainframe operating system was an important operating system that used this technique, OS/MVT (Multiprogramming with a Variable Number of Tasks). The partitions are of variable length and number with dynamic partitioning.

Process is allocated exactly as much memory as it requires and no more, when a process is brought into main memory. Let us understand with an example, using 64 Mbytes of main memory, Initially, main memory is empty, except for the operating system (a) is as shown in Figure 7.5. Consider the first three processes (a, b & c) are loaded in, occupying just enough space for each process (b, c, d) and starting where the operating system ends. At the end of memory this leaves a "hole" which is too small for a fourth process. Which cause, none of the processes in memory is ready at some point. The process 2 (e) is swapped out by operating system, which leaves enough room to load a new process, process 4 (f). Because process 2 is greater than process 4, another small hole is created. Which result, none of the processes in main memory is ready after some time ahead, but process 2, in the Ready-Suspend state, is available. The operating system swaps process 1 out (g) and swaps process 2 back in (h), As there is insufficient room in memory for process 2.

As this example shows, it leads to a situation in which there are a lot of small holes in memory even if this method starts out well. Slowly memory get more and more fragmented as time goes on, and memory utilization declines. This phenomenon is indicating that the memory that is external to all partitions becomes increasingly fragmented and referred to as external fragmentation. Which is in contrast to internal fragmentation. One technique for overcoming external fragmentation is compaction: From time to time, the operating system shifts the processes so that they are contiguous and so that all of the free memory is together in one block. Let us understand with an example, compaction will result in a block of free memory of length 16M as shown in figure. To load in an additional process this may well be sufficient. The difficulty with compaction is that it is a time-consuming procedure and wasteful of processor time. The need for a dynamic relocation capability is implies by compaction. That is, it must be possible to move a program from one region to another in main memory without invalidating the memory references in the program

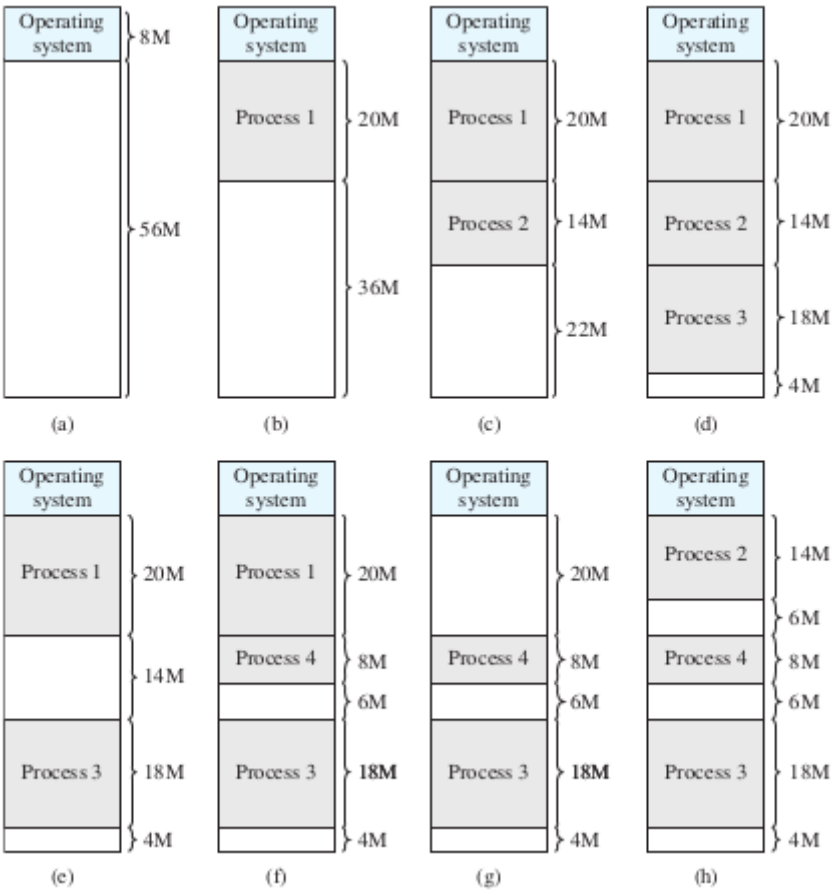


Figure 7.5: Dynamic **Partitioning**

Placement Algorithm, The operating system designer must be clever in deciding how to assign processes to memory or how to plug the holes, Because memory compaction is time consuming. If there is more than one free block of memory of sufficient size and when it is time to load or swap a process into main memory, then the operating system must decide which free block to allocate. The placement algorithms those are classified in three types that might be considered are best-fit, first-fit, and next-fit. All, of course, are limited to choosing among free blocks of main memory that are equal to or larger than the process to be brought in. The block which is closest in size to the request is chosen by Best-fit method. First-fit begins to scan memory from the beginning and chooses the first available block that is large enough. Next-fit chooses the next available block that is large enough and begins to scan memory from the location of the last placement. After a number of placement and swapping-out operations memory configuration example is as shown in Figure. A partition of 14-Mbyte created from the last block that was used, which was of a 22-Mbyte block. The difference between the best-, first-, and next-fit placement algorithms in satisfying a 16-Mbyte allocation request is as shown in figure. To make use of the 18-Mbyte block, leaving a 2-Mbyte fragment, Best-fit will search the entire list of available blocks. Next-fit results in a 20-Mbyte fragment and First-fit results in a 6-Mbyte fragment.

Depending on the exact sequence of process swapping that occurs and the size of those processes, which of these approaches is best will be decided. The first-fit algorithm is best and fastest as well as is the simplest algorithm. The next-fit algorithm tends to produce slightly worse results than the first-fit. The next- fit algorithm will more frequently lead to an allocation from a free block at the end of memory. The result is that the largest block of free memory, which usually appears at the end of the memory space, is quickly broken up into small fragments. Thus, with next-fit more

compaction may be required frequently. On the other hand, let us talk about first-fit algorithm, with small free partitions that need to be searched over on each subsequent first-fit pass this algorithm may litter the front end. The best-fit algorithm as this algorithm looks for the smallest block that will satisfy the requirement; it guarantees that the fragment left behind is as small as possible. Hence the best-fit algorithm is usually the worst performer and it despite its name. Although each memory request always wastes the smallest amount of memory, the result is that main memory is quickly littered by blocks too small to satisfy memory allocation requests. Therefore, As compared to the other algorithms memory compaction must be done more frequently .

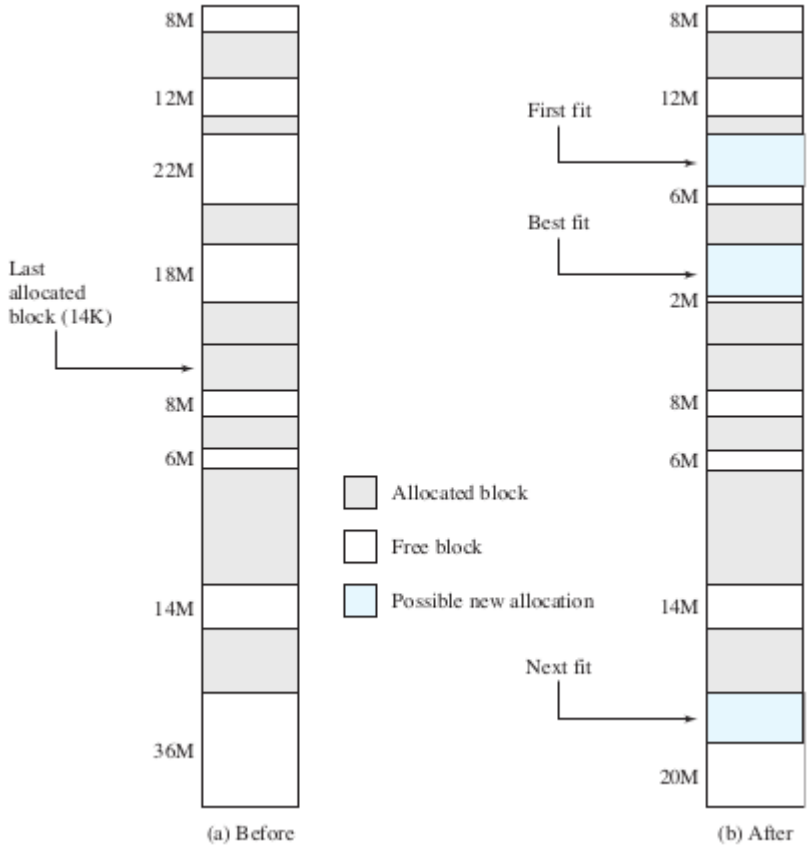


Figure 7.6 : Dynamic **Partitioning**

Replacement Algorithm In a multiprogramming system using dynamic partitioning, a time will come when there is insufficient memory and all of the processes in main memory are in a blocked state, even after compaction, for an additional process. The operating system has to swap one of the processes out of main memory to make room for a new process or for a process in a Ready-Suspend state and in order to avoid wasting processor time waiting for an active process to become unblocked. Hence, which process to replace must choose by the operating system.

7.4.1 Buddy System

Both fixed and dynamic partitioning schemes have drawbacks. A fixed partitioning scheme limits the number of active processes and if there is a poor match between available partition sizes and process sizes this system may use space inefficiently. A dynamic partitioning scheme is more complex to maintain and includes the overhead of compaction.

In a buddy system, memory blocks are available of size 2 K words, L <=K <= U,

where 2 L= smallest size block that is allocated 2 U = largest size block that is allocated; generally, 2U is the size of the entire memory available for allocation To begin, the entire space available for allocation is treated as a single block of size 2 U . If a request of size s such that 2 U-1 < s <= 2 U is made, then the entire block is allocated. Otherwise, the block is split into two equal buddies of size 2 U-1 . If 2 U-2 < s <= 2 U-1 , then the request is allocated to one of the two buddies. Otherwise, one of the buddies is split in half again. This process continues until the smallest block greater than or equal to s is generated and allocated to the request. At any time, the buddy system maintains a list of holes (unallocated blocks) of each size 2 i . A hole may be removed from the (i # 1) list by splitting it in half to create two buddies of size 2 i in the i list. Whenever a pair of buddies on the i list both become unallocated, they are removed from that list and coalesced into a single block on the (i # 1) list. Presented with a request for an allocation of size k such that 2 i-1 < k <= 2 i , the following recursive algorithm is used to find a hole of size 2 i :

```
void get_hole(int i)
{
  if (i == (U + 1)) <failure>;
  if (<i_list empty>) {
      get_hole(i + 1);
      <split hole into buddies>;
      <put buddies on i_list>;
  }
  <take first hole on i_list>;
}
```

Below figure gives an example using a 1-Mbyte initial block. The first request, A, is for 100 Kbytes, for which a 128K block is needed. The initial block is divided into two 512K buddies. The first of these is divided into two 256K buddies, and the first of these is divided into two 128K buddies, one of which is allocated to A. The next request, B, requires a 256K block. Such a block is already available and is allocated. The process continues with splitting and coalescing occurring as needed. Note that when E is released, two 128K buddies are coalesced into a 256K block, which is immediately coalesced with its buddy.

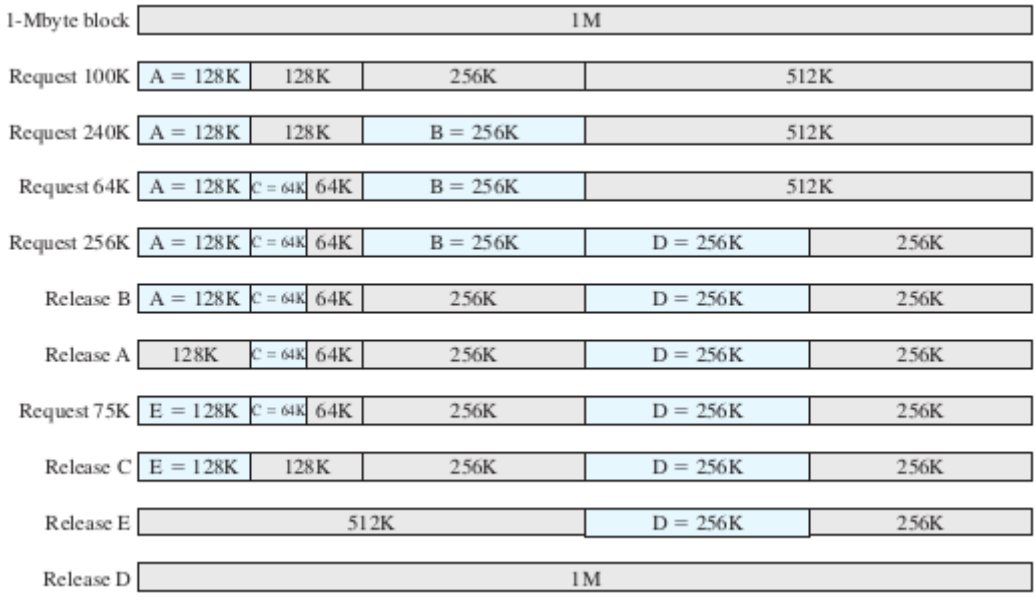


Figure 7.6: Buddy System

Below Figure shows a binary tree representation of the buddy allocation immediately after the Release B request. The leaf nodes represent the current partitioning of the memory. If two buddies are leaf nodes, then at least one must be allocated; otherwise, they would be coalesced into a larger block. The buddy system is a reasonable compromise to overcome the disadvantages of both the fixed and variable partitioning schemes, but in contemporary operating systems, virtual memory based on paging and segmentation is superior. However, the buddy system has found application in parallel systems as an efficient means of allocation and release for parallel programs (e.g., see [JOHN92]).

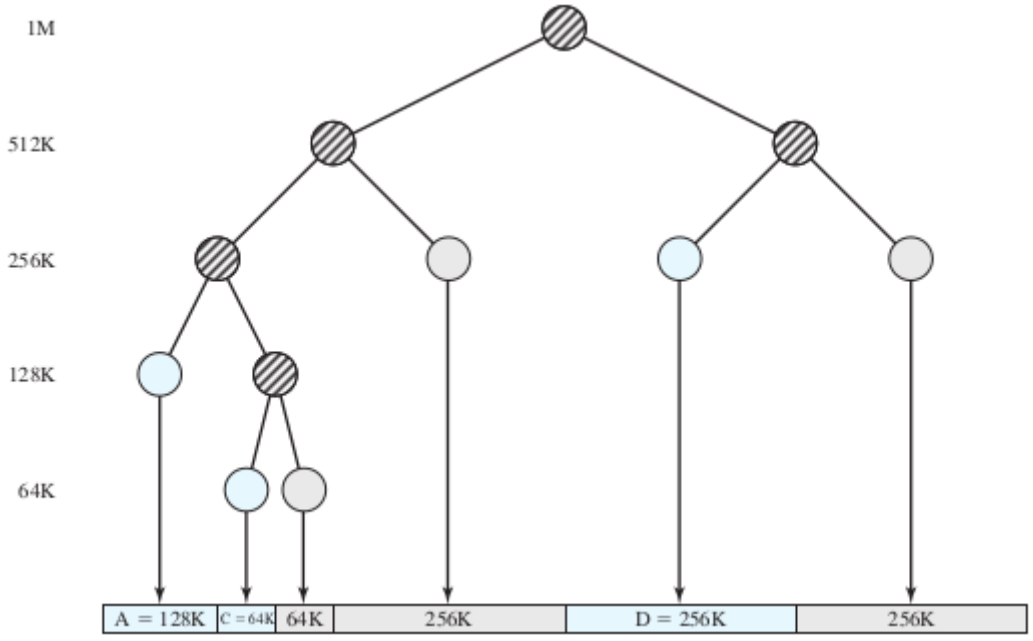


Figure 7.7: Binary tree representation of the buddy allocation

7.4.2 Relocation

We must clear up one loose end, which relates to the placement of processes in memory, Before we consider ways of dealing with the shortcomings of partitioning. One can expect that a process will always be assigned to the same partition, when the fixed partition scheme of Figure is used. That is, whichever partition is selected when a new process is loaded will always be used to swap that process back into memory after it has been swapped out. In that case, a simple relocating loader, can be used: When the process is first loaded, all relative memory references in the code are replaced by absolute main memory addresses, determined by the base address of the loaded process. A process may occupy different partitions during the course of its life In the case of equal-size partitions Figure and in the case of a single process queue for unequal-size partitions Figure. When a process image is first created, it is loaded into some partition in main memory. When it is subsequently swapped back in Later, the process may be swapped out; it may be assigned to a different partition than the last time. The same is true for dynamic partitioning. Observe in Figures and h that process 2 occupies two different regions of memory on the two occasions when it is brought in. Furthermore, processes are shifted while they are in main memory, when compaction is used. Thus, the locations of instructions and data referenced by a process are not fixed. They will change each time a process is swapped in or shifted. A distinction is made among several types of addresses, to solve this problem. Independent of the current assignment of data to memory a logical address is a reference to a memory location; a translation

must be made to a physical address before the memory access can be achieved. A relative address is a particular example of logical address, in which the address is expressed as a location relative to some known point, usually a value in a processor register. A physical address, or absolute address, is an actual location in main memory. Programs that employ relative addresses in memory are loaded using dynamic run-time loading. Typically, all of the memory references in the loaded process are relative to the origin of the program. Hence, at the time of execution of the instruction that contains the reference a hardware mechanism is needed for translating relative addresses to physical main memory addresses. One can observe in Figure shown below, it is showing the way in which this address translation is typically accomplished. A special processor register, sometimes called the base register, is loaded with the starting address in main memory of the program When a process is assigned to the Running state. The "bounds" register that indicates the ending location of the program; these values must be set when the process image is swapped in or when the program is loaded into memory. The relative addresses are encountered during the course of execution of the process. These include the contents of the instruction register, instruction addresses that occur in branch and call instructions, and data addresses that occur in load and store instructions. Each such relative address goes through two steps of manipulation by the processor.

Initially, to produce an absolute address the value in the base register is added to the relative address. Second, the resulting address is compared to the value in the bounds register. The instruction execution may proceed, if the address is within bounds. Otherwise, an interrupt is generated to the operating system, which must respond to the error in some fashion. The scheme of Figure allows programs to be swapped in and out of memory during the course of execution. It also provides a measure of protection: Each process image is isolated by the contents of the base and bounds registers and safe from unwanted accesses by other processes.

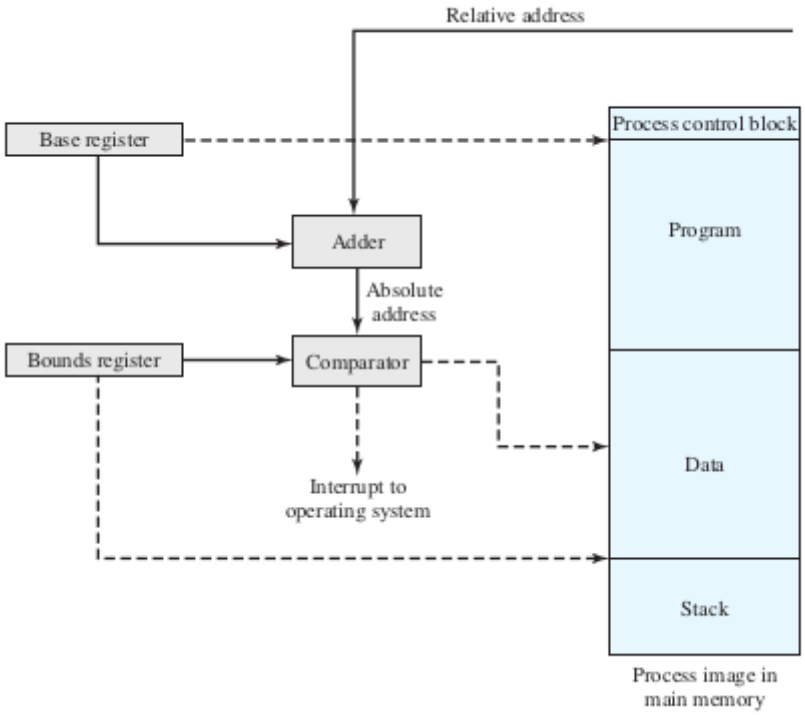


Figure 7.8: Address translation process

7.4.3 Paging

In the use of memory unequal fixed-size and variable-size partitions both methods are inefficient; the former results in internal fragmentation, the latter in external fragmentation. Suppose, however, that main memory is partitioned into equal fixed-size chunks that are relatively small, and that each process is also divided into small fixed-size chunks of the same size. Then the chunks of a process, known as pages, could be assigned to available chunks of memory, known as frames, or page frames. Due to internal fragmentation consisting of only a fraction of the last page of a process, the wasted space in memory for each process. There is no external fragmentation.

Below Figure illustrates the use of pages and frames. Some of the frames in memory are in use and some are free at a given instance of time. The operating system maintains a list of free frames. Process A consists of four pages and is stored on disk. The operating system finds four free frames when it comes time to load this process, and loads the four pages of process A into the four frames Process B, consisting of three pages, and process C, consisting of four pages, are subsequently loaded. Then process B is swapped out of main memory as it gets suspended.

Later, The operating system needs to bring in a new process, process D, which consists of five pages, as all of the processes in main memory are blocked later. Now let us consider, that there are not sufficient unused contiguous frames to hold the process, as mentioned in this example. Does this reason is more enough to prevent the operating system from loading D? As the concept of logical address can be used again and again, so the answer is no. A simple base address register will no longer suffice. A page table is maintained for each process by the operating system. For each page of the process the page table shows the frame location. Each logical address consists of an offset within the page and page number, within the program. Recall that in the case of simple partition, a logical address is the location of a word relative to the beginning of the program; the processor translates that into a physical address. The logical-to-physical address translation is done with paging and is still done by processor hardware. Now how to access the page table of the current process, it must be known to the processor. The processor uses the page table to produce a physical address which is presented with a logical address.

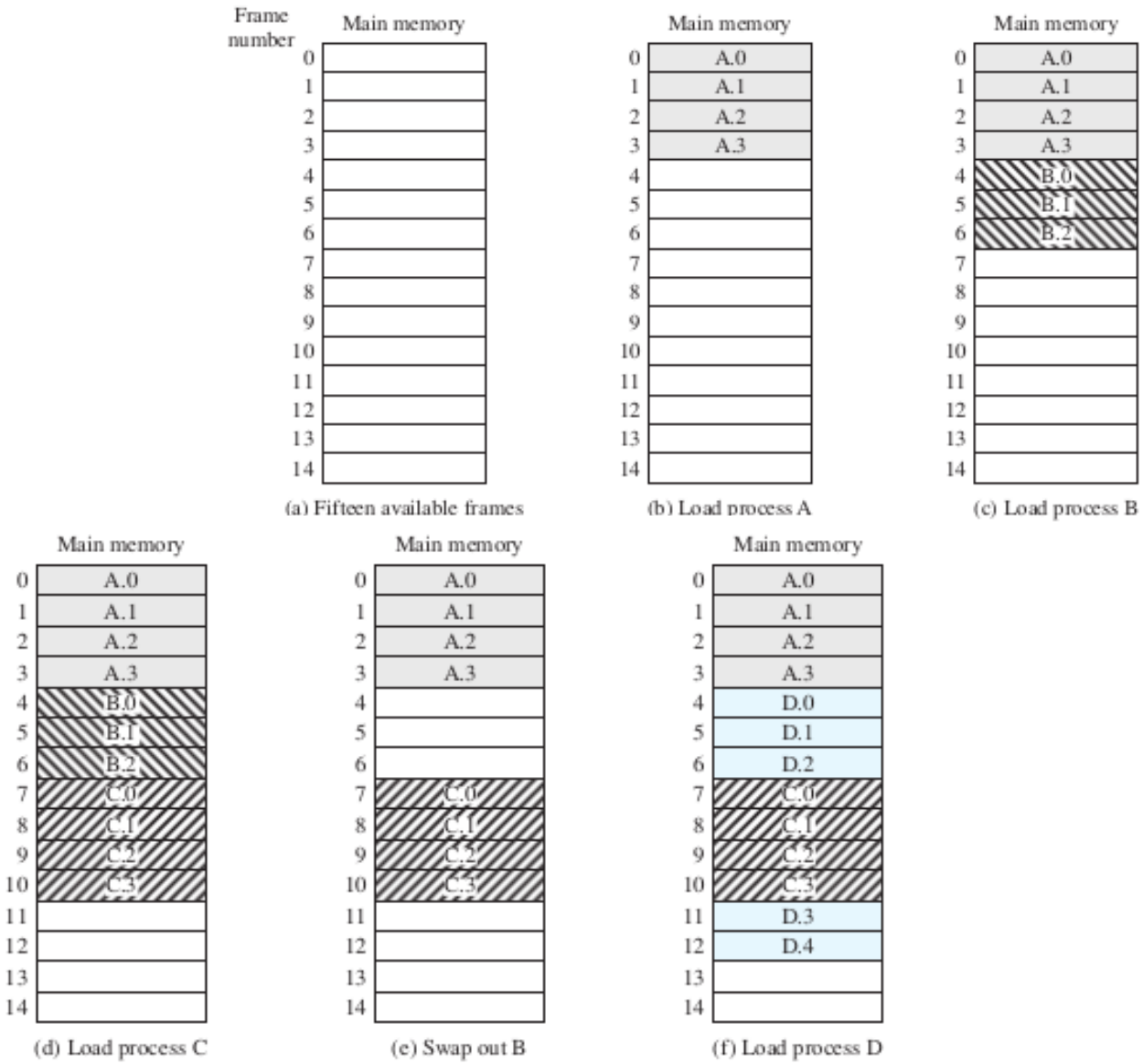


Figure 7.9: Paging

us observe the various page tables as shown in figure, the five pages of process D are loaded into frames 4, 5, 6, 11, and 12. The table is easily indexed by the page number (starting at page 0), Because of a page table contains one entry for each page of the process. Each page table entry contains the number of the frame in main memory, if any, that holds the corresponding page. In main memory the operating system maintains a single free-frame list of all frames that are currently unoccupied and available for pages.

Thus, one can say conclude that simple paging, is similar to fixed partitioning. A program may occupy more than one partition; where as these partitions need not be contiguous, this only what the differences between the paging, and partitions.

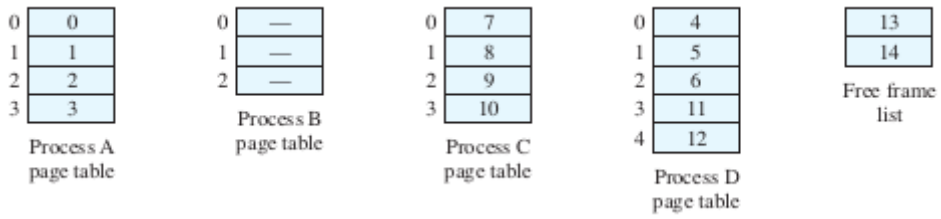


Figure 7.10: Page Table

Let us make this paging scheme convenient, let us dictate that the page size, hence the frame size, must be a power of 2. With the use of a page size that is a power of 2, the logical address, expressed as a page number and offset, are the same and it is easy to demonstrate that the relative address, which is defined with reference to the origin of the program. An example is shown in Above Figure In this example, 16-bit addresses are used, and the page size is 1K # 1024 bytes. Let now represent1502 relative address, in binary form, as 0000010111011110. With a page size of 1K, an offset field of 10 bits is needed, leaving 6 bits for the page number. Thus, a program can consist of a maximum of 2 6 # 64 pages of 1K bytes each. As Figure shows, relative address 1502 corresponds to an offset of 478 (0111011110) on page 1 (000001), which yields the same 16-bit number, 0000010111011110

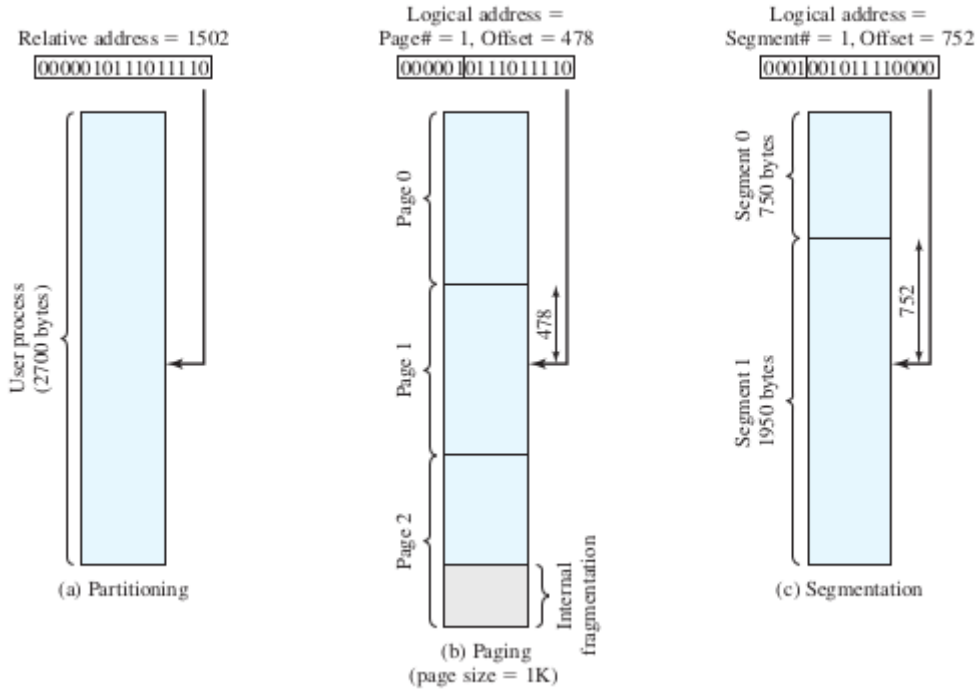


Figure 7.11: Partitioning, Paging and Segmentation

The consequences of using a page size that is a power of 2 are twofold. First, the logical addressing scheme is transparent to the programmer, the assembler, and the linker. Each logical address (page number, offset) of a program is identical to its relative address. Second, it is a relatively easy matter to implement a function in hardware to perform dynamic address translation at run time. Consider an address of n # m bits, where the leftmost n bits are the page number and the rightmost m bits are the offset. In our example (Figure), n # 6 and m # 10. The following steps are needed for address translation:

- The leftmost n bits of the logical address have to be extracted as page number.
- In the process page table to find the frame number, k Use the page number as an index.
- The starting physical address of the frame is k × 2 m , and the physical address of the referenced byte is that number plus the offset. By appending the frame number to the offset physical address is easily constructed, It need not be calculated.

In our example, we have the logical address 0000010111011110, which is page number 1, offset 478. Suppose that this page is residing in main memory frame 6 =binary 000110. Then the physical address is frame number 6, offset 478 = 0001100111011110 (below Figure).

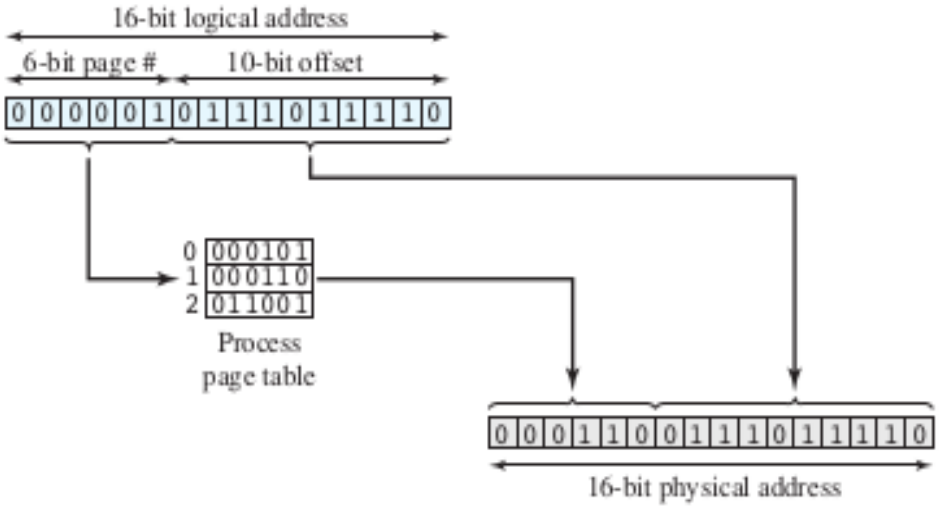


Figure 7.12 (a): Paging

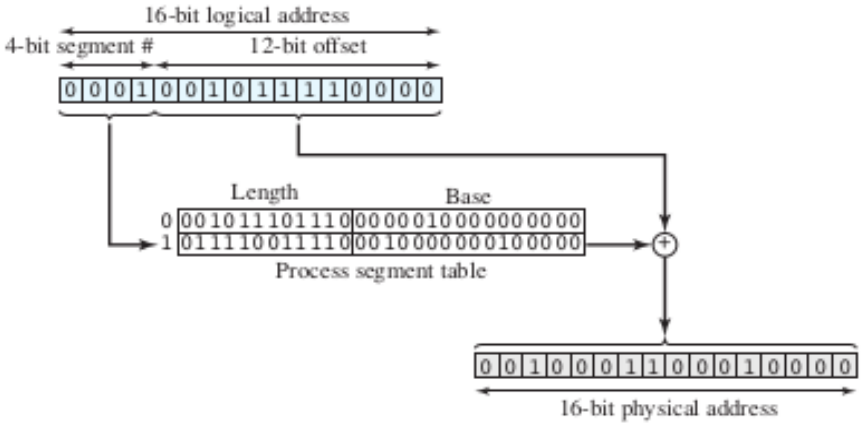


Figure 7.12 (b): Segmentation

Main memory is divided into many small equal-size frames to summarize, with simple paging. Each process is divided into frame-size pages; smaller processes require fewer pages; larger processes require more. When a process is brought in, all of its pages are loaded into available frames, and a page table is set up. This approach solves many of the problems inherent in partitioning.

7.4.4 SEGMENTATION

Using segmentation, a user program can be subdivided, during this the program and its associated data are divided into a number of segments. Although there is a maximum segment length, it is not required that all segments of all programs be of the same length. using segmentation and with paging, a logical address consists of two parts, that is a segment number and an offset.

Segmentation is similar to dynamic partitioning because of the use of unequal-size segments. It would be required that all of a program's segments be loaded into memory for execution, In the absence of an overlay scheme or the use of virtual memory. The difference, compared to dynamic partitioning, a program may occupy more than one partition with segmentation, and these partitions need not be contiguous. Just like dynamic partitioning, it suffers from external fragmentation When Segmentation eliminates internal fragmentation. However, because a process is broken up into a number of smaller pieces, the external fragmentation should be less.

Segmentation is usually visible and is provided as a convenience for organizing programs and data, whereas paging is invisible to the programmer. Naturally, will assign programs and data will be assigned to different segments by the programmer or compiler. The program or data may be further broken down into multiple segments, for purposes of modular programming. The programmer must be aware of the maximum segment size limitation, is the principal inconvenience of this service.

There is no simple relationship between logical addresses and physical addresses, which is another consequence of unequal-size segments. Equivalent to paging, for each process and a list of free blocks of main memory a simple segmentation scheme would make use of a segment table. The starting address in main memory of the corresponding segment should be given by each segment table entry. To assure that invalid addresses are not used, the entry should also provide the length of the segment. The address of its segment table is loaded into a special register used by the memory management hardware, When a process enters the Running state. To understand it better, Let us consider an address of n =m bits, where the leftmost n bits are the segment number and the rightmost m bits

are the offset. In our example (Figure), n = 4 and m = 12. Thus, the maximum segment size is 2 12 = 4096. For address translation the following steps are needed to proceed:

- As the leftmost n bits of the logical address extract the segment number.
- To find the starting physical address of the segment use the segment number as an index into the process segment table.
- If the offset is greater than or equal to the length, the address is invalid, otherwise Compare the offset, expressed in the rightmost m bits, to the length of the segment.
- The sum of the starting physical address of the segment plus the offset is the desired physical address.

Let us consider the example with, the logical address 0001001011110000, which is segment number 1, offset 752. Suppose in main memory starting at physical address 0010000000100000 this segment is residing. Then the physical address is 0010000000100000 + 001011110000 = 0010001100010000 .

A process is divided into a number of segments that need not be of equal size in order to summarize, with simple segmentation. All of its segments are loaded into available regions of memory, and a segment table is set up, When a process is brought in.

7.5 Combined Systems:

Segmented Paging

Many of the operating systems are not using the Pure segmentation as it is not very popular. However, to get the best features out of both the techniques segmentation can be combined with Paging.
The main memory is divided into variable size segments which are further divided into fixed size pages in Segmented Paging.

1.Pages are smaller than segments.

2.Each Segment has a page table which means every program has multiple page tables.

3.The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number → the appropriate Segment Number can be pointed by it.

Page Number →**T**he exact page within the segment can be represented by it.

P**age Offset** → **W**ithin the page frame it can be used as an offset

The various information about every page of the segment contained by each Page table. The information about every segment is contained in Segment Table. Each segment table entry points to a page table entry and every page table entry are mapped to one of the pages within a segment.

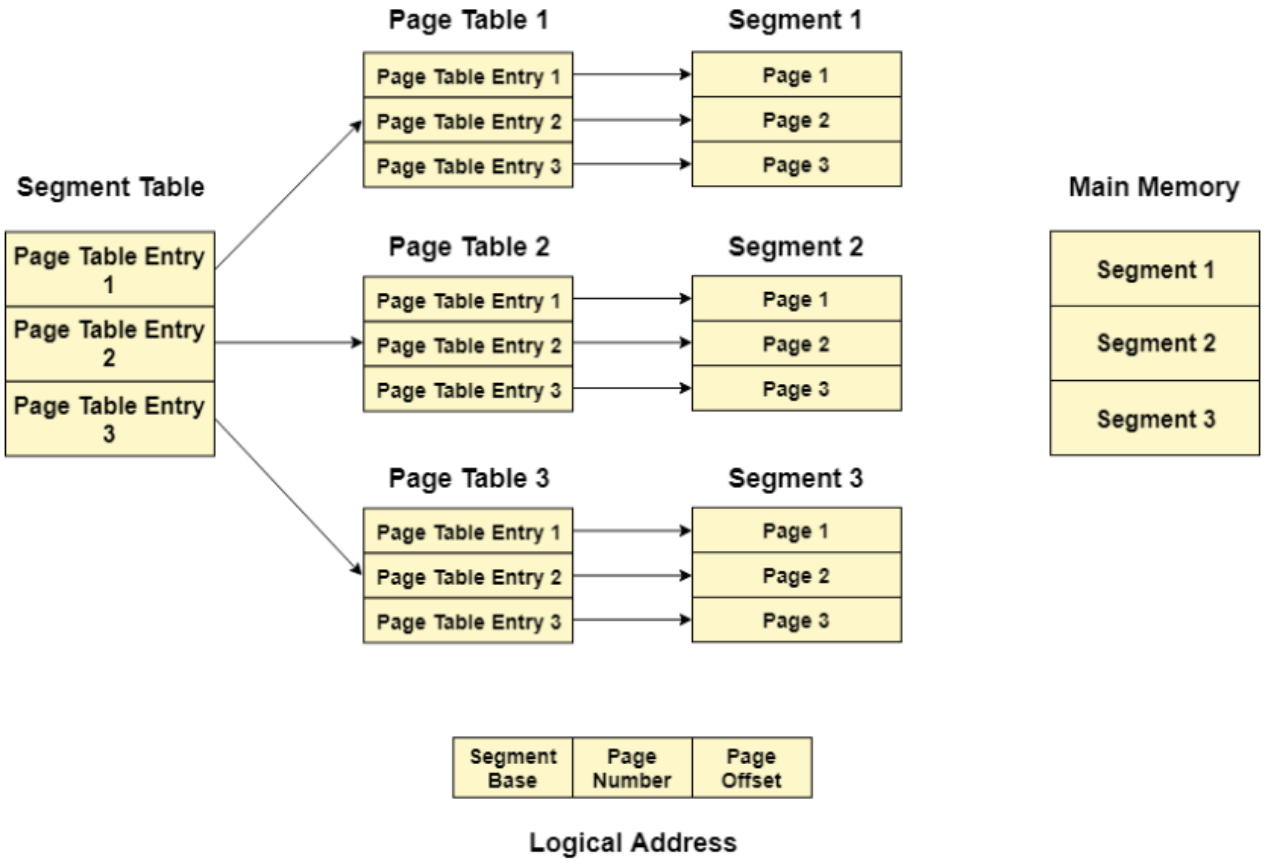


Figure 7.13: Combined Systems

Translation of logical address to physical address

A logical address generated by CPU is divided into two parts: Segment Number and Segment Offset. The segment limit must be greater than the Segment Offset, which is further divided into Page number and Page Offset. The page number is added into the page table base, to map the exact page number in the page table.

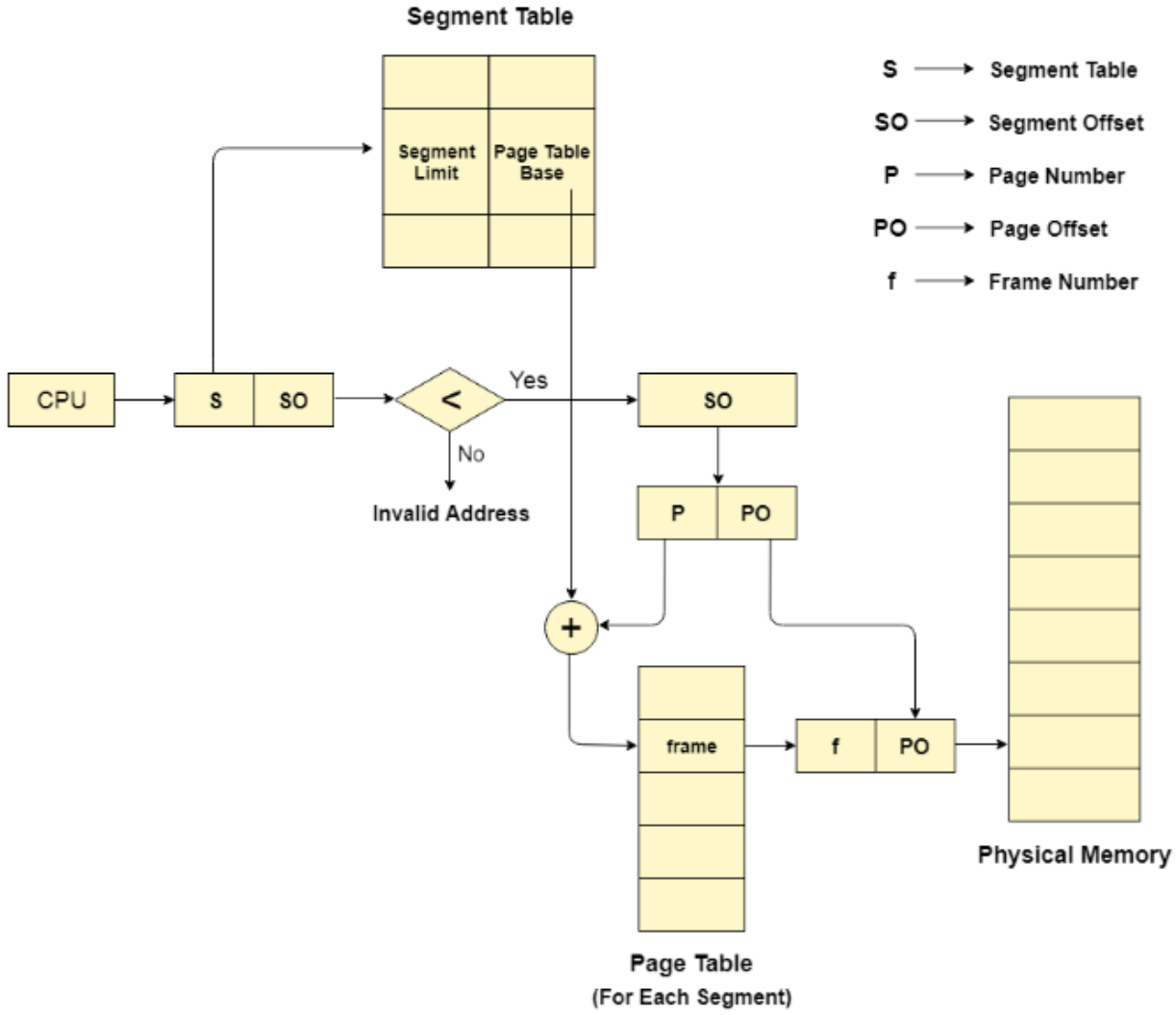


Figure 7.14: Translation of logical address to physical address

To get the desired word in the page of the certain segment of the process the actual frame number with the page offset is mapped to the main memory.

Advantages of Segmented Paging

1. It reduces memory usage.
2. The segment size limits the Page table size.
3. Corresponding to one actual segment, segment table has only one entry.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

Disadvantages of Segmented Paging

1. Internal Fragmentation will be there.
2. As compare to paging the complexity level will be much higher.
3. Page Tables need to be contiguously stored in the memory.

7.6 Virtual memory management system :

An illusion of having a very big main memory offered to user by **Virtual Memory, which** is a storage mechanism. It is done by treating a part of secondary memory as the main memory. With a bigger size than the available main memory the user can store processes, In Virtual memory.

Therefore, the OS loads the various parts of more than one process in the main memory instead of loading one long process in the main memory. With demand paging and demand segmentation, Virtual memory is mostly implemented.

Why Need Virtual Memory?

- Whenever your computer doesn't have space in the physical memory it writes what it needs to remember to the hard disk in a swap file as virtual memory.
- If a computer running Windows needs more memory/RAM, then installed in the system, it uses a small portion of the hard drive for this purpose.

7.6.1 How Virtual Memory Works?

In the modern world, virtual memory has become quite common these days. It is used whenever some pages require to be loaded in the main memory for the execution, and the memory is not available for those many pages.

So, in that case, instead of preventing pages from entering in the main memory, the OS searches for the RAM space that are minimum used in the recent times or that are not referenced into the secondary memory to make the space for the new pages in the main memory.
Let's understand virtual memory management with the help of one example.

For example:

Let's assume that an OS requires 300 MB of memory to store all the running programs. However, there's currently only 50 MB of available physical memory stored on the RAM.

- The OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM) to manage that 250 MB.
- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.
- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

What is Demand Paging?

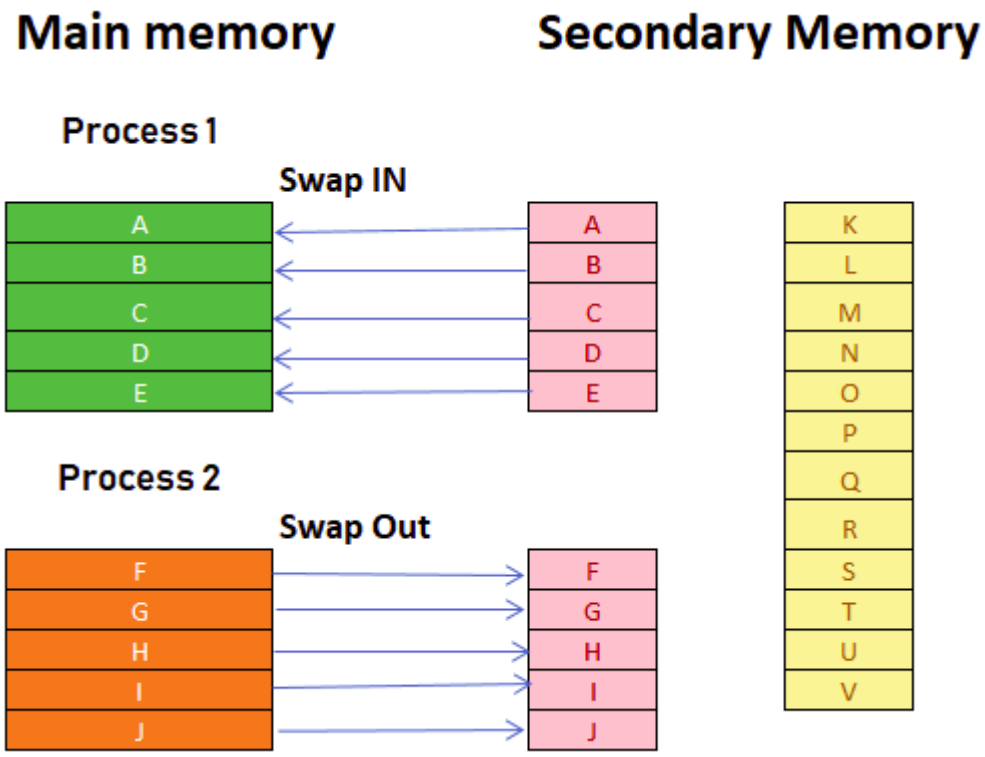


Figure 7.15: Demand Paging

A demand paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.

So, when a context switch occurs, the OS never copy any of the old program's pages from the disk or any of the new program's pages into the main memory. Instead, it will start executing the new program after loading the first page and fetches the program's pages, which are referenced.

During the program execution, if the program references a page that may not be available in the main memory because it was swapped, then the processor considers it as an invalid memory reference. That's because the page fault and transfers send control back from the program to the OS, which demands to store page back into the memory.

7.6.2 Types of Page Replacement Methods

Here, are some important Page replacement methods

- FIFO
- Optimal Algorithm
- LRU Page Replacement

a. FIFO Page Replacement

FIFO (First-in-first-out) is a simple implementation method. In this method, memory selects the page for a replacement that has been in the virtual address of the memory for the longest time.

On a page fault, the frame that has been in memory the longest is replaced.

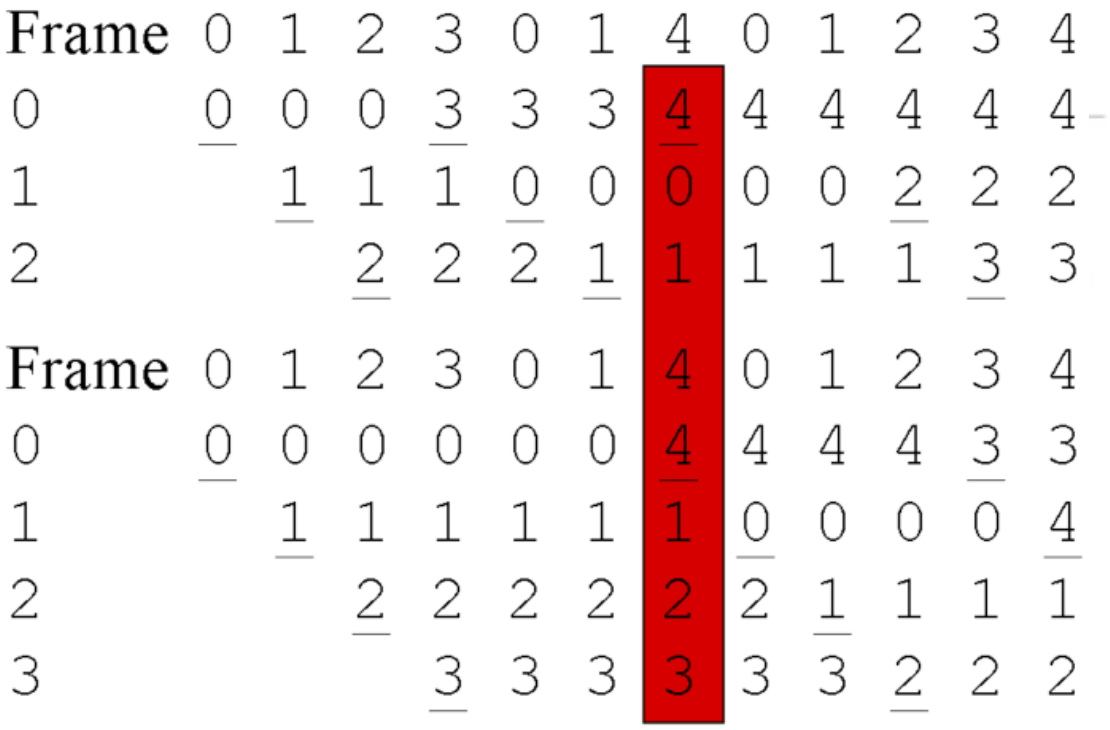


Figure 7.16: FIFO Page Replacement

FIFO is not a stack algorithm. In certain cases, the number of page faults can actually increase when more frames are allocated to the process. In the example below, there are 9-page faults for 3 frames and 10-page faults for 4 frames.

Features:

- Whenever a new page loaded, the page recently comes in the memory is removed. So, it is easy to decide which page requires to be removed as its identification number is always at the FIFO stack.
- The oldest page in the main memory is one that should be selected for replacement first.

b. Optimal Algorithm

The optimal page replacement method selects that page for a replacement for which the time to the next reference is the longest.

The Belady's optimal algorithm cheats. It looks forward in time to see which frame to replace on a page fault. Thus, it is not a real replacement algorithm. It gives us a frame of reference for a given static frame access sequence.

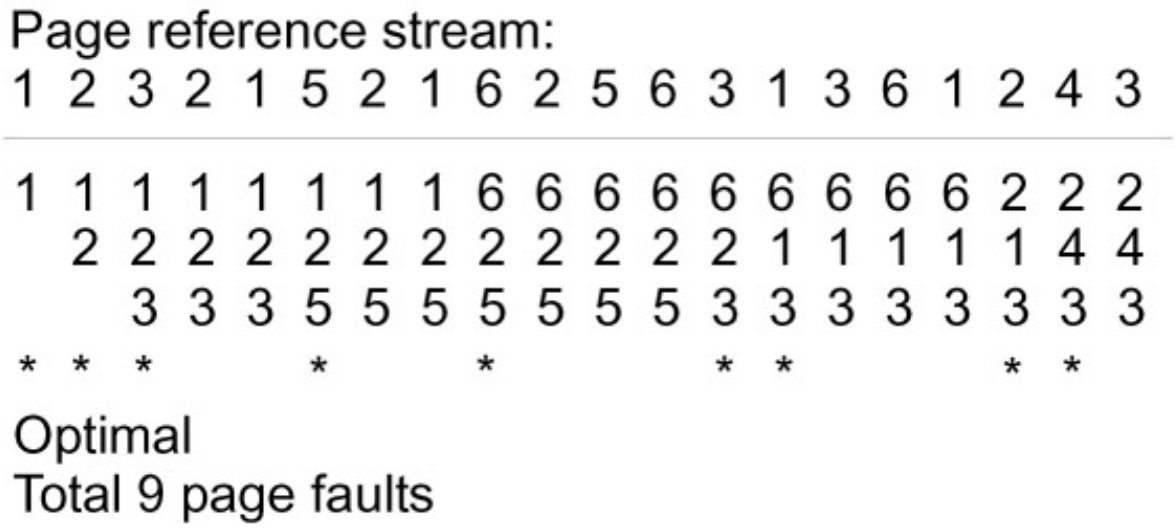


Figure 7.17: Optimal Page Replacement

Features:

- Optimal algorithm results in the fewest number of page faults. This algorithm is difficult to implement.
- An optimal page-replacement algorithm method has the lowest page-fault rate of all algorithms. This algorithm exists and which should be called MIN or OPT
- Replace the page which unlike to use for a longer period of time. It only uses the time when a page needs to be used.

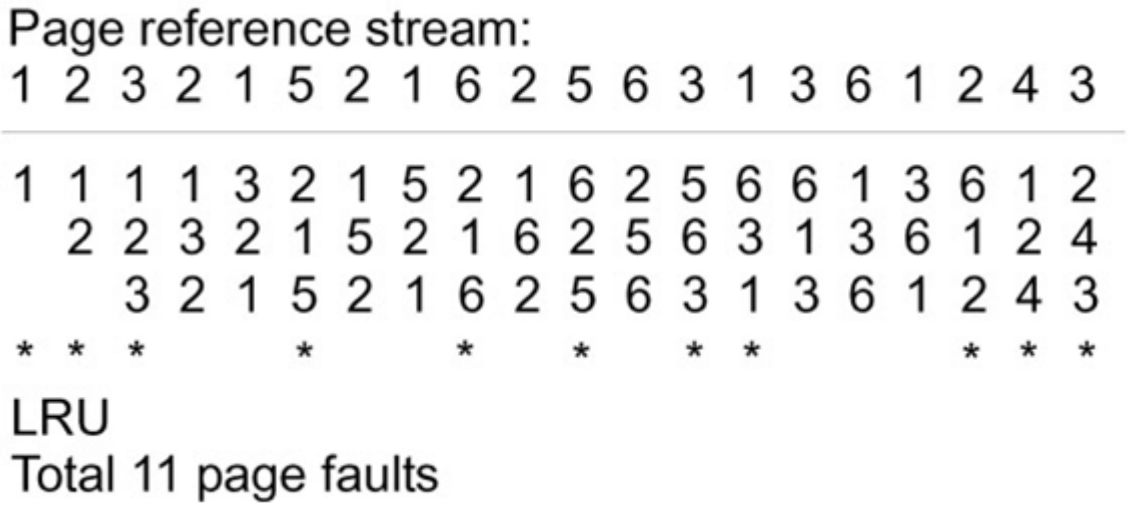
c. LRU Page Replacement

The full form of LRU is the Least Recently Used page. This method helps OS to find page usage over a short period of time. This algorithm should be implemented by associating a counter with an even- page.

How does it work?

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

On a page fault, the frame that was least recently used in replace



On a page fault, the frame that was least recently used in replaced.

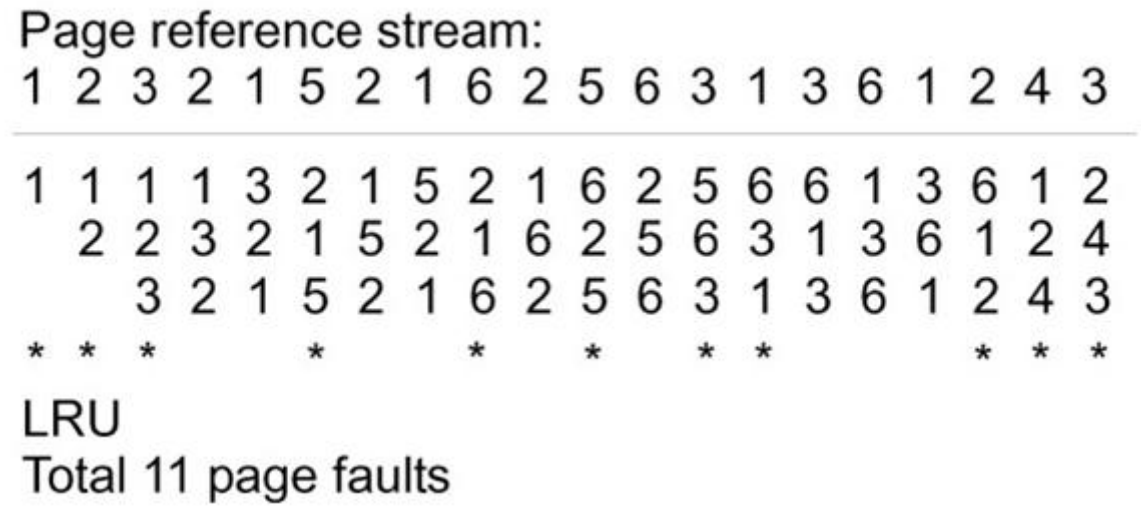


Figure 7.18: LRU Page Replacement

LRU Approximation

Pages with a current copy on disk are first choice for pages to be removed when more memory is needed. To facilitate Page Replacement Algorithms, a table of valid or invalid bits (also called dirty bits) is maintained.

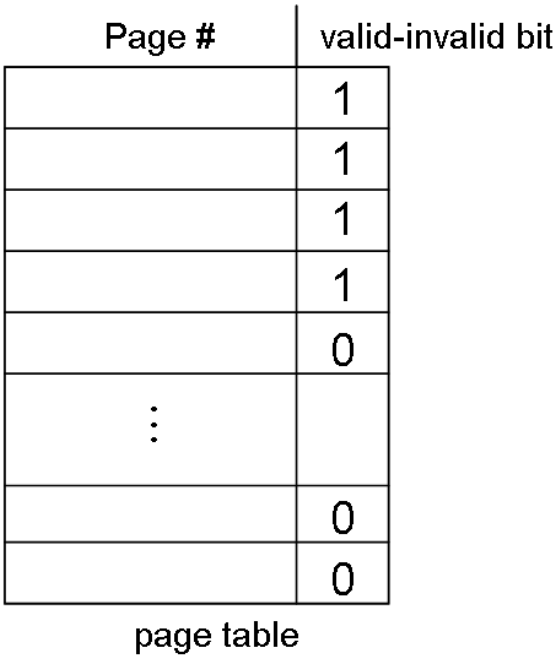


Figure 7.19: Page Table

LRU Page Replacement

With each page table entry, a valid-invalid bit is associated:

- 1 (in-memory)
- 0 (not-in-memory)

Initially valid-invalid but is set to 0 on all entries.

In idle times, dirty frame is copied to disk.

Additional R**eference Bit:** whether the frame was recently referenced.

Features:

- The LRU replacement method has the highest count. This counter is also called aging registers, which specify their age and how much their associated pages should also be referenced.
- The page which hasn't been used for the longest time in the main memory is the one that should be selected for replacement.
- It also keeps a list and replaces pages by looking back into time.

Fault rate

Fault rate is a frequency with which a designed system or component fails. It is expressed in failures per unit of time. It is denoted by the Greek letter λ (lambda).

Advantages of Virtual Memory

Here, are pros/benefits of using Virtual Memory:

- Virtual memory helps to gain speed when only a particular segment of the program is required for the execution of the program.
- It is very helpful in implementing a multiprogramming environment.
- It allows you to run more applications at once.
- It helps you to fit many large programs into smaller programs.
- Common data or code may be shared between memory.
- Process may become even larger than all of the physical memory.
- Data / code should be read from disk whenever required.
- The code can be placed anywhere in physical memory without requiring relocation.
- More processes should be maintained in the main memory, which increases the effective use of CPU.
- Each page is stored on a disk until it is required after that, it will be removed.
- It allows more applications to be run at the same time.
- There is no specific limit on the degree of multiprogramming.
- Large programs should be written, as virtual address space available is more compared to physical memory.

Disadvantages of Virtual Memory

Here, are drawbacks/cons of using virtual memory:

- Applications may run slower if the system is using virtual memory.
- Likely takes more time to switch between applications.
- Offers lesser hard drive space for your use.
- It reduces system stability.
- It allows larger applications to run in systems that don't offer enough physical RAM alone to run them.
- It doesn't offer the same performance as RAM.
- It negatively affects the overall performance of a system.

- Occupy the storage space, which may be used otherwise for long term data storage.

Exercise Questions:

1. What is Fixed-Sized and variable-Sized Partition.
2. What is Demand Paging?
3. Explain in detail Dynamic Partitioning.
4. Explain in detail Fixed Partitioned Memory Management
5. Define the term Paging, Relocation and Segmentation.
6. Explain in detail Buddy System?
7. Write a short note on Combined Systems?
8. **Describe in detail** How Virtual Memory Works
9. Define Virtual Memory. Explain the need of it.
10. Explain various types of Page Replacement Methods.

References:

1. Operating System Concepts, 8th Edition, by Galvin et al, 2008, Wiley Publications.
2. Lecture notes and ppt of Ariel J. Frank, Bar-Ilan University.
3. **Operating Systems | Internals and Design Principles | by William Stallings, Ninth Edition | By Pearson Publications**
4. **Web Portal:** https://www.geeksforgeeks.org

Unit 8 Protection and Security

Objectives:

- To study concept of viruses, worms, and bots.
- To study computer security concepts.
- To study concept of threats, attacks, and assets.
- To study the Means of Authentication.
- To study Authentication and various Authentication methods.

Introduction:

In this age of universal electronic connectivity, viruses and hackers, electronic eavesdropping, and electronic fraud, security has become a central issue. to make the topic of this part of vital interest two trends have come together to make the topic of this part of vital interest. First, he explosive growth in computer systems and their interconnections via networks has increased the dependence of both organizations and individuals on the information stored and communicated using these systems. This, in turn, has led to a height-end awareness of the need to protect data and resources from disclosure, to guarantee the authenticity of data and messages, and to protect systems from network-based attacks. Second, the disciplines of cryptography and computer security have matured, leading to the development of practical, readily available applications to enforce security.

8.1 COMPUTER SECURITY CONCEPTS

Computer Security: To an automated information system the protection afforded in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications)

This definition introduces three key objectives that are at the heart of computer security:

- Confidentiality: Two related concepts are covered in this term:
  - Data confidentiality: Assures that private or confidential information is not made available or disclosed to unauthorized individuals
  - Privacy: It is a Assures that individual's control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.
- Integrity: Two related concepts are covered in this term:
  - Data integrity: Assures that in a specified and authorized manner only the information and programs are changed.
  - System integrity: Assures that a system is free from deliberate or inadvertent unauthorized manipulation of the system, and it performs its intended function in an unimpaired manner.

- Availability: Assures that systems service is not denied to authorized users and work promptly.

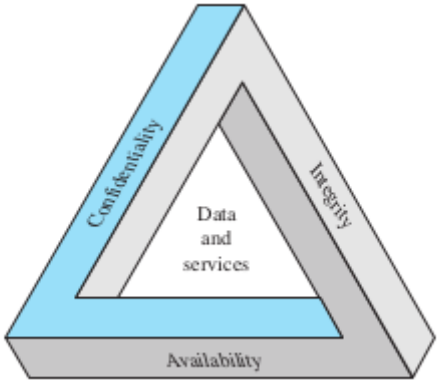


Figure 8.1 The Security Requirements Triad

The three concepts we discussed work together and are often referred to as the CIA triad as shown in Figure 8.1. For both data and for information and computing services the three concepts embody the fundamental security objectives. For example, The Standards for Security Categorization of Federal Information and Information Systems which are also referred as NIST standard FIPS 199, They lists confidentiality, integrity, and availability as the three security objectives for information and for information systems. A useful characterization of these three objectives is provided by FIPS PUB 199 in terms of requirements and the definition of a loss of security in each category:

- Confidentiality: For protecting personal privacy and proprietary information, it Preserving authorized restrictions on information access and disclosure, including means. The unauthorized disclosure of information is referred as A loss of confidentiality.
- Integrity: ensuring information non-repudiation and authenticity, including Guarding against improper information modification or destruction. The unauthorized modification or destruction of information is a loss of integrity.
- Availability: It gives assurance to access and to use of information in timely and reliable manner. The disruption of access to or use of information or an information system is called as loss of availability.

Although to define security objectives to use of the CIA triad is well established, some in the security field feel that additional concepts are needed to present a complete picture. Here following are mentioned two most of the commonly used concepts are as follows:

- Authenticity: The property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator. In other words, is a process to verify that user is genuine and also to ensure that each input arriving at the system came from a trusted source.
- Accountability: The requirement generated by the security goal for actions of an entity to be traced uniquely to that entity. This include and support various activity such as fault isolation, nonrepudiation, deterrence, intrusion detection and prevention, and after-action recovery and legal action. Because truly secure systems aren’t yet an achievable goal, we must be able to

trace a security breach to a responsible party. To permit later forensic analysis Systems must keep records of their activities to trace security breaches or to aid in transaction disputes.

8.2 THREATS, ATTACKS, AND ASSETS

Threats and Attacks

The kinds of attacks listed that result in each consequence, based on RFC 2828, describes four kinds of threat consequences. Unauthorized disclosure is a threat to confidentiality. The attacks that can result in this threat consequence are as mention as following type:

- Exposure: This can be deliberate, as when an insider intentionally releases sensitive information, such as credit card numbers, to an outsider. Exposure can also be caused due to several reasons such as a human errors, hardware, or software error. It results in an entity gaining unauthorized knowledge of sensitive data. There have been numerous instances of this, such as universities accidentally posting student confidential information on the Web.
- Interception: Interception is a common attack in the context of communications. Any device attached to the LAN can receive a copy of packets intended for another device, on a shared local area network (LAN), such as a wireless LAN or a broadcast Ethernet. On the Internet, An unexpected access to e-mail traffic and other data transfers can be gained by determined hacker . The potential for unauthorized access to data is cratered by all of these situations.
- Inference: An example of inference is known as traffic analysis, in which an adversary is able to gain information from observing the pattern of traffic on a network, such as the amount of traffic between particular pairs of hosts on the network. Let us consider one more example in which the inference of detailed information from a database by a user who has only limited access; this is accomplished by repeated queries whose combined results enable inference.
- Intrusion: An example of intrusion is an adversary to sensitive data gaining unauthorized access by overcoming the system's access control protections

To either system integrity or data integrity Deception is a threat. The types of attacks mentioned bellow can cause this threat consequence:

- Masquerade: One example of masquerade is by posing as an authorized user an attempt by an unauthorized user to gain access to a system; this could happen if the unauthorized user has learned another user's logon ID and password. Malicious logic is another example, such as a Trojan horse, It actually gains unauthorized access to system resources or tricks a user into executing other malicious logic but it appears just like to perform a useful or desirable function.

8.2.1 Threats and Assets

The hardware, software, data, and communication lines and networks are the various classifications of assets of a computer system. In this subsection, we briefly describe these four categories and relate these to the concepts of integrity, confidentiality, and availability introduced in above Section

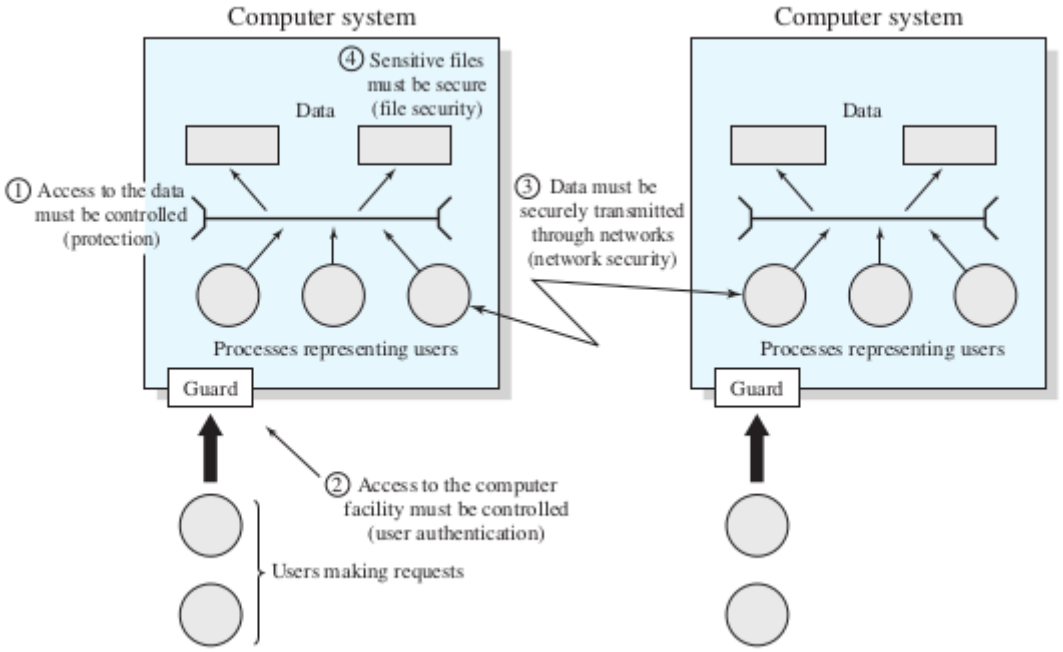


Figure 8.2 Scope of System Security

Table 8.1 Computer and Network Assets, with Examples of Threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is either disabled or not available, thus denying service.		
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended tasks.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Either new files are fabricated or Existing files are modified.
Communication Lines	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable	Messages are read. The traffic pattern of messages are observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

Hardware: The threat to availability is a major threat to computer system hardware. Hardware is the least susceptible to automated controls and is the most vulnerable to attack. Threats consist of both deliberate and accidental damage to equipment as well as theft. The widespread use of LANs along with the proliferation of personal computers and workstations increase the potential

for losses in this area. The loss of confidentiality can be caused due to theft of CD-ROMs and DVDs. To deal with these threats and such types of threats Physical and administrative security measures are needed.

Software: Software comprises application programs, utilities as well as the operating system. An attack on availability is a key threat to software. Software, especially application software, is often easy to delete. To render the software as useless it can also be altered or damaged. Careful software configuration management is a most important task, to maintain high availability making backups of the most recent version of software is required. When a program that still functions but that behaves differently than before, which software is a threat to integrity/authenticity is a more difficult problem to deal with a software modification. Computer viruses and related attacks are some of the examples of this category. The protection against software piracy is a major and final problem. The problem of unauthorized copying of software till has not been solved, Although certain countermeasures are available.

The computing centre professionals or individual concerns of personal computer users are typically concerned for Data Hardware and software security. Data security, which involves files and other forms of data controlled by individuals, groups, and business organizations and which is a much more widespread problem.

The most importantly the security is concerns with respect to data are broad, encompassing secrecy, availability, and integrity. The major concern with availability, is the destruction of data files, which can occur either accidentally or maliciously.

The unauthorized reading of data files or databases is the obvious concern with secrecy, and this area has been the subject of possibly more research and effort than any other area of computer security. A less obvious threat to secrecy involves the analysis of data and manifests itself in the use of so-called statistical databases, which provide summary or aggregate information. Likely, the privacy of the individuals involved does not get threaten by the existence of aggregate information. There is an increasing potential for disclosure of personal information, as the use of statistical databases grows. In essence, through careful analysis characteristics of constituent individuals may be identified. For example, if one table records the aggregate of the incomes of respondents A, B, C, and D and another records the aggregate of the incomes of A, B, C, D, and E, the difference between the two aggregates would be the income of E. So, by the increasing desire to combine data sets, this problem is exacerbated. In many cases, at different levels of aggregation, matching several sets of data for consistency requires access to individual units. Thus, at various stages in the processing of data sets the individual units, which are the subject of privacy concerns, are available.

Lastly, in most installations, data integrity is a major concern. Modifications to data files can have consequences ranging from minor to disastrous.

Communication Lines and Networks: Network security attacks are of two types which are as passive attacks and active attacks. An attack which attempts to learn or make use of information from the system without affecting system resources is called as passive attack. An attack which tries to alter system resources or affect their operation is called as active attack.

The nature of passive attack is transmissions or of eavesdropping on, or monitoring of, or such types of activities. The goal of the attacker is to obtain information that is being transmitted. Some kinds of passive attacks are performed to release of message contents and traffic analysis.

The **release of message contents** is easily understood. In case of passive attack, we need to prevent an opponent from learning the contents of confidential transmissions such as A telephone conversation, an electronic mail message, and a transferred file.

Traffic analysis, is as a second type of passive attack which is subtler. To prevent this, we use the method of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. This technique of masking contents is called as encryption. an opponent might still be able to observe the pattern of these messages even after masking the masking with encryption. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

As Passive attacks do not involve any alteration of the data, they are very difficult to detect. Naturally, neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern because the message traffic is sent and received in an apparently normal fashion. However, by means of encryption, it is feasible to prevent the success of these attacks. Hence rather than detection of passive attack, the emphasis in dealing with passive attacks is on prevention.

Active attacks are carried out with the intention of modification of the data stream or the creation of a false stream. and is classified I four sub types: replay, masquerade, modification of messages, and denial of service.

Replay attack involve the passive capture of a data unit and to produce an unauthorized effect it includes subsequent retransmission.

A **masquerade** is a passive attack and when it takes place one entity pretends to be a different entity. A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Modification of messages In simple words it is the result of altering some portion of a legitimate message, or that messages are delayed or reordered, to produce an unauthorized effect. For example, a message stating "Allow John Smith to read confidential file accounts" is modified to say "Allow Fred Brown to read confidential file accounts."

The **denial of service** is a type of attack which actually prevents or constrains the normal use or management of communications services or amenities. With some specific target in mind this attack may be carried out; Let us understand with an example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). The disruption of an

entire network is another form of service denial, Which can be achieved either by disabling the network or by overloading it with messages so as to degrade performance.

The characteristics of Active attacks are exactly opposite of passive attacks. However, it's very difficult to detect passive attacks, but measures are available to prevent their success. But to prevent these attacks we require physical protection of all communications facilities and paths at all times, and hence it is quite difficult to prevent active attacks absolutely. As an alternative, it is better to detect them and to recover from any disruption or delays caused by them. It may also contribute to prevention, as the detection has a deterrent effect.

8.2.2 INTRUDERS

One of the two most publicized threats to security is the intruder (the other is viruses), often referred to as a hacker or cracker. According to study of intrusion, they are classified in three different classes:

- **Masquerader**: Is a person who is not authorized to use the computer. And to exploit a legitimate user's account he penetrates a system's access controls.
- **Misfeasor**: A authentic user who accesses data, programs, or resources, and who is authorized for such access but misuses his or her privileges, or for which such access is not authorized.
- **Clandestine user**: Is a person or individual who take hold of supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

The masquerader is likely to be an outsider; the misfeasance generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks may be small (beginning) attack or serious attack. When there are many people who simply wish to explore internets and see what is out there is come under beginning attack. On the other hand, an individual who are attempting to perform unauthorized modifications to data, to read privileged data, or disrupt the system are serious attacks.

lists the following examples of intrusion:

- To compromise a remote root of an e-mail server
- Defacing a Web server
- Guessing and cracking passwords
- Creating a Copy of a database comprising credit card numbers
- Accessing the data without authorization such as sensitive data, including payroll records and medical information,
- Running a packet sniffer on a workstation to capture usernames and passwords Using a permission error on an anonymous FTP server to distribute pirated software and music files
- Gaining internal network access Posing as an executive and dialling into an unsecured modem, calling the help desk, resetting the executive's e-mail password, and learning the new password
- Without permission using an unattended, logged-in workstation.

Intruder Behaviour Patterns

To exploit newly discovered weaknesses and to evade detection and countermeasures, the behaviour patterns and techniques of intruders are regularly shifting. Intruders classically trail one of a number of recognizable behaviour patterns, and these patterns typically differ from those of ordinary users. In the following, we look at three broad examples of intruder behaviour patterns to give the reader some feel for the challenge facing the security administrator. The Table mentioned bellow summarizes the behaviour patterns.

Hackers Usually, are those persons who hack into computers do so for the excitement of it or for status. The hacking community is a strong meritocracy in which status is determined by level of competence. Hence, attackers frequently look for targets of opportunity and then share the information with others. There are various examples which describe a break-in at a large financial institution. The corporate network which are running unprotected services, some of which were not even needed, The intruder took advantage of such opportunities. In this case, the key to the break-in was the pc Anywhere application. The manufacturer, Symantec, advertises this program as a remote-control solution that enables secure connection to remote devices. But the attacker had an easy time gaining access to pc Anywhere; the administrator used the same three-letter username and password for the program. In this scenario mentioned above, there was no intrusion detection system on the 700-node corporate network. The intruder was only discovered when a vice president walked into her office and saw the cursor moving files around on her Windows workstation.

Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However, there is no way in advance to know whether an intruder will be benign or malign. Consequently, even for systems with no particularly sensitive resources, there is a motivation to control this problem.

Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs), of the type, are designed to counter this type of hacker threat. In addition to using such systems, organizations can consider restricting remote logons to specific IP addresses and/or use virtual private network technology.

Table 8.2. Some Examples of Intruder Patterns of Behaviour

(a) Hacker

1. Select the target using IP lookup tools such as NSLookup, Dig, and others.
2. Map network for accessible services using tools such as NMAP.
3. Identify potentially vulnerable services (in this case, pcAnywhere).
4. Brute force (guess) pcAnywhere password.
5. Install remote administration tool called DameWare.
6. Wait for administrator to log on and capture his password.
7. Use that password to access remainder of network.

(b) Criminal Enterprise

1. Act quickly and precisely to make their activities harder to detect.
2. Exploit perimeter through vulnerable ports.
3. Use Trojan horses (hidden software) to leave back doors for reentry.
4. Use sniffers to capture passwords.
5. Do not stick around until noticed.
6. Make few or no mistakes.

(c) Internal Threat

1. Create network accounts for themselves and their friends.
2. Access accounts and applications they wouldn't normally use for their daily jobs.
3. E-mail former and prospective employers.
4. Conduct furtive instant-messaging chats.
5. Visit Web sites that cater to disgruntled employees, such as f'dcompany.com.
6. Perform large downloads and file copying.
7. Access the network during off hours.

One of the results of the growing awareness of the intruder problem has been the establishment of a number of computer emergency response teams (CERTs). These cooperative ventures collect information about system vulnerabilities and disseminate it to systems managers. Hackers also routinely CERT reports. Thus, it is important for system administrators to quickly insert all software patches to discovered vulnerabilities. Unfortunately, given the complexity of many IT systems and the rate at which patches are released, this is increasingly difficult to achieve without automated updating. Even then, there are problems caused by incompatibilities resulting from the updated software (hence the need for multiple layers of defence in managing security threats to IT systems).

A widespread and common threat to Internet-based systems is Criminals Organized groups of hackers. Hackers of these groups may be government servant or employ of a corporation or but often are loosely affiliated gangs of hackers. Typically, these gangs are young, often Eastern European, Russian, or southeast Asian hackers who do business on the Web. To trade tips and data and coordinate attacks they meet in underground forums with names like DarkMarket.org and theftservices.com. A credit card file at an e-commerce server is a common target of hackers group. Attackers attempt to gain root access. To purchase expensive items the card numbers are used by organized crime gangs and are then posted to carder sites, where others can access and use the account numbers; this obscures usage patterns and complicates investigation.

Criminal hackers usually have specific targets, or at least classes of targets in mind Whereas traditional hackers look for targets of opportunity. The attacker acts quickly once a site is penetrated, hackers gather up as much valuable information as possible and exiting.

For these types of attackers IDSs and IPSs can also be used but because of the quick in-and-out nature of the attack they may be less effective. Hence it is expected to use database encryption for

sensitive customer information of e-commerce sites, especially for credit cards. The e-commerce organization should make use of a dedicated server for hosted e-commerce sites which are provided by an outsider service and should closely monitor the provider's security services.

Insider Attacks Insider attacks are among the most difficult to detect and prevent. The structure and content of corporate databases can be easily accessed by employees as they already have access to and knowledge of it. By revenge or simply a feeling of entitlement, insider attacks can be motivated. To understand this let us discuss the example of the former is the case of Kenneth Patterson, fired from his position as data communications manager for American Eagle Outfitters. Patterson disabled the company's ability to process credit card purchases during five days of the holiday season of 2002. There have always been many employees who felt entitled to take extra office supplies for home use as for a sense of privilege, but this now extends to corporate data. Let us take an example of that is a vice president of sales for a stock analysis firm who quit to go to a competitor. She copied the customer database to take with her, before she left. The offender reported feeling no animus toward her former employee; she simply wanted the data because it would be useful to her.

Although IDS and IPS facilities can be useful in countering insider attacks, other more direct approaches are of higher priority. Examples include the following:

- Enforce minimum pleasure, only permitting access to the resources employees need to do their job.
- To see what command, they are entering and what user access, one need to set logs.
- Protect sensitive resources with strong authentication.
- One need to delete employee's computer and network access upon termination.
- Upon termination, before reissuing computer hard drive make a mirror image of employee's hard drive. If your company information turns up at a competitor, then that evidence might be needed.

Intrusion Techniques

To gain access to a system or to increase the range of privileges accessible on a system is the initial objective of the intruder. Most initial attacks use system or software vulnerabilities that allow a user to execute code that opens a back door into the system. By exploiting attacks such as buffer overflows on a program that runs with certain privileges, intruders can get access to a system.

Simultaneously, the information that should have been protected is tried to be acquire by intruder. Sometime, user password is contained in this information. An intruder can log in to a system and exercise all the privileges accorded to the legitimate user with knowledge of some other user's password.

8.2.3 VIRUSES, WORMS, AND BOTS

Viruses

A piece of code that can "infect" other programs by modifying them is called computer virus; the modification includes injecting the original program with a routine to make copies of the virus program. These infected copies and modification then go on to infect other programs.

Biological viruses are tiny scraps of genetic code—DNA or RNA—that can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. A computer virus carries in its instructional code the recipe for making perfect copies of itself, Just like counterpart of biological virus. In a program on a computer the typical virus gets embedded. A fresh copy of the virus passes into the new program, whenever the infected computer comes into contact with an uninfected piece of software, Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network. The ability to access system services and applications on other computers provides a perfect culture for the spread of a virus in a network environment.

The Nature of Viruses A virus can perform any action that other programs can do. It attaches itself to another program and executes secretly when the host program is run, this is only difference in virus and other program. When a virus starts its execution, by the privileges of the current user it can perform any function that is allowed, such as erasing files and programs.

A computer virus has three parts:

- Infection mechanism: It is a method by which a virus spreads, enabling it to replicate. The mechanism is also referred to as the infection vector.
- Trigger: The event or condition that determines when the payload is activated or delivered.
- Payload: What the virus does, besides spreading. The payload may involve damage or may involve benign but noticeable activity.

Following are the four phases in which a typical virus may goes through during its life cycle:

- Dormant phase: The virus is idle. During this phase by some event the virus will ultimately get activated, various event such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- Propagation phase: In this phase virus starts propagation by placing an identical copy of itself into other programs or into certain system areas on the disk. a clone of the virus is now contained by each infected program, which will itself enter a propagation phase.
- Triggering phase: In this phase virus get activated. It gets activate to perform the function for which it was intended. By a variety of system events, the triggering phase can be caused as with the dormant phase. The Event may include a count of the number of times that this copy of the virus has made copies of itself.
- Execution phase: The function is performed, which may be harmless, such as a message on the screen, or it may be damaging as well, such as the destruction of programs and data files.

Specific to a particular operating system and, in some cases, specific to a particular hardware platform most viruses carry out their work. Hence, one can say to take advantage of the details and weaknesses of particular systems, they are designed.

Virus Structure By prepended or postponed a virus can be embedded to an executable program, or it can be embedded in various fashion as well. When the infected program is invoked, will first execute the virus code and then execute the original code of the program.

Figure 8.3 describe a very general representation of virus structure. In the given example, the virus code V, is prepended to programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program.

The infected program starts with the virus code and its execution flow or works as follows. The first line of code is a jump to the main virus program. To determine whether or not a potential victim program has already been infected with this virus, the second line is a special marker that is used by the virus. Control is immediately transferred to the main virus program, when the program is invoked. In order to infect uninfected executable files, The virus program may first seek out uninfected files. Usually detrimental to the system Next, the virus may perform some action. The various action could be performed by the virus every time the program is invoked, or it could be triggered only under certain conditions just like a logic bomb. Finally, the original program gains the control from virus. A user is unlikely to notice any difference between the execution of an infected and an uninfected program, If the infection phase of the program is reasonably rapid.

Just because an infected version of a program is longer than the corresponding uninfected one, A virus such as the one just described is easily detected. A technique to prevent such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. Figure 8.4 describes in general terms the logic required. The important lines in this virus are numbered. We assume that program P 1 is infected with the virus CV. The control passes to its virus, when this program is invoked, which performs the following steps:

1. The virus first finds the uninfected files and then for each uninfected file P 2 that is found first compresses that file to produce P' 2, which is shorter than the original program by the size of the virus.
2. the compressed program is get prepended by a copy of the virus.
3. The compressed version of the original infected program, P 9 1 , is uncompressed.
4. The uncompressed original program is executed.

```
    program V :=

{goto main;
     1234567;

    subroutine infect-executable :=
         {loop:
         file := get-random-executable-file;
         if (first-line-of-file = 1234567)
            then goto loop
            else prepend V to file; }

    subroutine do-damage :=
         {whatever damage is to be done}

    subroutine trigger-pulled :=
         {return true if some condition holds}

main:    main-program :=
         {infect-executable;
         if trigger-pulled then do-damage;
         goto next;}

next:

}
```

Figure 8.3. A Simple Virus

```
program CV :=

{goto main;
    01234567;

   subroutine infect-executable :=
         {loop:
              file := get-random-executable-file;
         if (first-line-of-file = 01234567) then goto loop;
       (1)    compress file;
       (2)    prepend CV to file;
      }

main:   main-program :=
          {if ask-permission then infect-executable;
       (3)     uncompress rest-of-file;
       (4)     run uncompressed file;}
       }
```

Figure 8.4. Logic for a Compression Virus

In the example mentioned here one can observe clearly, the virus is doing only the propagation. As the virus may include a logic bomb.

Initial Infection By infecting a single program once a virus has gained entry to a system, it is in a position to potentially infect some or all other executable files on that system when the infected program executes. Thus, by preventing the virus from gaining entry in the first place, viral infection can be completely prevented. Inappropriately, because a virus can be part of any program outside a system, hence prevention is extraordinarily difficult. Thus, one is vulnerable unless and until one is content to take an absolutely bare piece of iron and write all one's own system and application programs. By denying normal users right to modify programs on the system, many forms of infection can also be blocked.

A key reason why traditional machine code-based viruses spread rapidly on these systems is the lack of access controls on early PCs. In contrast, while it is easy enough to write a machine code virus for UNIX systems, because the existence of access controls on these systems prevented effective propagation of the virus, they were almost never seen in practice. AS modern PC operating systems have more effective access controls, hence traditional machine code-based viruses are now less prevalent. However, virus creators have found other avenues, such as macro and e-mail viruses, as discussed subsequently.

Viruses Classification Since viruses first appeared, there is a continuous arms competition between virus writers and writers of antivirus software. Newer types of viruses are developed As effective countermeasures are developed for existing types of viruses, There is no simple or universally agreed upon classification scheme for viruses, In this section, we follow [AYCO06] and classify viruses along two orthogonal axes: the type of target the virus tries to infect and the method the virus uses to conceal itself from detection by users and antivirus software.

A virus **classification by target** includes the following categories:

- **Boot sector infector**: Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus
- **File infector**: Infects files that the operating system or shell consider to be executable.
- **Macro virus**: Infects files with macro code that is interpreted by an application

A virus classification by concealment strategy includes the following categories:

- **Encrypted virus**: A typical approach is as follows. A portion of the virus creates a random encryption key and encrypts the remainder of the virus. The key is stored with the virus. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected. Because the bulk of the virus is encrypted with a different key for each instance, there is no constant bit pattern to observe.
- **Stealth virus**: A form of virus explicitly designed to hide itself from detection by antivirus software. Thus, the entire virus, not just a payload, is hidden.
- **Polymorphic virus**: A virus that mutates with every infection, making detection by the "signature" of the virus impossible.

- **Metamorphic virus**: As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses may change their behaviour as well as their appearance.

One example of a stealth virus was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, stealth is not a term that applies to a virus as such but, rather, refers to a technique used by a virus to evade detection.

Replication copies created by a polymorphic virus are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the purpose is to defeat programs that scan for viruses. In such situation, with each copy the "signature" of the virus will vary. The virus may randomly insert superfluous instructions or interchange the order of independent instructions in order to achieve this variation. Encryption is a more effective approach to use for such cases. The strategy of the encryption virus is followed. The part of the virus that is responsible for performing encryption/decryption and generating keys is referred to as the mutation engine. With each use the mutation engine itself is altered.

Virus Kits Another weapon in the virus writers' armoury is the virus-creation toolkit. To quickly create a number of different viruses, such a toolkit enables a relative novice. Even if viruses formed with toolkits tend to be less sophisticated than viruses designed from scratch, the total number of new viruses that can be created using a toolkit generates a problem for antivirus schemes.

Macro Viruses macro viruses became the most dominant type of virus in the mid of 1990s. For a number of reasons macro viruses are particularly threatening:

1. A macro virus is platform independent. Microsoft Word documents or other Microsoft Office documents are the common target of many macro viruses. It can also affect any operating system and hardware platform that supports these applications.
2. As Most of the information introduced onto a computer system is in the form of a document rather than a program hence, macro viruses infect documents, not executable portions of code.
3. Macro viruses are easily spread, and electronic mail is a very common method of it.
4. Traditional file system access controls are of limited use in preventing their spread, Because macro viruses infect user documents rather than system programs.

Macro feature found in Word and other office applications such as Microsoft Excel, the micro viruses take advantage of it. A macro is an executable program embedded in a word processing document or other type of file. Characteristically, to automate repetitive tasks and thereby save keystrokes users employ macros. Usually, it is some form of the Basic programming language. To invoke the micros user might define a sequence of keystrokes in a macro and set it up so that when a function key or special short combination of keys is input.

increased protection against macro viruses is provided by successive releases of MS Office products. For example, Microsoft offers an optional Macro Virus Protection tool that detects

suspicious Word files and alerts the customer to the potential risk of opening a file with macros. To detect and correct macro viruses, various antivirus product vendors have also developed tools. The arms race continues in the field of macro viruses, as in other types of viruses, but they no longer are the predominant virus threat.

8.2.4 E-Mail Viruses

The e-mail virus is a more recent development in malicious software. Melissa which is the first rapidly spreading e-mail virus made use of a Microsoft Word macro embedded in an attachment. The Word macro is activated, If the recipient opens the email attachment, Then

1. On the mailing list in the user's e-mail package the e-mail virus sends itself to everyone.
2. On the user's system the virus does local damage.

A more powerful version of the e-mail virus is found in the year 1999. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The Visual Basic scripting language supported by the e-mail package was used by the virus.

Hence, a new generation of malware virus replicate itself across the Internet and that arrives via e-mail and uses e-mail software features. So as soon as it is activated either by opening an e-mail attachment or by opening the e-mail, the virus propagates itself to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. Thus, to respond before much damage is done is very difficult for antivirus company. The virus propagates itself. Ultimately, to counter the growing threat a greater degree of security must be built into application software on PCs and Internet utility.

8.2.5 Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. To replicate and propagate again the worm may be activated upon arrival. Worm usually performs some unwanted function, in addition to propagation. Because it propagates itself from system to system, an e-mail virus has some of the characteristics of a worm. Because it uses a document modified to contain viral macro content and requires human action, hence we can still classify it as a virus.

A worm aggressively searches for more machines to infect and each machine that is infected serves as an automated launching pad for attacks on other machines.

To spread from system-to-system network worm programs use network connections. A network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions, when it active within a system.

A network worm uses some sort of network vehicle, to replicate itself. Examples include the following:

- **Electronic mail facility**: A worm mails a copy of itself to other systems, so that its code is run when the e-mail or an attachment is received or viewed.

- **Remote execution capability**: A worm executes a copy of itself on another system, either using an explicit remote execution facility or by exploiting a program flaw in a network service to subvert its operations (such as buffer overflow)
- **Remote login capability**: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other, where it then executes.

On the remote system the new copy of the worm program is then run where, it continues to spread in the same fashion as well it also performs any functions that it performs at that system,

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The functions that are generally carried out by propagation phase are mentioned as the following:

1. By examining host tables or similar repositories of remote system addresses it search for other systems to infect
2. With a remote system it establishes a connection.
3. To the remote system create a copy of itself and cause the copy to be run.

To determine whether a system has previously been infected before copying itself to the system, the network worm may also attempt. In a multiprogramming system, by naming itself as a system process or using some other name that may not be noticed by a system operator it may also cover its presence. Network worms are difficult to counter, as with viruses.

Worm Propagation Model describes a model for worm propagation based on an analysis of recent worm attacks. The total number of hosts infected and the speed of propagation and depend on a number of factors. Which may include the vulnerability or vulnerabilities exploited, the mode of propagation and the degree of similarity to preceding attacks. For the latter factor, an attack that is a variation of a recent previous attack may be countered more effectively than a more novel attack.

Figure 8.5 shows the dynamics for one typical set of parameters. Propagation proceeds through three phases. One can observe that the number of hosts increases exponentially in the initial phase. Consider a simplified case to understand this, in which a worm is launched from a single host and infects two nearby hosts. Each of these hosts infects two more hosts, and so on. This results in exponential growth. After which infecting hosts start attacking already infected hosts and spend some time there, which reduces the rate of infection. The diagram shows the growth is approximately linear during this middle phase, but the rate of infection is rapid. The attack enters a slow finish phase when most vulnerable computers have been infected, as the worm seeks out those remaining hosts that are difficult to identify.

Obviously, to catch the worm in its slow start phase, at a time when few hosts have been infected, is the primary objective in countering a worm is to.

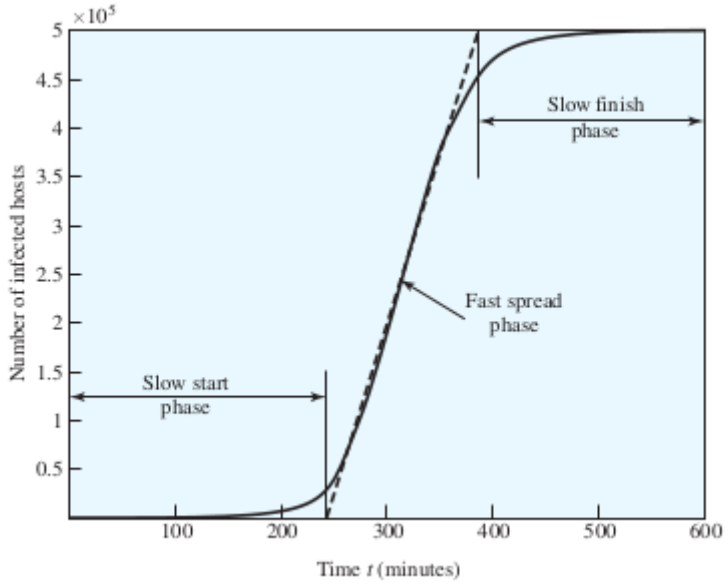


Figure 8.5. Worm Propagation Model

State of Worm Technology The state of the art in worm technology includes the following:

- **Multiplatform**: Newer worms can attack a variety of platforms, especially the popular varieties of UNIX, they not limited to Windows machines.
- **Multiexploit**: a variety of ways are used by worms such as using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications to penetrate system.
- **Ultrafast spreading**: To conduct a prior Internet scan to accumulate Internet addresses of vulnerable machines is one of the popular techniques to accelerate the spread of a worm.
- **Polymorphic**: Worms adopt the virus polymorphic technique to evade detection, skip past filters, and foil real-time analysis. Each copy of the worm has new code generated on the fly using functionally equivalent instructions and encryption techniques.
- **Metamorphic**: In addition to changing their appearance, metamorphic worms have a repertoire of behaviour patterns that are unleashed at different stages of propagation.
- **Transport vehicles**: They are ideal for spreading other distributed attack tools, such as distributed denial of service bots. Because worms can rapidly compromise a large number of systems.
- **Zero-day exploit:** A worm should exploit an unknown vulnerability to achieve maximum surprise and distribution, that is only discovered by the general network community when the worm is launched.

8.2.6 Bots

A bot (robot), is a program that secretly takes over another Internet-attached computer they are also known as a zombie or drone. Bot then uses that Internet-attached computer to launch attacks that are difficult to trace to the bot's creator. Hundreds or thousands of computers belonging to unsuspecting third parties are planted by the typical bot. The collection of bots often is capable of acting in a coordinated manner; such a collection is referred to as a botnet.

Three characteristics that exhibits by botnet: a remote-control facility, the bot functionality and a spreading mechanism to propagate the bots and construct the botnet. Each of these characteristics we will examine in turn.

Uses of Bots lists the following uses of bots:

- **Distributed denial-of-service attacks**: A DDoS attack is an attack that causes a loss of service to users. It takes place on a computer system or network.
- **Spamming**: An attacker is able to send massive amounts of bulk e-mail (spam) with the help of a botnet and thousands of bots.
- **Sniffing traffic**: To watch for interesting clear- text data passing by a compromised machine Bots can use a packet sniffer. Mostly to retrieve sensitive information like usernames and passwords the sniffers are used.
- **Keylogging**: Just sniffing the network packets on the victim's computer is useless, if the compromised machine uses encrypted communication channels (e.g., HTTPS or POP3S), because the appropriate key to decrypt the packets is missing. A keylogger captures keystrokes on the infected machine, by using it an attacker can retrieve sensitive information. An implemented filtering mechanism (e.g., "I am only interested in key sequences near the keyword 'paypal.com'") further helps in stealing secret data.
- Spreading new malware: To spread new bots botnets are used. Since all bots implement mechanisms to download and execute a file via HTTP or FTP, this is very easy to carry out. A botnet with 10,000 hosts that acts as the start base for a worm or mail virus allows very fast spreading and thus causes more harm.
- **Installing advertisement add-ons and browser helper objects (BHOs): T**o gain financial advantages Botnets can also be used. By setting up a fake Web site with some advertisements this can be achieved: The operator of this Web site negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be "automated" so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start page of a compromised machine so that the "clicks" are executed each time the victim uses the browser.
- **Attacking IRC chat networks**: Botnets are also used for attacks against Internet relay chat (IRC) networks. Popular among attackers is especially the so-called clone attack: In this kind of attack, the controller orders each bot to connect a large number of clones to the victim IRC network. The victim is flooded by service request from thousands of bots or thousands of channel-joins by these cloned bots. In this way, the victim IRC network is brought down, similar to a DDoS attack.
- **Manipulating online polls/games**: Online polls/games are getting more and more attention and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way.

Remote Control Facility The remote-control facility is distinguishing a bot from a worm. A worm propagates itself and activates itself, whereas a bot is controlled from some central facility, at least initially.

The remote-control facility is typically implemented on an IRC server. By joining a specific channel on this server all bots treat incoming messages as commands. More recent botnets tend to use covert communication channels via protocols such as HTTP and to avoid IRC mechanisms To avoid a single point of failure, distributed control mechanisms are used.

The control module can activate the bots, once a communications path is established between a control module and the bots. To causes the bot to execute routines that are already implemented in the bot, the control module simply issues command to the bot. To achieve larger flexibility, update commands can be issued by the control module that instruct the bots to download a file from some Internet location and execute it. The bot in this latter case becomes a more general-purpose tool that can be used for multiple attacks.

Constructing the Attack Network, To infect a number of machines with bot software that will ultimately be used to carry out the attack, is the first step in a botnet attack is for the attacker. In this phase of the attack the essential ingredients are the following:

1. Software that can carry out the attack. The software must be able to run on a large number of machines, must be able to conceal its existence, must be able to communicate with the attacker or have some sort of time-triggered mechanism, and must be able to launch the intended attack toward the target.
2. A vulnerability in a large number of systems. The attacker must become aware of a vulnerability that many system administrators and individual users have failed to patch and that enables the attacker to install the bot software.
3. A process for locating and identifying vulnerable machines, known as **scanning** or **fingerprinting**.

A number of vulnerable machines are identified first in the scanning process by the attacker, and start to infects them. Then, the same scanning process is repeated by bot software that is installed in the infected machines, until a large distributed network of infected machines is created. The following are the various types of scanning strategies:

- **Random**: Using a different seed each compromised host probes random addresses in the IP address space. A high volume of Internet traffic is produced by this technique, which may cause generalized disruption even before the actual attack is launched.
- **Hit list**: a long list of potential vulnerable machines is first compiled by the attacker. This can be a slow process done over a long period to avoid detection that an attack is underway. Then the attacker begins infecting machines on the list once the list is compiled. A portion of the list to scan is provided to each infected machine. This strategy results in a very short scanning period, which may make it difficult to detect that infection is taking place.
- **Topological**: To find more hosts to scan this method uses information contained on an infected victim machine.
- **Local subnet**: The host looks for targets in its own local network, If a host can be infected behind a firewall. To find other hosts that would otherwise be protected by the firewall, the host uses the subnet address structure.

8.3 AUTHENTICATION

User authentication is the primary line of defence and the fundamental building block in most computer security contexts. For user accountability and for most types of access control user authentication is the basis. User authentication defined by RFC 2828 is as follows:

The process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

- Identification step: Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- Verification step: Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

Let us understand with an example, user Alice Toklas could have the user identifier ABTOKLAS. Alice wishes to use and could be known to system administrators and other users; this information needs to be stored on any server or computer system. A password associated with this user ID is typical item of authentication information, which is kept secret (known only to Alice and to the system). The combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity, If no one is able to obtain or guess Alice's password. Because Alice's ID is not secret, system users can send her e-mail, but because her password is secret, no one can pretend to be Alice.

In essence, a user provides a claimed identity to the system by means of identification; user authentication is the means of establishing the validity of the claim. One should understand here and should make a note that user authentication is distinct from message authentication. The procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic is Message authentication.

Means of Authentication

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows**: Examples includes a password, a personal identification number (PIN), or answers to a prearranged set of questions
- **Something the individual possesses**: Examples include electronic key cards, smart cards, and physical keys. This type of authenticator is referred to as a token.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics)**: Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All methods mentioned here can provide secure user authentication, if they are properly implemented and used. However, each method has problems. One can able to guess or steal a password using adversary. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Further, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. There are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience with respect to biometric authenticators.

8.3.1 Password-Based Authentication

A widely used line of defence against intruders is the password system. Virtually all multiuser systems, network-based servers, Web-based e-commerce sites, and other similar services require that a user provide not only a name or identifier (ID) but also a password. The system compares the password to a previously stored password for that user ID, maintained in a system password file. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

The Use of Hashed Passwords A widely used password security technique is the use of hashed passwords and a salt value. This scheme is found on virtually all UNIX variants as well as on a number of other operating systems. The following procedure is employed (Figure 15.1a). To load a new password into the system, the user selects or is assigned a password. This password is combined with a fixed-length salt value [MORR79]. In older implementations, this value is related to the time at which the password is assigned to the user. Newer implementations use a pseudo- random or random number. The password and salt serve as inputs to a hashing algorithm to produce a fixed-length hash code. The hash algorithm is designed to be slow to execute to thwart attacks. The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID. The hashed-password method has been shown to be secure against a variety of cryptanalytic attacks.

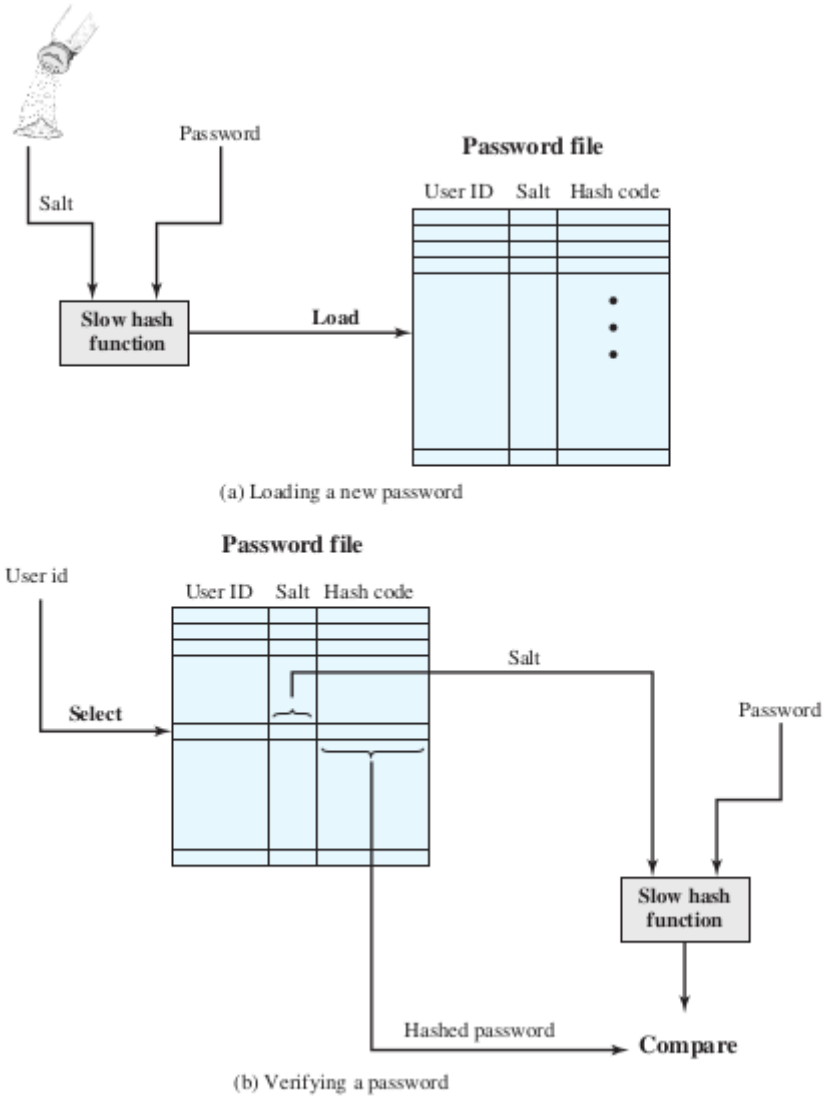


Figure 8.6. UNIX Password Scheme

When a user attempts to log on to a UNIX system, the user provides an ID and a password (Figure 6b). The operating system uses the ID to index into the password file and retrieve the plaintext salt and the encrypted password. The salt and user-supplied password are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned different salt values. Hence, the hashed passwords of the two users will differ.
- It greatly increases the difficulty of offline dictionary attacks. For a salt of length b bits, the number of possible passwords is increased by a factor of 2 b, increasing the difficulty of guessing a password in a dictionary attack.
- It becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them.

To see the second point, consider the way that an offline dictionary attack would work. The attacker obtains a copy of the password file. Suppose first that the salt is not used. The attacker's goal is to guess a single password. To that end, the attacker submits a large number of likely passwords to the hashing function. If any of the guesses matches one of the hashes in the file, then the attacker has found a password that is in the file. But faced with the UNIX scheme, the attacker must take each guess and submit it to the hash function once for each salt value in the dictionary file, multiplying the number of guesses that must be checked.

There are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check many thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through millions of possible passwords in a reasonable period.

UNIX Implementations

Since the original development of UNIX, most implementations have relied on the following password scheme. Each user selects a password of up to eight printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The hash routine, known as crypt(3), is based on DES. A 12-bit salt value is used. The modified DES algorithm is executed with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence. The modification of the DES algorithm converts it into a one-way hash function. The crypt(3) routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25.

This particular implementation is now considered woefully inadequate. For example, reports the results of a dictionary attack using a supercomputer. The attack was able to process over 50 million password guesses in about 80 minutes. Further, the results showed that for about $10,000 anyone should be able to do the same in a few months using one uniprocessor machine. Despite its known weaknesses, this UNIX scheme is still often required for compatibility with existing account management software or in multivendor environments.

There are other, much stronger, hash/salt schemes available for UNIX. The recommended hash function for many UNIX systems, including Linux, Solaris, and FreeBSD, is based on the MD5 secure hash algorithm (which is similar to, but not as secure as SHA-1). 1 The MD5 crypt routine uses a salt of up to 48 bits and effectively has no limitations on password length. It produces a 128-bit hash value. It is also far slower than crypt(3). To achieve the slowdown, MD5 crypt uses an inner loop with 1000 iterations

Probably the most secure version of the UNIX hash/salt scheme was developed for OpenBSD, another widely used open-source UNIX. This scheme, reported, uses a hash function based on the Blowfish symmetric block cipher. The hash function, called Bcrypt, is quite slow to execute.

Bcrypt allows passwords of up to 55 characters in length and requires a random salt value of 128 bits, to produce a 192-bit hash value. Bcrypt also includes a cost variable; an increase in the cost variable causes a corresponding increase in the time required to perform a Bcyrpt hash. The cost assigned to a new password is configurable, so that administrators can assign a higher cost to privileged users.

8.3.2 Token-Based Authentication

Objects that a user possesses for the purpose of user authentication are called tokens. In this subsection, we examine two types of tokens that are widely used; these are cards that have the appearance and size of bank cards.

Memory Cards Memory cards can store but not process data. The most common such card is the bank card with a magnetic stripe on the back. A magnetic stripe can store only a simple security code, which can be read (and unfortunately reprogrammed) by an inexpensive card reader. There are also memory cards that include an internal electronic memory.

Memory cards can be used alone for physical access, such as a hotel room. For computer user authentication, such cards are typically used with some form of password or personal identification number (PIN). A typical application is an automatic teller machine (ATM).

The memory card, when combined with a PIN or password, provides significantly greater security than a password alone. An adversary must gain physical possession of the card (or be able to duplicate it) plus must gain knowledge of the PIN. Among the potential drawbacks are the following:

- **Requires special reader**: This increases the cost of using the token and creates the requirement to maintain the security of the reader's hardware and software.
- **Token loss**: A lost token temporarily prevents its owner from gaining system access. Thus, there is an administrative cost in replacing the lost token. In addition, if the token is found, stolen, or forged, then an adversary now need only determine the PIN to gain unauthorized access.
- **User dissatisfaction**: Although users may have no difficulty in accepting the use of a memory card for ATM access, its use for computer access may be deemed inconvenient.

Smart Cards A wide variety of devices qualify as smart tokens. These can be categorized along three dimensions that are not mutually exclusive:

- **Physical characteristics**: Smart tokens include an embedded microprocessor. A smart token that looks like a bank card is called a smart card. Other smart tokens can look like calculators, keys, or other small portable objects.
- **Interface**: Manual interfaces include a keypad and display for human/token interaction. Smart tokens with an electronic interface communicate with a compatible reader/writer.
- **Authentication protocol**: The purpose of a smart token is to provide a means for user authentication. We can classify the authentication protocols used with smart tokens into three categories:

- **Static**: With a static protocol, the user authenticates himself or herself to the token and then the token authenticates the user to the computer. The latter half of this protocol is similar to the operation of a memory token.
- **Dynamic password generator**: In this case, the token generates a unique password periodically (e.g., every minute). This password is then entered into the computer system for authentication, either manually by the user or electronically via the token. The token and the computer system must be initialized and kept synchronized so that the computer knows the password that is current for this token.
- **Challenge-response**: In this case, the computer system generates a challenge, such as a random string of numbers. The smart token generates a response based on the challenge. For example, public-key cryptography could be used and the token could encrypt the challenge string with the token's private key.

For user authentication to computer, the most important category of smart token is the smart card, which has the appearance of a credit card, has an electronic interface, and may use any of the type of protocols just described. The remainder of this section discusses smart cards.

A smart card contains within it an entire microprocessor, including processor, memory, and I/O ports. Some versions incorporate a special co-processing circuit for cryptographic operation to speed the task of encoding and decoding messages or generating digital signatures to validate the information transferred. In some cards, the I/O ports are directly accessible by a compatible reader by means of exposed electrical contacts. Other cards rely instead on an embedded antenna for wireless communication with the reader.

8.3.3 Biometric Authentication

A biometric authentication system attempts to authenticate an individual based on his or her unique physical characteristics. These include static characteristics, such as fingerprints, hand geometry, facial characteristics, and retinal and iris patterns; and dynamic characteristics, such as voiceprint and signature. In essence, biometrics is based on pattern recognition. Compared to passwords and tokens, biometric authentication is both technically complex and expensive. While it is used in a number of specific applications, biometrics has yet to mature as a standard tool for user authentication to computer systems.

A number of different types of physical characteristics are either in use or under study for user authentication. The most common are the following:

- **Facial characteristics**: Facial characteristics are the most common means of human-to-human identification; thus, it is natural to consider them for identification by computer. The most common approach is to define characteristics based on relative location and shape of key facial features, such as eyes, eyebrows, nose, lips, and chin shape. An alternative approach is to use an infrared camera to produce a face thermogram that correlates with the underlying vascular system in the human face.
- **Fingerprints**: Fingerprints have been used as a means of identification for centuries, and the process has been systematized and automated particularly for law enforcement purposes. A fingerprint is the pattern of ridges and furrows on the surface of the fingertip. Fingerprints are believed to be unique across the entire human population. In practice,

automated fingerprint recognition and matching system extract a number of features from the fingerprint for storage as a numerical surrogate for the full fingerprint pattern.

- **Hand geometry**: Hand geometry systems identify features of the hand, including shape, and lengths and widths of fingers.
- **Retinal pattern**: The pattern formed by veins beneath the retinal surface is unique and therefore suitable for identification. A retinal biometric system obtains a digital image of the retinal pattern by projecting a low-intensity beam of visual or infrared light into the eye.
- **Iris**: Another unique physical characteristic is the detailed structure of the iris.
- **Signature**: Each individual has a unique style of handwriting, and this is reflected especially in the signature, which is typically a frequently written sequence. However, multiple signature samples from a single individual will not be identical. This complicates the task of developing a computer representation of the signature that can be matched to future samples.
- **Voice**: Whereas the signature style of an individual reflects not only the unique physical attributes of the writer but also the writing habit that has developed, voice patterns are more closely tied to the physical and anatomical characteristics of the speaker. Nevertheless, there is still a variation from sample to sample over time from the same speaker, complicating the biometric recognition task.

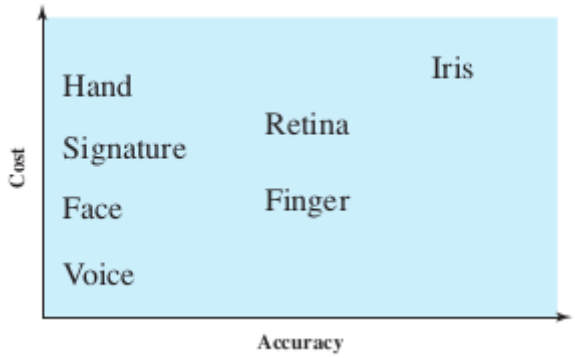


Figure 8.7. Cost versus Accuracy of Various Biometric Characteristics in User Authentication Schemes

igure 8.7. gives a rough indication of the relative cost and accuracy of these biometric measures. The concept of accuracy does not apply to user authentication schemes using smart cards or passwords. For example, if a user enters a password, it either matches exactly the password expected for that user or not. In the case of biometric parameters, the system instead must determine how closely a presented biometric characteristic matches a stored characteristic. Before elaborating on the concept of biometric accuracy, we need to have a general idea of how biometric systems work.

Exercise Questions:

1. Define with one example of viruses, worms, and bots.
2. Write a short note on computer security concepts.
3. Discuss the various terms threats, attacks, and assets.
4. Write a sort note on Token-Based Authentication.
5. Write a sort note on Biometric Authentication.
6. Explain in short, the Means of Authentication.
7. What is Authentication? Explain in short various Authentication methods.

References:

1. Operating System Concepts, 8th Edition, by Galvin et al, 2008, Wiley Publications.
2. Lecture notes and ppt of Ariel J. Frank, Bar-Ilan University.
3. **Operating Systems | Internals and Design Principles | by William Stallings, Ninth Edition | By Pearson Publications**
4. **Web Portal:** https://www.geeksforgeeks.org