DJA3C - OPERATING SYSYTEMS

SYLLABUS

UNIT I Introduction: What is operating systems do-Computer System Architecture-Operating System Structure –Operating System Operations-Computing Environments-Open Source Operating Systems.

UNIT II System Structures: Operating System Services-System Calls-System Programs-Operating System Structures. Process Management: Process Concept-Process Scheduling-Interprocess communication.

UNIT III Process Scheduling: Basic Concepts-Scheduling Criteria-Scheduling Algorithms (FCFS, SJF & Round Robin only) - Algorithm Evaluation. Synchronization: Back ground-The Critical Section Problem-Peterson's SolutionSynchronization Hardware-Mutex Locks-Semaphores-The Dining-Philosopher's Problem.

UNIT IV Deadlock: Deadlock Characterization-Methods Handling Deadlocks-Recovery from Deadlock. Memory Management Strategies: Background-Contiguous Memory Allocation-SegmentationPaging.

UNIT V Virtual Memory Management: Demand Paging-Page Replacement. File System: Directory and Disk Structure Implementing File-Systems: Allocation Methods Mass Storage Structure: Disk Scheduling.

Text Book: OPERATING SYSTEMS CONCEPTS- Abraham Silberschatz, Peter B Galvin, Gerg Gagne-NINTH EDITION (2015 INDIA EDITION)-WILEY

Reference Book: THE MINIX BOOK OPERATING SYSTEMS DESIGN AND IMPLENTATION-Third Edition-ANDREW S.TANENBAUM and ALBERT S WOODHULL. – PEARSON

UNIT I

1.1 INTRODUCTION

An operating system can be considered as computer program (software) that manages the computer hardware and acts as an intermediary between the computer user and the hardware. With the help of an operating system environment, a user can do things in a convenient and efficient manner.

Kernel is the program running at all times on the computer. ***System programs*** are associated with the operating system but are not necessarily forms part of the kernel. ***Application programs*** include all programs not associated with the operation of the system.

Different operating systems are available under various environments as listed below:

- Mainframe operating systems: designed to optimize utilization of hardware.
- Personal computer (PC) operating systems: designed to support user friendly applications, business applications and computer games.

Functions of Operating System

- Booting the computer
- Performing basic computer tasks like managing the peripheral devices
- Provides file management like manipulating storing, retrieving and saving
- Handling system resources like computer's memory, and printer
- Handling errors by applying error prevention methods.
- Providing user interface like Graphical User Interface (GUI)

Operating System Management Tasks

Application allows standard communication between software and computer. The User interface allows communicating with computer. Other tasks are:

- ***Processor management*** arranges the tasks into order and pairing them into manageable size before processing by CPU.
- ***Memory management*** coordinates data to and from Random Access Memory (RAM) and applies techniques like paging and segmentation.
- ***Device management*** provides interface between connected devices.
- ***Storage management*** directs permanent data storage.
- ***Security*** prevents unauthorized access to programs and data.

- ***Control over system performance*** considers the request for a service and response from the system.
- ***Job accounting*** keeps track of time and resources used by various jobs and users.
- ***Error detecting*** by producing traces, error messages, debugging and error detecting.
- ***Coordination between other softwares and users*** Coordinating and assigning compilers, interpreters, assemblers and other software to the various users of the computer systems.

1.2 WHAT OPERATING SYSTEMS DO?

The three major components of a computer system are,(Fig 1.1)

- The hardware consisting of the Central Processing Unit (CPU), the memory, and the Input/Output devices
- The operating system that controls the hardware resources and coordinates its use among the various application programs for the users.
- The application programs such as Office tools (Word Processors, spreadsheets etc.,) Web browsers (Google, Opera etc.,) uses the hardware resources to solve computing problems for the users.

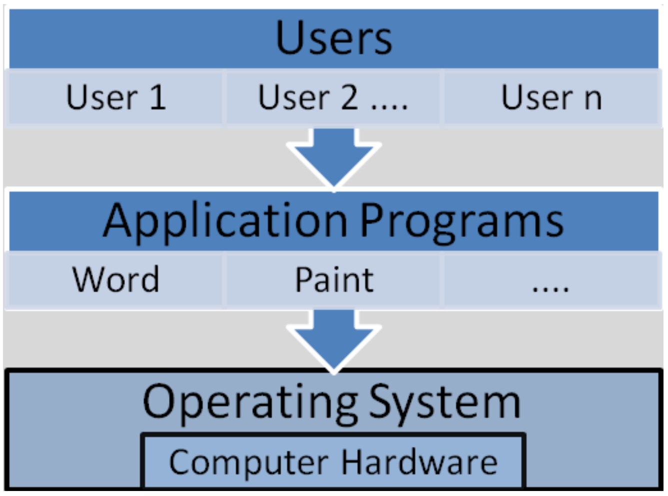


Fig 1.1 Components of computer system

A user expects an operating system to be user friendly, to perform well and to maximize resource utilization. When system is concerned, an OS can be considered as a resource allocator.

An OS does the following activities:

- The operating system acts as the manager of the resources like CPU time, memory space, file-storage space, peripheral devices, etc.,

- It decides how to allocate the available resources to needed programs and users so that it can operate the computer system efficiently and effectively.
- An operating system also acts as a control program controlling various I/O devices and user programs by managing the execution of user programs to prevent errors.

Types of Operating Systems

Some of the most widely used types of Operating system are,

- Simple Batch System
- Multiprogramming Batch System
- Multiprocessor System
- Distributed Operating System
- Realtime Operating System

Simple Batch Systems

- The user submits a job written on cards/ tape to a computer operator.
- The computer operator places a batch of several jobs on an input device.
- Jobs are grouped together according to the type of language and requirement.
- The OS program called as monitor, manages the execution of each program in the batch.
- The monitor program resides in the main memory and is always available for execution.

Disadvantages:

- In batch processing method, there is no direct interaction between user and the computer.
- Priority to important process cannot be provided.
- CPU can be idle.

Multiprogramming Batch Systems

- Aims to maximize CPU utilization.
- The operating system picks and executes one job from memory.
- If the picked job needs an I/O operation, the OS immediately switches to another job for executing that job making the CPU busy always.
- If several jobs are ready for execution at the same time, CPU Scheduling is carried out.
- In multiprogrammed system, CPU is always busy executing one process or the other.

- Time-Sharing Systems aim is to minimizing the response time.

Multiprocessor Systems

- It has several processors that share a common physical memory.
- Execution of several tasks by different processors concurrently.
- If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors.
- Provides higher computing power and speed.
- All processors operate under single operating system.
- Enhanced performance is achieved
- Increases the system's throughput without speeding up the execution of a single task.

Distributed Operating Systems

- Many computers are inter-connected by communication networks.
- Multiple systems are involved in communication.
- User at one system can utilize the resources of other systems.
- Load can be distributed among systems.
- Fast processing.

Real-Time Operating System

- The Real-Time Operating system guarantees the maximum time for critical operations and complete them on time.
- Gives maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- The critical task will get priority over other tasks

1.3 COMPUTER-SYSTEM ARCHITECTURE

A computer system can be organized as follows, based on number of general-purpose processors used.

- Single-Processor Systems
- Multiprocessor Systems
- Clustered Systems

Single-Processor Systems: If there is only one general-purpose CPU, then the system is a single-processor system. A single processor system, has one main CPU for executing a general-purpose instruction set. It also has processors for specific purposes. Device-specific processors like disk controller, graphics controllers and I/O processor for quick access to data are special purpose processors. The operating system sends these processors information about their next task and monitors their status. The overhead of the main CPU is relieved by the arrangement.

Multiprocessor Systems: Multiprocessor systems are also known as parallel systems have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices. Three important advantages of Multiprocessor systems are

- Increased throughput - more work is carried out in less time.
- Low cost – since it shares peripheral devices there is much reduction in cost. Also processors share data stored on a disk thereby reducing requirement of storage media.
- Increased reliability – since many processors are used, failure of a single processor would only slow down the system rather than halting altogether thereby increasing reliability of a computer system.

  The two types of multiple-processor systems are:
- ***Asymmetric multiprocessing*** - each processor is assigned a specific task. One main processor controls the system; the other processors get instructions from it or have predefined tasks. The main processor schedules and allocates work to the remaining processors.
- ***Symmetric multiprocessing*** (SMP) - each processor has its own set of registers and cache memory but all processors share common physical memory. Each processor performs all tasks within the operating system. All processors are main peers. Figure 1.2 illustrates a typical SMP architecture.
- Latest CPU design includes multiple computing cores on a single chip. Such multiprocessor systems are termed multicore. Recent development is blade servers in which multiple processor boards, I/O boards, and networking boards are placed in the same chassis. Each blade-processor board boots independently and runs its own operating system.

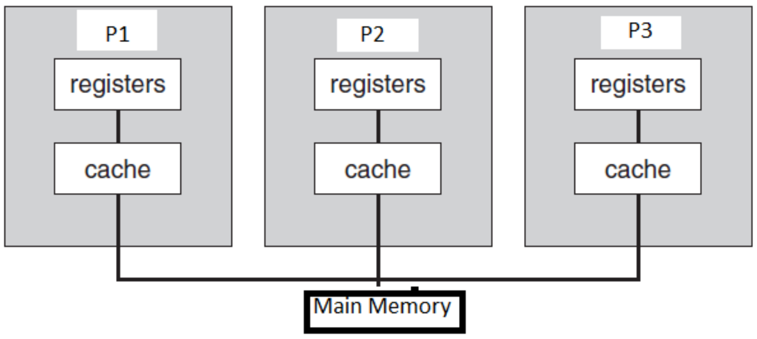


Fig.1.2 SMP Architecture

Clustered Systems: Multiple CPUs are gathered together. Composes of two or more individual systems / nodes joined together. Each node may be a single processor system or a multicore system. Clustered computers share storage and are closely linked via a Local Area Network (LAN). Clustering provides high reliability - service will continue even if one or more systems in the cluster fail. Clustering can be structured asymmetrically or symmetrically.

In asymmetric clustering, one machine is in standby mode while the other is running the applications. The standby host machine monitors the active server. If the server fails, the standby host becomes the active server.

In symmetric clustering, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware. More than one application must be available to run.

1.4 OPERATING-SYSTEM STRUCTURE

An important aspect of operating systems is its multi-programming capability. A single program cannot, keep the CPU and I/O devices busy at all times. **Multiprogramming** increases CPU utilization by organizing jobs i.e. code and data, so that the CPU always has a job to execute. The operating system keeps several jobs in memory simultaneously (Figure 1.3).

Normally, main memory is too small to accommodate all jobs. So, the jobs are kept initially on the disk in the **job pool** consisting of all processes residing on disk awaiting main memory allocation. The operating system simply switches between jobs. As long as at least one job needs to execute, the CPU is never idle. Thus in Multi-programmed systems various system resources like CPU, memory, and peripheral devices are utilized effectively, even without providing for user interaction with the computer system.

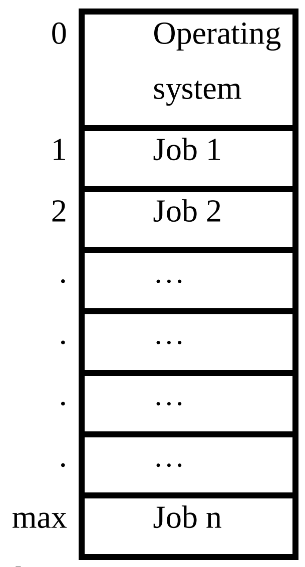


Fig. 1.3 Multiprogramming system memory

Multitasking or **Time sharing** is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

A time-shared operating system allows many users to share the computer simultaneously. Since , only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is under the impression that the entire computer system is dedicated to his/her use, even though it is being shared among many users.

Multitasking requires an interactive computer system, which provides ***direct communication between the user and the system.*** The user gives instructions to the operating system or to a program directly, using an input device and waits for immediate results on an output device. The **response time** is typically less than one second.

A time-shared operating system uses CPU scheduling and multiprogramming, to provide each user with a small portion of a time-shared computer. **Job scheduling, CPU scheduling, Swapping** and **Virtual memory** are some of the techniques that are applied to carry out multi-tasking in an efficient manner.

1.5 OPERATING-SYSTEM OPERATIONS

Modern operating systems are **Interrupt Driven.** The next event to happen is signaled by the occurrence of an interrupt or a trap. A **trap** (or an **exception**) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program. For each type of interrupt, the operating system determines corresponding action should be taken. If this method is not strictly followed, correct program execution is not possible in a multi-program or time-sharing environment.

Dual-Mode and Multimode Operation

To ensure the proper execution, an operating system, must be able to identify the code correctly. Two separate ***modes*** of operation: **user mode** and **kernel mode** also called as **supervisor mode/ system mode/privileged mode**. A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode:

Kernel – mode bit = 0

User – mode bit = 1

With the mode bit, one can distinguish between a task executed under kernel mode and the user mode. When the computer system is executing on behalf of a user application, the system is in user mode. Likewise, when a user application requests a service from the operating system (via a system call), the system transition takes place from user to kernel mode to fulfill the request. (Figure 1.4)

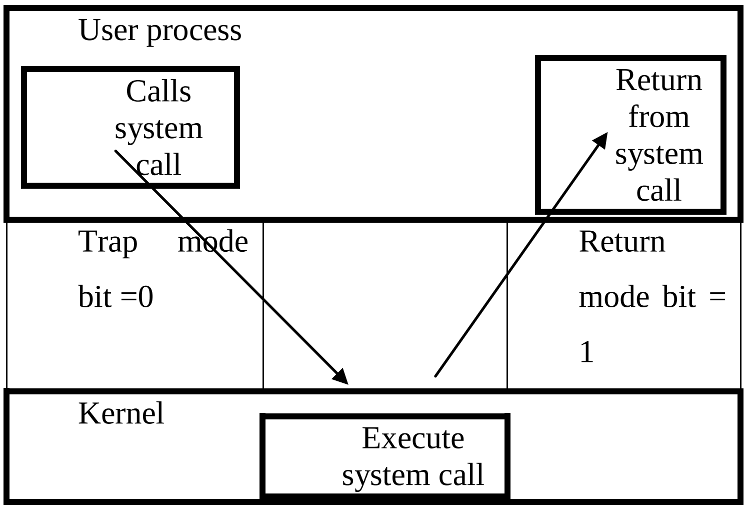


Fig 1.4 Dual mode operation

- At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0).
- Whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

Instruction execution Life Cycle in a computer system:

- Initial control resides in the operating system, where instructions are executed in kernel mode.
- When control is given to a user application, the mode is set to user mode.
- Eventually, control is switched back to the operating system through an interrupt, a trap, or a system call.

Advantages:

- Protecting the operating system from errant users
- Allows privileged instructions to be executed only in kernel mode.

Timer

For the operating system to maintain control over the CPU a timer can be used. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A variable timer is generally implemented by a fixed-rate clock and a counter.

Setting the Timer: The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

- Before turning over control to the user, the operating system ensures that the timer is set to interrupt.
- If the timer interrupts, control transfers automatically to the operating system.

Advantage: Preventing a user program from running too long.

1.6 COMPUTING ENVIRONMENTS

Various types of computing environments for an operating system are listed below:

- Traditional computing

- Mobile computing
- Distributed computing
- Client-server computing
- Peer-to-peer computing

Traditional Computing – Traditional systems are time-sharing systems. Early computers systems were either batch or interactive systems. Batch systems processed jobs in bulk, with predetermined input from files or other data sources. Interactive systems waited for input from users. To optimize the use of the computing resources, multiple users shared time on these systems.

Time-sharing systems used a timer and scheduling algorithms to cycle processes rapidly through the CPU, giving each user a share of the resources. Time sharing systems are used on desktop computers, laptops, servers, and even mobile computers, but frequently all the processes are owned by the same user (or a single user and the operating system). It applied the simple fact that it can perform different tasks at the same time.

Mobile Computing: It refers to computing on handheld smart phones and tablet computers. These devices share the distinguishing physical features of being portable and lightweight. Playing music and video, reading digital books, taking photos, and recording high-definition video are all possible using a mobile phone. Its unique features include Global Positioning System (GPS) chips, accelerometers and gyroscopes. In a Mobile OS

- The memory capacity and processing speed of mobile devices, are more limited than those of PCs.
- Mobile devices use processors that are smaller, are slower, and offer fewer processing cores.
- Two widely used mobile computing operating systems are: **Apple iOS** and **Google Android**. iOS was designed to run on Apple iPhone and iPad mobile devices. Android powers smartphones and tablet computers available from many manufacturers.

Distributed Computing: It is a collection of physically separate, heterogeneous computer systems that are networked together. It provides users with access to the various resources that the system maintains.

Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interface's device driver. Others make users specifically invoke network functions. Normally a blend of both is applied. The protocols that create a distributed system might affect that system's utility.

Distributed systems depend on networking for their functionality. Networks vary by the protocols used, the distances between nodes, and the transport media. **TCP/IP** is the most common network protocol, and it provides the fundamental architecture of the Internet. Most operating systems support TCP/IP, including all general-purpose ones. To an operating system, a network protocol needs an interface device like network adapter and related software to handle data.

A **network operating system** is one that provides the following features:

- File sharing across the network,
- Communication scheme that allows different processes on different computers to exchange messages.
- Acting autonomously from all other computers on the network

A distributed operating system provides a less autonomous environment. The different computers communicate closely enough to provide the illusion that only a single operating system controls the network.

Client–Server Computing: Systems that act as server systems satisfy the requests generated by client systems. A **client–server** system, has the general structure depicted in Figure 1.5.

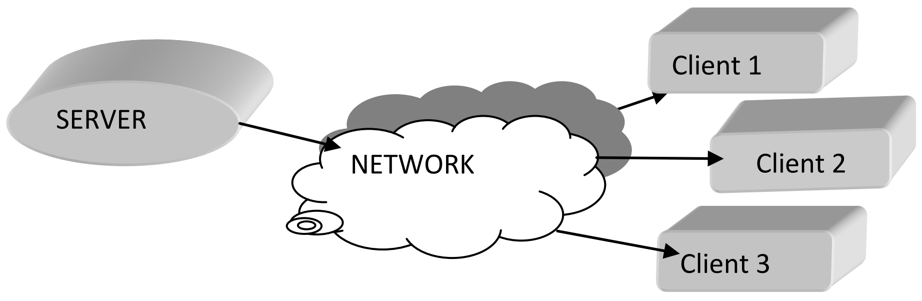


Fig 1.5 Client server system

Server systems can be broadly categorized as compute servers and file servers:

• The **compute-server system** provides an interface to which a client can send a request to perform an action. In response, the server executes the action and sends the results to the client.

• The **file-server system** provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

Peer-to-Peer Computing (P2P): All nodes within the system are considered peers, that is each node may act as either a client or a server, depending on whether it is requesting or providing a service. Services can be provided by several nodes distributed throughout the network.

To participate in a peer-to-peer system, a node must initially join the network of peers. Once a node has joined the network, it can provide services. Service providing methods are,

- Using centralized lookup service table
- Broadcasting request for service

Centralized lookup service : When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service contacts this centralized lookup service to determine which node provides the service and communicates with it for service.

Broadcasting request: The peer acting as a client must discover which node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request. To carry out this, a ***discovery protocol*** must be provided that allows peers to discover services provided by other peers in the network. Figure 1.6 illustrates P2P scenario.

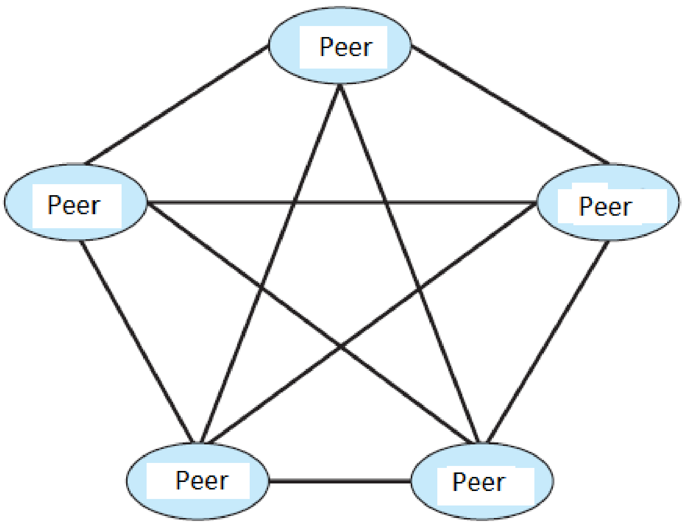


Fig1.6 P2P network without centralized server

Cloud Computing: Cloud computing is a type of computing that delivers computing, storage, and applications as a service across a network.

Types of cloud computing:

• Public cloud—a cloud available through the Internet to the public willing to pay for the service.

• Private cloud—a cloud for a private company for its own personal use

• Hybrid cloud—a cloud that includes both public and private cloud components.

Cloud services:

• Software as a service (SaaS)—one or more applications (such as text editors, spreadsheets) available via the Internet

• Platform as a service (PaaS)—a software stack ready for application use via the Internet (a database server)

• Infrastructure as a service (IaaS)—servers or storage available over the Internet (backup storage)

1.7 OPEN-SOURCE OPERATING SYSTEMS

Open source operating systems are those available in source-code format rather than as compiled binary code. Linux is the most famous open source operating system. Starting with the source code allows the programmer to produce binary code that can be executed on a system. Learning operating systems by examining the source code is possible. The source code can be modified for the operating system, compiled and executed to try out the changes.

Benefits:

- Any interested programmer can contribute to the code by helping to debug it, analyze it, provide support, and suggest changes.
- Open-source code is more secure than closed-source code because many more eyes are viewing the code. List of Open Source OS:
    - Linux
    - BSD UNIX
    - Solaris
- **Linux:** In 1991, Linus Torvalds from Finland, released Linux which used UNIX-like kernel. GNU/Linux operating system has unique **distributions**, or custom builds, of the system. Major distributions are:

    - RedHat
    - SUSE
    - Fedora
    - Debian
    - Slackware
    - Ubuntu.

  Distributions vary in function, utility, installed applications, hardware support, user interface, and purpose. PCLinuxOS is a **LiveCD**—an operating system that can be booted and run from a CD-ROM without being installed on a system's hard disk. ***PCLinuxOS Supergamer DVD*** —is a **LiveDVD** that includes graphics drivers and games. A gamer can run it on any compatible system by booting from the DVD.

- **BSD-UNIX:** A fully functional, open-source version, 4.4BSD-lite, was released in 1994. Distributions of BSD UNIX, are:
    - FreeBSD
    - NetBSD
    - OpenBSD
    - DragonflyBSD.
    -

- **Solaris** is the commercial UNIX-based operating system of Sun Microsystems. Several groups interested in using OpenSolaris have expanded its features. Their working set is Project Illumos, which has more features than OpenSolaris base.

  The usage of free software and open sourcing are increasing day-by-day. Improved quality of open-source projects has lead to an increase in the number open source software users.

* * *

UNIT II

2.1 OPERATING SYSTEM STRUCTURES

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs. An operating system provides the environment within which programs are executed. Various algorithms and strategies are applied in designing an Operating System. The major concentration must be on the following:

- The services that the system provides;
- The interface that it makes available to users and programmers;
- The components and their interconnections.

2.2 OPERATING-SYSTEM SERVICES

The various services provided by an OS (Fig 2.1) can be categorized into two types:

- User Services
    - User Interface
    - Program Execution
    - I/O operations
    - File system services
    - Communication service
    - Error Detection

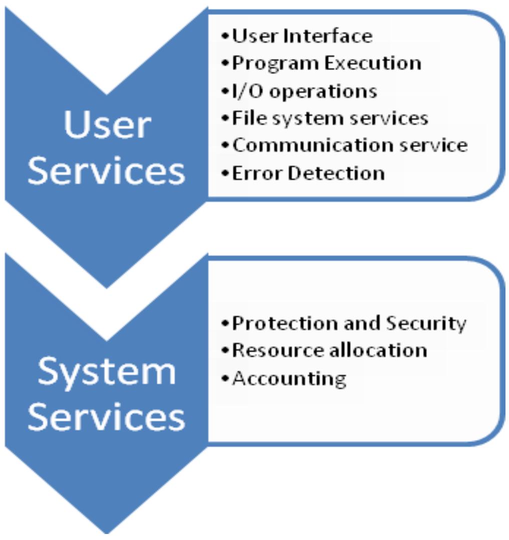


Fig 2.1 OS services

- System Services

    - Protection and Security Service
    - Resource allocation
    - Accounting
- ***User interface:*** Different types of interface are available.
    - Command-line interface: (CLI), which uses text commands and a method for entering them.
    - Batch interface(BI): A Batch file with commands and directives to control those commands are the contents of a batch files which are executed when needed.
    - Graphical User Interface (GUI): A window system with a pointing device like mouse to direct I/O for choosing and selecting and a keyboard to enter text.
- ***Program execution***: The capability of the system to load a program into memory and to execute it. The program must be able to end its execution if necessary.
- ***I/O operations:*** While executing a program it may require performing I/O operations from a file or an I/O device. The operating system must perform such I/O operations.
- ***File-system services***: Read, write. Create and delete operations are common activity in a computing system. Searching for a given file, listing file information, accessing rights to users of files and directories are other activities.
- ***Communication services:*** Exchange of information among processes takes place on the same computer or between processes on different computer systems connected in a network. Such communications might happen through a shared memory or message passing.
- ***Error detection:*** Common errors that may occur are
  - CPU and memory hardware errors - memory error, power failure
  - I/O devices error – network connection failure, parity error on disk, out of paper in printer
  - User program error - arithmetic overflow, trying to access an illegal memory location

  Each type of error needs an appropriate action to ensure proper functioning. Under certain conditions error detection is possible.
- ***Resource allocation:*** Resources must be allocated to multiple users or multiple jobs running at the same time. Operating system efficient management of different types of resources like allocating

    - CPU cycles
    - main memory
    - file storage
    - I/O devices like printers, USB storage drives.

- ***Accounting:*** Keeps track of which users use how much and what kinds of computer resources for the purpose of billing and knowing usage statistics.
- ***Protection and security:*** Information that are stored in a multiuser or networked computer system needs protection and security. During concurrent processes execution, interfering into another should not be allowed. Securing the system requires each user to authenticate to the system, usually by means of a password, to gain access to system resources.

2.3 SYSTEM CALLS

System calls are routines normally written in C and C++, which provide an interface to the services made available by an operating system. Some low level tasks are written using assembly-language instructions. Example: A routine to read data from one file and copy them to another file. Steps are as follows:

- Naming the input file and the output file as the program to ask the user for the names by prompting message on the screen and then to read from the keyboard the characters that define the two files.
- Displaying a menu of file names in a window the user selecting the source name and destination name in order by using the mouse.

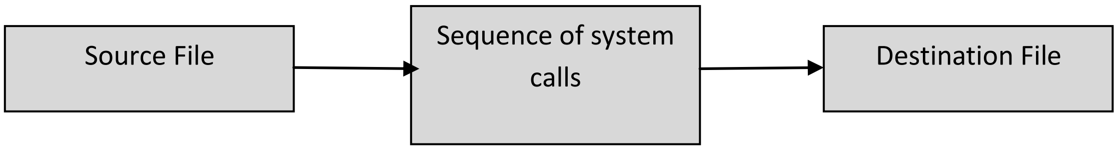


Fig 2.2 System Calls

Many system calls are required to perform these operations.(Figure 2.2) Some of them are listed below:

- Calls for possible error conditions for each operation
- Calls for the program to print a message on the console
- Calls for terminating abnormally under certain situations
- Calls for aborting from a program
- Calls for deleting an existing file

- Calls for creating a new file
- Calls for closing the input and out files
- Calls for normal termination.

Application Program Interface - Application developers design programs according to an Application Programming Interface (API) that specifies a set of functions with required parameters that are passed to each function for performing the task. Common APIs are:

- Windows API for Windows systems
- POSIX API for POSIX-based systems
- Java API for Java Virtual Machine.

An API is assessed through the library of code (.lib) provided by the operating system. For example, Windows function CreateProcess() invokes the NTCreateProcess() system call in the Windows kernel.

System call interface: In most of the programming languages, the run-time support system provides a system call interface that serves as the link to system calls made available by the operating system.

- The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.
- Each system call has a unique number associated to it, and the system-call interface maintains a table indexed according to these numbers.

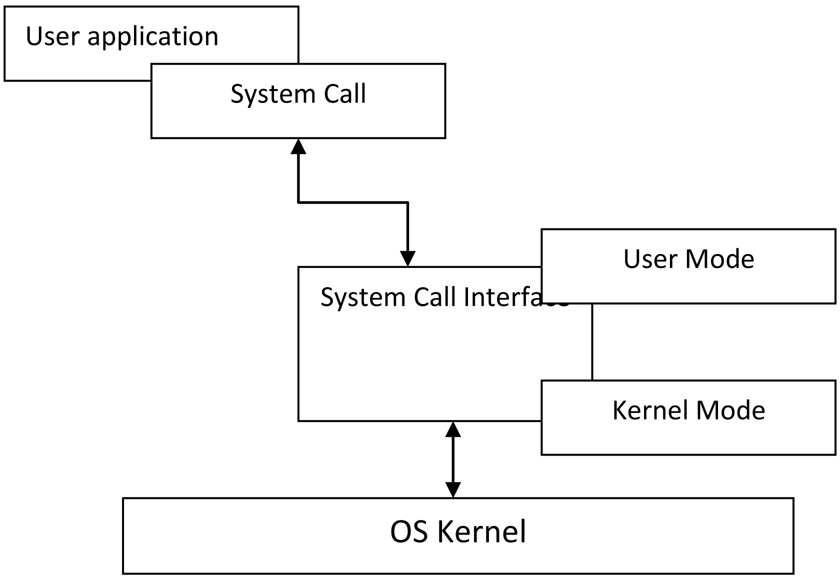


Fig 2.3 Links in OS

The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call. The link between an API, the system-call interface, and the operating system is shown in Figure 2.3

Methods used to pass parameters to the operating system are as follows:

- Pass the parameters in registers: the parameters stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register(E.g. Linux and Solaris.
- Parameters are placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system.
- The block or stack method that do not limit the number or length of parameters being passed.

Types of System calls:

- Process control
    - end, abort, load, execute
    - create process, terminate process
    - get process attributes, set process attributes
    - wait for time, wait event, signal event
    - allocate and free memory
- File manipulation
    - create file, delete file
    - open, close, read, write, reposition
    - get file attributes, set file attributes
- Device manipulation
    - request device, release device
    - read, write, reposition
    - get device attributes, set device attributes
    - logically attach or detach devices
- Information maintenance
    - get time or date, set time or date
    - get system data, set system data
    - get process, file, or device attributes
    - set process, file, or device attributes
- Communications

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

Some of the system calls in UNIX are open(), read(),write(), close(), sleep(), chmod(). Some of the system calls in Windows are CreateFile(), ReadFile(), WriteFile(), CloseHandle(), Sleep(), SetFileSecurity().

2.4 SYSTEM PROGRAMS

System programs, are called as **system utilities**. It provide a convenient environment for program development and execution. Example: A simple user interface to system calls. Various categories of System Programs are as follows:

- **File management:** Programs to create, delete, copy, rename, print, list, and manipulate files and directories.
- **Status information:** Programs for showing status such as system date, time, available memory or disk space, number of users, providing performance details, logging, and debugging information etc., Support of a **registry**, which is used to store and retrieve configuration information is also available.
- **File modification:** Text editors programs to create and modify the content of files stored, to search contents of files, to perform transformations of the text etc.,
- **Programming-language support:** Programs for Compilers, assemblers, debuggers, and interpreters for programming languages like C, C++ and Java.

- **Program loading and execution**: Programs for executing the assembled or compiled programs. Absolute loaders, re-locatable loaders, linkage editors, and overlay loaders are provided in systems.
- **Communications:** Programs for creating virtual connections among processes, users, and computer systems. It also includes programs to allow users to send messages to one another's screens, to browse Web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.
- **Background services:** Constantly running system-program processes like process schedulers, error monitoring services are known as **services**, **subsystems**, or daemons are provided as background services.

The categories of Application Programs include:

- Web browsing
- Word processing
- Text formatting
- Spreadsheets
- Database
- Games.

2.5 Operating System Structures

The operating system structure defines the interconnections and bindings of various components / modules into the kernel.

MS-DOS Structure

- In MS-DOS, the interfaces and levels of functionality are not well separated.

- Application programs are able to access the basic I/O routines.
- Limited by the hardware and its base hardware is accessible.
- Its simple structure leads MS-DOS vulnerable to malicious programs due to which the entire system crashes when user programs fail.

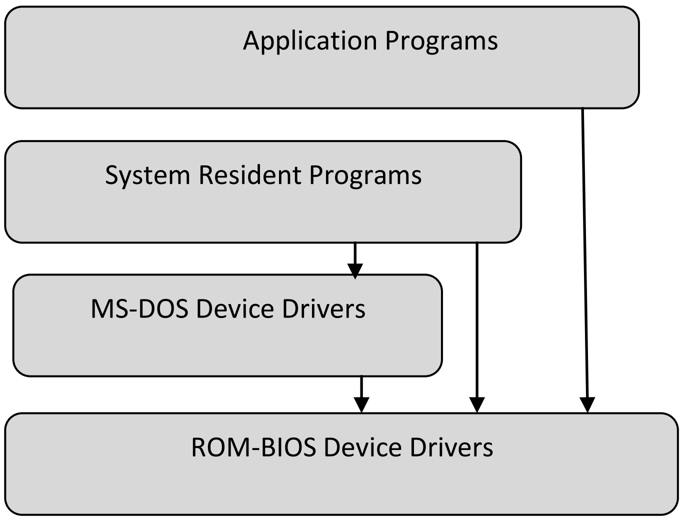


Fig. 2.4 MS-DOS system structure

UNIX Structure (Original)

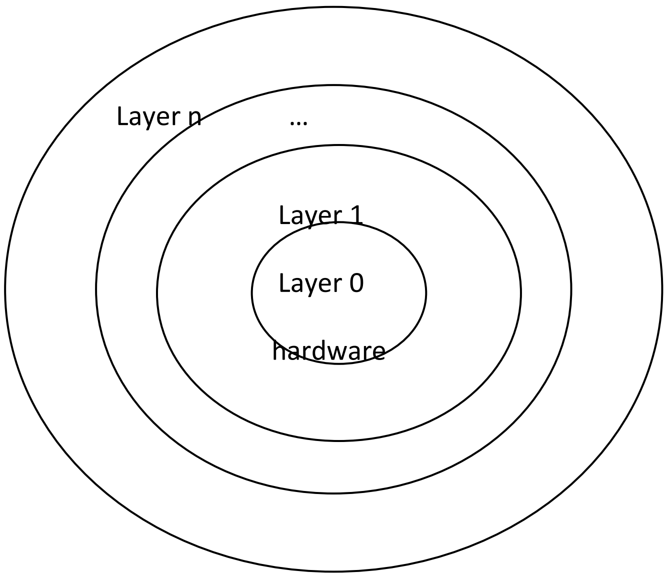
- Has two separable parts: the kernel and the system programs.
- The kernel is further separated into a series of interfaces and device drivers
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

- It had a distinct performance advantage, however: there is very little overhead in the system call interface or in communication within the kernel.

Users
Shell Commands, system libraries etc.,
System call interface to the kernel
KERNEL
kernel interface to the hardware
Hardware

Fig. 2.5 UNIX system structure

Layered Structure



2.6 PROCESS MANAGEMENT

Process, which is a program in execution. A process is the unit of work in a modern time-sharing system. A system has operating system processes for executing system code and user processes for executing user code. Both the processes can be executed concurrently. By switching the CPU between processes, the operating system can make the computer more efficient.

A batch system executes **jobs**, whereas a time-shared system has **user programs**, or **tasks**. Even on a single-user system, a user may be able to run several programs at one time: a word processor, a Web browser, and an e-mail package. A process is a program code that includes,

- the current activity, represented by the value of the **program counter**
- the contents of the processor's registers,
- the process **stack** containing temporary data
- a **data section**, containing global variables,
- a **heap**, which is memory that is dynamically allocated during process run time.

A process is an ***active*** entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program is a passive entity and it becomes a process when an executable file is loaded into memory. Two common techniques for loading executable files are
-- double-clicking an icon representing the executable file
-- entering the name of the executable file on the command line.

Even though two processes are associated with the same program, they are actually considered as two separate execution sequences. Its data, heap, and stack sections are not the same.

Process State : During process execution, it state changes. The state of a process is defined by its current activity. A process may be in one of the following states:

- **New**. The process is being created.
- **Running**. Instructions are being executed.
- **Waiting**. The process is waiting for some event to occur.
- **Ready**. The process is waiting to be assigned to a processor.
- **Terminated**. The process has finished execution.

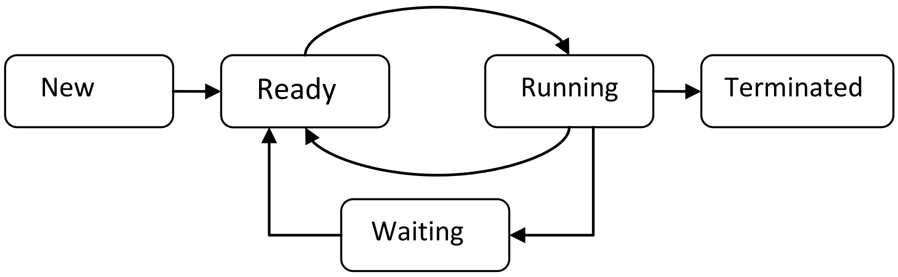


Fig 2.6 Process State Diagram

2.7 PROCESS CONTROL BLOCK

Each process is represented in the operating system by a **process control block (PCB)**—also called a **task control block**. A PCB contains many pieces of information associated with a specific process. (Fig 2.7)

Process state. The state may be new, ready, running, waiting, halted, and so on.

- **Program counter**. The counter indicates the address of the next instruction to be executed for this process.

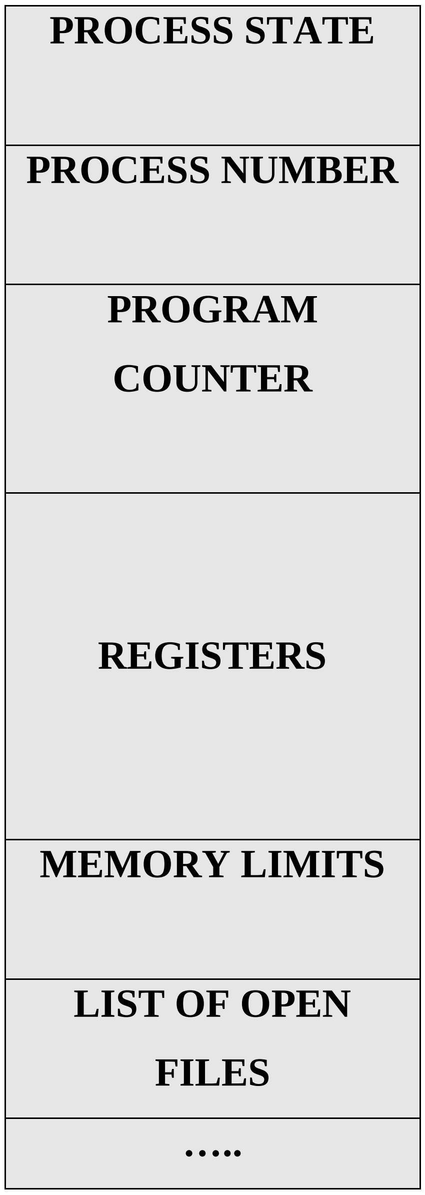


Fig 2.7 Process Control Block

- **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

- **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on. In brief, the PCB simply serves as the repository for any information that may vary from process to process.

THREADS

Processes are used to group resources together and threads are the entities scheduled for execution on the CPU.

- A thread is a single sequence stream within in a process also called as *lightweight processes*.
- A single thread of control allows the process to perform only one task at a time.
- A thread can be in any of several states
    - Running
    - Blocked
    - Ready
    - Terminated.

- Each thread has its own stack.
- A thread has
    - program counter (PC)
    - register set
    - stack space.
- Threads are not independent of one other like processes
- Threads shares their code section, data section, OS resources with other threads. Only one thread active (running) at a time.
- Threads within a process execute sequentially.
- If one thread is blocked, another thread can run.

2.8 PROCESS SCHEDULING

The objective of multiprogramming is to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so that the users can interact with each program while it is running. The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling. The process scheduler selects an available process for program execution on the CPU. In a multiprocessor environment, where there are many processes, some processes have to wait until the CPU is free and can be rescheduled. The objective of process scheduling system is,

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

The scheduler must apply appropriate rules for swapping processes ***in*** and ***out*** of CPU. Schedulers categories are,

- **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to give priority to another process, pre-empting the currently executing process.

Scheduling Queues

- ***Job Queue:***
    - All processes when enters into the system are stored in the **job queue**. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming.
- ***Ready Queue:***
    - Processes in the Ready state are placed in the **ready queue**.
- ***Device Queue:***
    - Processes waiting for a device to become available are placed in **device queues**.

The three types of schedulers are,

- ***Long Term Scheduler*** :
    - Decides which program must get into the job queue.
    - From the job queue, the Job Processor, selects processes and loads them into the memory for execution.
    - Long term scheduler runs less frequently.
    - The long-term scheduler controls the degree of multiprogramming i.e., the number of processes in memory.

    - The long-term scheduler selects a good process mix of I/O-bound and CPU-bound processes.
- ***Short Term Scheduler*** :
    - Also known as CPU Scheduler
    - Runs very frequently.
    - Enhances CPU performance and increases process execution rate.
- ***Medium Term Scheduler :***
    - Picks out big processes from the ready queue for some time, to allow smaller processes to execute.
    - The process is swapped out, and is later swapped in, by the medium-term scheduler
    - Reduces the number of processes in the ready queue.

Context Switch

A context switch is the process of storing and restoring the state of a process or thread so that execution can be resumed from the same point at a later time. The core of the operating system performs the following context switching activities with regard to processes including threads on the CPU:

- Suspending the progression of one process and storing the CPU's *state* (i.e., the context) for that process in memory,
- Retrieving the context of the next process from memory and restoring it in the CPU's registers
- Returning to the location indicated by the program counter in order to resume the process.

A context switch can also be described as the kernel suspending the progression of a process execution of one process on the CPU and resuming execution of some other process that had previously been suspended. Modes in Context switching

- **Kernel Mode:** Kernel mode is a privileged mode of the CPU in which only the kernel runs and which provides access to all memory locations and all other system resources. Context switches can occur only in kernel mode.
- **User Mode:** All other programs, including applications, initially operate in user mode.

Operations on Processes

The two major operatins performed on process are

- Process creation
- Process Termination

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing methods
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution modes
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Address space representation
  - Child duplicate of parent

    - Child has a program loaded into it

Process Termination

- ***Exit*** - Process executes last statement and asks the operating system to delete it
    - Output data from child to parent (**via wait**)
    - Process' resources are deallocated by operating system
- ***Abort*** - Parent may terminate execution of children processes under the following situations
    - Child has exceeded allocated resources
    - Task assigned to child is no longer required
    - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
    - All children terminated - cascading termination

2.9 INTERPROCESS COMMUNICATION

A process can either be independent or co-operate with other processes. Any process that shares data with other processes can be called as a cooperating process. Cooperating processes require an **interprocess communication (IPC)** mechanism for exchanging data and information between processes. There are several reasons for providing an environment that allows process cooperation:

- **Information sharing** – Providing an environment to allow concurrent access to the information to be shared by many users.
- **Computation speedup**. – Providing an environment to allow concurrent execution of subtasks to complete the task quickly.

- **Modularity** - Constructing the system in a modular fashion, by dividing the system functions into separate processes or threads
- **Convenience -** Providing an environment to individual users to work on many tasks simultaneously.

There are two fundamental models of interprocess communication:

- Shared momory
- Message passing

Shared memory

In this model, a region of memory is shared by cooperating processes. Processes can then exchange information by reading and writing data to the shared region. (Figure 2.8)

- Inter-process communication using shared memory requires communicating processes to establish a region of shared memory.
- A shared-memory region resides in the address space of the process creating the shared-memory segment.
- Processes must do the following
  - must attach to their address space.
  - can then exchange information by reading and writing data in the shared areas.
  - Determine the form of the data and the location independently without operating system's control.
  - responsible for ensuring that they are not writing to the same location simultaneously.
- A buffer will reside in a region of memory that is shared by the processes
- Two types of buffers used are

  - **unbounded buffer –** no limit on the size of the buffer.
  - **bounded buffer** - a fixed buffer size

***Advantage* -** Shared memory can be faster than message passing,

Message passing : In this model, communication takes place by exchanging messages between the cooperating processes. (Figure 2.9) Message-passing systems are typically implemented using system calls.

- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
- Useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.
- A message-passing facility provides at least two operations:
  - send(message)
  - receive(message)
- Messages sent by a process can be either fixed or variable in size. Sending fixed-sized messages is easy and variable-sized messages require a more complex.

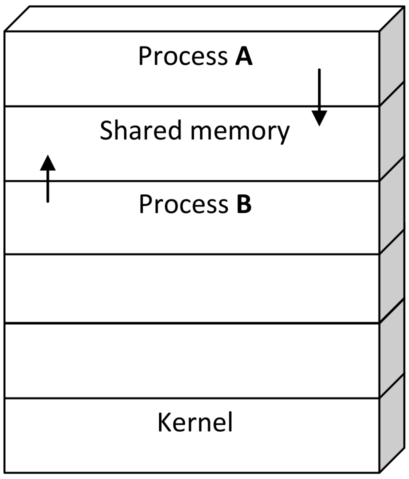


Fig 2.9 Message Passing

- A ***communication link*** must exist between sender and receiver. For logical link implementation, the following methods:
    - Direct or indirect communication
    - Synchronous or asynchronous communication
    - Automatic or explicit buffering

Advantages:

- Message passing is useful for exchanging smaller amounts of data
- Message passing is also easier to implement in a distributed system

Direct or indirect communication

Direct communication: Each process that wants to communicate must explicitly name the recipient or sender of the communication. For example,

- *send(P, message)*—Send a message to process P.
- *receive(Q, message)*—Receive a message from process Q.

Its properties are,

- A link is established automatically between every pair of process.
- The processes need to know only other's identity
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

Indirect communication**:** The messages are sent to and received from ***mailboxes***, or ***ports***.

- Each mailbox has a unique identification.
- Two processes can communicate only if they have a shared mailbox.
- A mailbox may be owned either by a process or by the operating system.

The send() and receive() operations are defined as follows:

- *send(A, message)*—Send a message to mailbox A.
- *receive(A, message)*—Receive a message from mailbox A.

Its properties are,

- A link is established only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, a number of different links may exist, with each link corresponding to one mailbox.

Synchronous or asynchronous communication

Message passing may be either synchronous and asynchronous also known as blocking or nonblocking.

• **Blocking send**. The sending process is blocked until the message is received.

• **Nonblocking send**. The sending process sends the message and resumes operation.

• **Blocking receive**. The receiver blocks until a message is available.

• **Nonblocking receive**. The receiver retrieves either a valid message or a null.

Automatic or explicit buffering

Messages exchanged by communicating processes reside in a temporary queue that can be implemented as

- Message system with no buffering: ***Zero capacity.***
    - The queue has a maximum length of zero;
    - the link cannot have any messages waiting in it.
    - sender must block until the recipient receives the message.
- Message system with automatic buffering: ***Bounded and unbounded capacity***.
    - ***Bounded capacity:***
        - The queue has finite length n.
        - If the queue is not full when a new message is sent, the sender can continue execution without waiting.
        - If the queue is full, the sender must block until space is available in the queue.
    - ***Unbounded capacity***:
        - The queue's length is infinite;
        - Any number of messages can wait in it.
        - The sender never blocks since space is always available in the queue.

* * *

UNIT III

3.1 PROCESS SCHEDULING – BASIC CONCEPTS

A program in execution can be called as Process. Process is not the same as program. A process is an 'active' entity whereas a program is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc. Systems consist of a collection of processes: operating-system processes execute system code, and user processes execute user code.

Process memory is divided into four sections for efficient working :

- The text section - made up of the compiled program code
- The data section - made up the global and static variables
- The heap - used for the dynamic memory allocation
- The stack - used for local variables

PROCESS STATE

Processes can be any of the following states :

- **New** - The process is in the stage of being created.
- **Ready** - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- **Running** - The CPU is working on this process's instructions.
- **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur.
- **Terminated** - The process has completed.

PROCESS CONTROL BLOCK

Each process has a Process Control Block, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.

- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.

PROCESS SCHEDULING

The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling. The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. To achieve this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.
Scheduler categories:

- **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to favour another process, pre-empting the currently executing process.

SCHEDULING QUEUES

- All processes when enters into the system are stored in the **job queue**.
- Processes in the Ready state are placed in the **ready queue**.
- Processes waiting for a device to become available are placed in **device queues**. There are unique device queues for each I/O device available.

Types of Schedulers

The three types of schedulers available are:

1. **Long Term Scheduler** :

Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

2. **Short Term Scheduler** :

This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

3. **Medium Term Scheduler** :

During extra load, this scheduler picks out big processes from the ready queue for some time, to allow smaller processes to execute, thereby reducing the number of processes in the ready queue.

OPERATIONS ON PROCESS

Process Creation

Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.

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

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

A child process may receive some amount of shared resources with its parent depending on system implementation. To prevent runaway children from consuming all of a certain system resource, child processes may or may not be limited to a subset of the resources 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.

Process Termination

By making the exit(system call), typically returning an int, processes may request their own termination. This int is passed along to the parent if it is doing a wait(), and is typically zero on successful completion and some non-zero code in the event of any problem.

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

- The inability of the system to deliver the necessary system resources.
- In response to a KILL command or other unhandled process interrupts.
- A parent may kill its children if the task assigned to them is no longer needed i.e. if the need of having a child terminates.
- If the parent exits, the system may or may not allow the child to continue without a parent (In UNIX systems, orphaned processes are generally inherited by init, which then proceeds to kill them.)

When a process ends, all of its system resources are freed up, open files flushed and closed, etc. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to init if the process already became an orphan.

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

THREAD

A thread is a flow of control within a process. Two kinds of threads are

- **User-level threads:** Visible to the programmer and are unknown to the kernel. These threads are faster to create and manage. Thread libraries provide the application programmer with an API for creating and managing threads. Three primary thread libraries are in common use: POSIX Pthreads,Windows threads, and Java threads.
- **Kernel-level threads:** the operating-system kernel supports and manages kernel-level threads. To create this thread, intervention from the kernel is required.

A multi-threaded process contains several different flows of control within the same address space. The benefits of multithreading include

- increased responsiveness to the user
- resource sharing within the process
- cost effectiveness
- more efficient use of multiple processing cores

THREAD MODELS

- The many- to-one model maps many user threads to a single kernel thread.
- The one-to-one model maps each user thread to a corresponding kernel thread.
- The many-to- many model multiplexes many user threads to a smaller or equal number of kernel threads.

CLASSICAL PROBLEM OF SYNCHRONIZATION

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

Bounded Buffer Problem

- This problem is generalised in terms of the Producer-Consumer problem.
- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

The Readers Writers Problem

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

Dining Philosophers Problem

- The dining philosopher’s problem involves the allocation of limited resources from 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 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.

CPU SCHEDULING

It is the basis of multi-programmed operating systems. By switching the CPU among processes, the operating system can make the computer more efficient. Multiprogramming environment have some process running at all times thereby maximizing CPU utilization. Since several processes are kept in memory at one time, when one process has to wait, the operating system takes the CPU away from that process and gives it to another process.

Scheduling of this kind can be called as CPU scheduling. Almost all computer resources can be scheduled before use.

CPU SCHEDULER

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

Some of the scheduling algorithms :an be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. The records in the queues are generally process control blocks (PCBs) of the processes.

Preemptive Scheduling

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

1. When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)

3. When a process switches from the waiting state to the ready state (for example, at completion of I/O)

4. When a process terminates.

When a process in ready state can be selected for execution then it is preemptive scheduling. A timer is needed for preemptive scheduling. A preemptive scheduling can result in ***race conditions*** when data are shared among several processes.

CPU-scheduling function has a **dispatcher** module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

• Switching context

• Switching to user mode

• Jumping to the proper location in the user program to restart that program

The dispatcher is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

Non-Preemptive Scheduling:

When scheduling takes place only under circumstances where a new process must be selected for execution then, the scheduling scheme is called as **nonpreemptive** or **cooperative**. Under non- preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

3.2 SCHEDULING CRITERIA

Normally, many objectives must be considered in the design of a scheduling algorithm. In particular, a scheduler should consider fairness, efficiency, response time, turnaround time, throughput, etc., These goals mostly depends on the system one is using (batch system/ interactive system/ real-time system, etc). the following goals are desirable in any of the systems. General Goals are,

Fairness - Fairness is important under all circumstances. A scheduler must ensure that each process gets its fair share of the CPU and no process should ever suffer from indefinite postponement.

Policy Enforcement - The scheduler has to enure that system's policy is enforced. For example, if the local policy is safety then the *safety control processes* must be enforced by all means.

Efficiency- Scheduler has to ensure that the system in particular the CPU is kept busy always whenever possible. If such efficiency is achieved, more work is extracted within short time.

Different CPU-scheduling algorithms have different properties, The properties of the various algorithms is the deciding factor in choosing the algorithm.

Many criteria have been suggested for comparing CPU-scheduling algorithms. Important criteria include the following:

• **CPU utilization** – Keeping the CPU as busy as possible. CPU utilization in a real system ranges from 40% to 90% for a lightly loaded system to a heavily loaded system respectively.

• **Throughput** – It is a unit for measure of work done by the CPU. The number of processes that are completed per unit time is called **throughput**. It may vary depending on the length of the process. For long processes, throughput may be one process per hour and for short transactions, it may be even many processes per second.

• **Turnaround time -** The time taken to execute a particular process is the turnaround time. The time interval is from the time of submission of a process to the time of completion. Turnaround time includes the following:

- Time spent waiting to get into memory.
- Waiting time in the ready queue.
- Executing time on the CPU.
- Performing I/O related operations.

• **Waiting time -** Waiting time is the sum of the periods spent waiting in the ready queue. It is only the amount of time that a process spends waiting in the ready queue.

• **Response time –** Response time is a measure of the time from the submission of a request to the first response is has produced. In general, it is the time taken to start responding and not the time it takes to output the response.

3.3 SCHEDULING ALGORITHMS

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU-scheduling algorithms. In this section, we describe several of them.

First-Come, First-Served Scheduling

First-Come-First-Served algorithm is the simplest scheduling algorithm. The process that requests the CPU first is allocated the CPU first with a FIFO queue for implementation.

Working functionality:

- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.
- The running process is then removed from the queue.

The code for FCFS scheduling is simple to write and understand. Processes are dispatched according to their arrival time on the ready queue. Being a non-preemptive discipline, once a process has a CPU, it runs to completion.

Example: Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process CPU Burst

Time P1 20 P2 5 P3

2

CPU Burst

Time P1 20 P2 5 P3

2

P1 20 P2 5 P3 2

P1 20 P2 5 P3 2

20 P2 5 P3 2

P2 5 P3 2

P2 5 P3 2

5 P3 2

P3 2

P3 2

2

Process arrival order : P3 → P2 → P1

Process Completion time:

Completion Time

20 *25* *27* P1 P2 P3

25 *27* P1 P2 P3

27 P1 P2 P3

P1 P2 P3

P1 P2 P3

P2 P3

P3

The waiting time is 0 milliseconds for process *P*1, 20 milliseconds for process *P*2, and 25 milliseconds for process *P*3. The average waiting time is, (0+ 20 + 25)/3 = 15 milliseconds. If the processes arrive in the order *P*1→ *P*3→ *P2* then the average waiting time will be 7/3 = 2.3 milliseconds. The average waiting time in FCFS may vary substantially if the processes' CPU burst times vary greatly.

If there is one CPU-bound process and many I/O-bound processes, there might occur a situation where, the CPU-bound process will get and hold the CPU. During this time, all the other processes will finish their I/O and will move into the ready queue, waiting for the CPU making the I/O devices idle.

When I/O bound processes (which have short CPU bursts) are executed quickly and moves back to the I/O queues, the CPU is idle. A **convoy effect** occurs where all the other processes wait for the one big process to get off the CPU. This results in lower CPU and device utilization.

Advantage: FCFS is more predictable since it offers time.

Disadvantages: It cannot guarantee good response time. The average waiting time is often quite long. Not suitable for time-sharing systems, where each user get a share of the CPU at regular intervals.

3.4 SHORTEST-JOB-FIRST SCHEDULING

In **Shortest-Job-First (SJF)** scheduling algorithm each process is associated with the length of the process's next CPU burst. The process that has the smallest next CPU burst is allocated. Example of SJF scheduling, consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process CPU Burst Time □P1 16 □P2 18 □P3 17 □P4 13 □

CPU Burst

Time □P1 16 □P2 18 □P

3 17 □P4 13 □

□P1 16 □P2 18 □P3 17
□P4 13 □

P1 16 □P2 18 □P3 17

16 □P2 18 □P3 17 □P4

13 □

□P2 18 □P3 17 □P4 13
□

P2 18 □P3 17 □P4 13

18 □P3 17 □P4 13 □

□P3 17 □P4 13 □

P3 17 □P4 13 □

17 □P4 13 □

□P4 13 □

P4 13 □

13 □

□

SJF Processing order :P2 → P3 → P1 → P4

Process Completion time:

Completion Time *13*

29 *46* *54* □P4 P1 P3
P2 □

29 *46* *54* □P4 P1 P3 P
2 □

46 *54* □P4 P1 P3 P2 □

54 □P4 P1 P3 P2 □

□P4 P1 P3 P2 □

P4 P1 P3 P2 ▯

P1 P3 P2 ▯

P3 P2 ▯

P2 ▯

▯

The waiting time is 13 milliseconds for process *P*1, 46 milliseconds for process *P*2, 29 milliseconds for process *P*3, and 0 milliseconds for process *P*4. Thus, the average waiting time is (13 + 46 +29 + 0)/4 = 22 milliseconds. By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be (0+16+34+51)/4 = 25.3 milliseconds.

Advantages:

- An optimal algorithm that gives minimum average waiting time for a given set of processes.
- decreases the waiting time of the short process more than it increases the waiting time of the long process.
- Average waiting time

decreases.

Disadvantages:

- knowing the length of the next CPU request is difficult.
- it cannot be implemented at the level of short-term CPU scheduling where the length of the next CPU burst is unknown.

3.5 ROUND ROBIN

The **round-robin (RR)** scheduling algorithm is designed for timesharing systems.

- Preemption is added to enable the system to switch between processes.
- A small unit of time, called a **time quantum** or **time slice**, which ranges from 10 to 100 milliseconds in length is defined.
- The ready queue (FIFO) is treated as a circular queue.
- New processes are

added to the tail of the ready queue.

Working functionality

- The CPU scheduler allocates the CPU to each process for a time interval of up to 1 time quantum.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- CPU burst < 1 quantum time:
    - If the process has a CPU burst of less than 1 time quantum, the process itself will release the CPU automatically.
    - The scheduler will then proceed to the next process in the ready

queue.

- CPU burst > 1 quantum time:
    - If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system.
    - A context switch will be executed, and the process will be put at the tail of the ready queue.
    - The CPU scheduler will then select the next process in the ready queue.

Disadvantage : The average waiting time is often long.

Example:

Consider the following set of processes that arrive at time 0, with the length of the CPU burst
given in milliseconds:

Process CPU Burst

Time □P1 21 □P2 2 □P3 3 □

CPU Burst

Time □P1 21 □P2 2 □P3 3 □

□P1 21 □P2 2 □P3 3 □

P1 21 □P2 2 □P3 3 □

21 □P2 2 □P3 3 □

□P2 2 □P3 3 □

P2 2 □P3 3 □

2 □P3 3 □

□P3 3 □

P3 3 □

3 □

□

RR Processing order :P3 → P2 → P1

Time quantum 3 milliseconds

Process P1 gets the first 3 milliseconds and it needs another 18 milliseconds before which it is preempted and the CPU is given to the

P2, Process P2 does not need 3 milliseconds, so it quits before its time quantum expires. The CPU is then given to the next process, process P3. Once each process has received 1 time quantum, the CPU is returned to process P1
for an additional time quantum. The resulting RR schedule is as follows:

3 5 8 11 14 17 20 21
□P1 P2 P3 P1 P1 P1
P1 P1 □
5 8 11 14 17 20 21 □
8 11 14 17 20 21 □P1
P2 P3 P1 P1 P1 P1
11 14 17 20 21 □P1 P
2 P3 P1 P1 P1 P1 P1
□
14 17 20 21 □P1 P2 P
3 P1 P1 P1 P1 P1 □
17 20 21 □P1 P2 P3 P
1 P1 P1 P1 P1 □
20 21 □P1 P2 P3 P1 P
1 P1 P1 P1 □
21 □P1 P2 P3 P1 P1 P
1 P1 P1 □
□P1 P2 P3 P1 P1 P1 P
1 P1 □

P1 P2 P3 P1 P1 P1 P1

P1 ⍰

P2 P3 P1 P1 P1 P1 P1

⍰

P3 P1 P1 P1 P1 P1 ⍰

P1 P1 P1 P1 P1 ⍰

P1 P1 P1 P1 ⍰

P1 P1 P1 ⍰

P1 P1 ⍰

P1 ⍰

The average waiting time for this schedule is P1 waits for 5 milliseconds, P2 waits for 3 milliseconds, and P3 waits for 7 milliseconds.

Thus, the average waiting time is 15/3 = 5 milliseconds.

The performance of the RR algorithm depends on the size of the time quantum. If the time quantum is large, it behaves like FCFS policy. If the time quantum is small, it needs large number of context switches.

3.6 ALGORITHM EVALUATION

The criteria for selecting an algorithm is normally based on CPU utilization, response time, or throughput. Several measures, are considered to select an algorithm.

- Maximizing CPU utilization under the constraint that the maximum response time is 1 second
- Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time.

Algorithms are evaluated under these considerations. Various evaluation
methods used are:

- Deterministic Modeling
- Queueing Models
- Simulations

Deterministic Modeling

Deterministic modeling is one type of analytic evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number to evaluate the performance of the algorithm for that workload. It takes a particular predetermined workload and defines the performance of each algorithm for that workload.

For example, evaluation can be carried out for FCFS, SJF and RR, given the same set of workload, (i.e. the number of processes and the processing time are same for all) and determining the minimum average waiting time.

- Deterministic modeling is simple and fast.
- Given exact numbers for input, it gives exact results to compare the algorithms.
- Can be mainly used for evaluating scheduling algorithms.

Queuing Models

Queuing network analysis - The computer system can be described as a network of servers. Each server has a queue of waiting processes. The CPU is a server with its ready queue, and the I/O system is with its device queues. Knowing arrival rates and service rates, we can compute utilization, average queue length, average wait time, and so on. This analysis is called **queuing-network analysis**.

As an example, let *n* be the average queue length and ‘α’ be the average waiting time in the queue, and let β be the average arrival rate for new processes in the queue. During the time ‘α’ that a process waits, β x α new processes will arrive in the queue. If the system is in a steady state, then the number of processes leaving the queue must be equal to the number of processes that arrive. Thus, $n = \beta \times \alpha$. This equation, known as **Little’s formula**, is useful for any scheduling algorithm and arrival distribution. This formula can be used to compute one of the three variables if the other two are known.

Queueing analysis can be useful in comparing scheduling algorithms, with some limitations. Queueing models are often only approximations of real systems.

Simulations

Running simulations is by programming a model of the computer system. Software data structures represent the major components of the system.

Simulator: The simulator has a variable representing a clock. As time goes on, the simulator modifies the system state to reflect the activities of the devices, the processes, and the scheduler. As the simulation executes, statistics that indicate algorithm performance are gathered and printed. The data to drive the simulation can be generated using the following methods:

- ***random-number generator:*** The data for simulation can be collected using a random-number generator. It is programmed to generate processes, CPU burst times, arrivals, departures, etc., according to probability distributions. The results define the distribution of events in the real system; this distribution can then be used to drive the simulation.
- ***Trace tapes*:** Trace tapes are created by monitoring the real system and recording the sequence of actual events. We then use this sequence to drive the simulation. Trace tapes provide an excellent way to compare two algorithms on exactly the same set of real inputs. This method can produce accurate results for its inputs.

In general, analytic methods use mathematical analysis to determine the performance of an algorithm. Simulation methods determine performance by simulating the scheduling algorithm on a sample of processes and computing the resulting performance. Normally, simulation can provide an approximation of actual system performance. The only reliable technique for evaluating a scheduling algorithm is to implement the algorithm on an actual system and monitor its performance in a real-world environment.

3.7 PROCESS SYNCHRONIZATION

Concurrent or parallel execution can contribute to issues involving the integrity of data shared by several processes. Process Synchronization means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

A situation, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition**.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are listed below:

- Critical Section Problem
- The Dining-Philosophers Problem
- The Bounded-Buffer Problem
- The Readers-Writers Problem

3.8 CRITICAL SECTION PROBLEM

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

- ***Entry section:*** The code preceding the critical section, and which controls access to the critical section, is termed the entry section.
- ***Critical Section:*** The code between the entry and exit section.
- ***Exit Section:*** The code following the critical section is termed the exit section.
- ***Remainder section:*** The rest of the code not included in either the critical section or the entry or exit sections is termed the remainder section.

The general structure of a typical process can be illustrated as follows:

```
do
   {

                    Critical
section

              Remainder
section
} while (true);
```

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

- **Mutual Exclusion:** Among a group of cooperating processes, only one process can be in its critical section at a given point of time.

- **Progress:** If critical section has no process and other threads wants to execute its critical section then any one of these threads must be allowed to get into its critical section.
- **Bounded Waiting:** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections before that request is granted.

Two general approaches that are used to handle critical sections are,

- ***Preemptive kernels***: A preemptive kernel allows a process to be preempted while it is running in kernel mode. Preemptive kernels are especially difficult to design for SMP architectures, since it is possible for two kernel-mode processes to run simultaneously on different processors.
- ***Nonpreemptive kernels***: Only one process is active in the kernel at a time. A nonpreemptive kernel is free from race conditions on kernel data structures. It does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

3.9 PETERSON'S SOLUTION

Peterson's Solution is a software-based solution to the critical section problem. Peterson's solution is based on two processes, P_0 and P_1, which alternate between their critical sections and remainder sections. When current process is P_i, the other process is denoted by P_j.

```
do

   {

         flag[i] = true;
         turn = j;
         while (flag[j] && turn == j);

                    Critical section

          flag[i] = false;

                    Remainder section

} while (true);
```

Peterson's solution requires two shared data items:

- **int turn** - indicates whose turn it is to enter into the critical section.
  If turn == i, then process i is allowed into their critical section.
- **boolean flag[i]** - indicates when a process wants to enter into their critical section. When process i wants to enter their critical section, it sets flag[i] to true.

In the entry section, process i first raises a flag indicating a desire to enter the critical section. Then ***turn*** is set to ***j*** to allow the ***other*** process to enter their critical section if process j so desires. The while loop is a busy loop, which makes process i wait as long as process j has the turn and wants to enter the critical section. Process i lowers the ***flag[i]*** in the exit section, allowing process j to continue if it has been waiting.
To prove the correctness of the Peterson’s solution, the following conditions must be met:

- Mutual exclusion is preserved.
- The progress requirement is satisfied.
- The bounded-waiting requirement is met

Peterson’s solution is a software-based approach and is not guaranteed to work on modern computer architectures.

3.10 SYNCHRONIZATION HARDWARE

Locking mechanism is applied to generalize the solutions. Each process while entering its critical section must set a ***lock***, to prevent other processes from entering critical sections simultaneously. The process must release the lock when exiting their critical section so that other processes can proceed.

Setting a lock is possible only if no other process has already set it. Some hardware solutions can be provided.

- The first approach is to prevent a process from being interrupted while in their critical section. This approach is taken by non preemptive kernels. This approach is not suitable for multiprocessor environments.

- The second approach is to provide some ***atomic*** operations that guarantees to operate as a single instruction, without interruption. The ***"Test and Set"***, is one such operation that simultaneously sets a boolean lock variable and returns its previous value. TestandSet( ) can be defined as follows:

```
boolean TestandSet(Boolean * target)
{
    boolean ret = *target;
    *target = TRUE;
   Return ret;
}
```

The mutual exclusion implementation of TestandSet( ) by declaring a boolean variable lock, initialized to false can be described as follows:

```
do

    {

             while (TestandSet(&lock))
             ;
```

```
                  Critical section

          lock = false;

          Remainder section

} while (true);
```

This hardware solution is difficult for ordinary programmers to access, especially on multi-processor machines. Also it is often platform-dependent.

3.11 MUTEX LOCKS

Mutex lock is a software based approach to the critical section problem. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section no other process can gain access to it. It is a software API equivalent called ***mutex locks*** or ***mutexes.*** (meaning mutual exclusion) when used acquires a lock prior to entering a critical section, and releases it when exiting.

```
do

   {

          acquire lock;

                 Critical section

          release lock;

                 Remainder section

} while (true);
```

The definition of ***acquire( )*** and ***release( )*** are as follows:

```
        acquire( )
   {
        while (!available)
        ; /* busy wait */
        available = false;;
         }

   release( )
```

```
{
available = true;
}
```

Busy waiting: It is a state in which a process is in its critical section, any other process trying to enter its critical section must loop continuously in the call to ***acquire( )***. Busy waiting wastes CPU cycles.

***Spinlock*:** When a mutex lock is set, the process "spins" while waiting for the lock to become available. When locks are expected to be held for short times, spinlocks are useful.

3.12 SEMAPHORES

Semaphore S, is an integer variable for which only two standard atomic operations are defined, the wait() and signal( ) operations, The wait() operation is termed as P and signal() is called as V. The definition of ***wait( )*** and ***signal( )*** are as follows:

```
wait(S)
{
while (S <= 0)
; // busy wait
S--;
}

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

When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. The two basic types of semaphores are,

- ***Counting semaphores***: The value of a counting semaphore can range over an unrestricted domain. Counting semaphores can be used to control access to a given resource consisting of a finite number of instances.
- ***Binary semaphores***. The value of a binary semaphore can range only between 0 and 1. Binary semaphores acts like mutex locks.

The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a ***wait( )*** operation on the semaphore which decrements the count. When a process releases a resource, it performs a ***signal( )*** operation which increments the count. When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count is greater than 0. Semaphores can be used to solve various synchronization problems.

In general, the important properties of semaphores are,

- Works with many processes
- Can have many different critical sections with different semaphores
- Can permit multiple processes into the critical section at once, if needed

The major drawback of semaphore is, if not properly used it might lead towards a situation called Deadlock.

3.13 THE DINING PHILOSOPHERS PROBLEM

The *Dining Philosophers* problem is a classic synchronization problem. There are five philosophers sitting around a table, in which there are five chopsticks 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. The dining philosopher's problem involves the allocation of limited resources from a group of processes in a deadlock-free and starvation-free manner.

Problem statement: There are five philosophers sitting around a table, in which there are five chopsticks 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.

A philosopher can only eat when he has both left and right chopsticks. Each chopstick can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After he finishes eating, he needs to put down both chopsticks so they become available to others. A philosopher can take the fork on his right or the one on his left as they become available, but cannot start eating before getting both of them.

The problem: Design a discipline of behaviour (an algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.

Challenge: A deadlock situation

A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore. The chopstick is released by executing the signal() operation on the appropriate semaphores. The procedure adopted can be illustrated as follows:

```
semaphore chopstick[5];

        do
```

```
{
wait(chopstick[i]);
wait(chopstick[(i+1) % 5]);
. . .
/* eat for awhile */
. . .
signal(chopstick[i]);
signal(chopstick[(i+1) % 5]);
. . .
/* think for awhile */
. . .
} while (true);
```

This solution guarantees that no two neighbors are eating but it could create a deadlock. Suppose that all five philosophers become hungry at the same time and each grabs the left chopstick. All the elements of chopstick will now be equal to 0. When each philosopher tries to grab the right chopstick, it delays the process forever.

Solution to deadlock problem is possible by imparting some restrictions, like

- Allowing at most four philosophers to be sitting simultaneously at the table.
- Allowing a philosopher to pick up the chopsticks only if both chopsticks are available.
- Allowing an odd-numbered philosopher to pick up the left chopstick first and then the right chopstick, whereas an even numbered philosopher picks up the right chopstick and then the left.

Resource starvation situation:

It might also occur independently of deadlock if a particular philosopher is unable to acquire both chopsticks because of a timing problem.

Suppose is a rule is adopted that the philosophers put down a fork after waiting ten minutes for the other fork to become available and wait a further ten minutes before making their next attempt. This scheme eliminates the possibility of deadlock but suffers from the problem of **livelock**.

If all five philosophers appear in the dining room at exactly the same time and each picks up the left fork at the same time the philosophers will wait ten minutes until they all put their forks down and then wait a further ten minutes before they all pick them up again.

The *Dining Philosophers* problem clearly explains the situation where multiple processes access sets of data that are being updated. Systems such as operating system kernels use thousands of locks and synchronizations that require strict adherence to methods and protocols if such problems as deadlock, starvation, or data corruption are to be avoided.

* * *

UNIT IV

4.1 DEADLOCKS

It is a situation in which several processes compete for the available of resources and the process could not complete its job. Consider a situation where, a process - P1, request for a resource. If the requested resource is not available at that time, the process enters a waiting state. If the requested resource is held by other waiting process- P2, then P1 which is waiting can never change from its waiting state. This situation is termed as deadlock.

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

System Model

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources may be partitioned into several types (or classes), each consisting of some number of identical instances. CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples of resource types. If a system has two CPUs, then the resource type *CPU* has two instances. Similarly, the resource type *printer* may have five instances.

If a process requests an instance of a resource type, the allocation of any instance of the type should satisfy the request. If it does not, then the instances are not identical, and the resource type classes have not been defined properly.

For example, a system may have two printers. These two printers may be defined to be in the same resource class if no one cares which printer prints which output. However, if one printer is on the ninth floor and the other is in the basement, then people on the ninth floor may not see both printers as equivalent, and separate resource classes may need to be defined for each printer.

A lock is typically associated with protecting a specific data structure—that is, one lock may be used to protect access to a queue, another to protect access to a linked list, and so forth. For that reason, each lock is typically assigned its own resource class, and definition is not a problem.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system. In other words, a process cannot request three printers if the system has only two.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

> **1. Request**. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
>
> **2. Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
>
> **3. Release**. The process releases the resource.
>
> The request() & release() device, open() & close() file, allocate() & free() memory are system calls.

For each use of a kernel-managed resource by a process or thread, the operating system checks to make sure that the process has requested and has been allocated the resource. A system table records whether each resource is free or allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource. A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.

4.2 DEADLOCK CHARACTERIZATION

Features that characterize deadlocks are normally because of the following conditions held simultaneously in a system:

> **1. Mutual exclusion**. At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
>
> **2. Hold and wait**. A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
>
> **3. No preemption**. Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait. A set {*P*0, *P*1, ..., *Pn*} of waiting processes must exist such that *P*0 is waiting for a resource held by *P*1, *P*1 is waiting for a resource held by *P*2, ..., *Pn*−1 is waiting for a resource held by *Pn*, and *Pn* is waiting for a resource held by *P*0.

All four conditions must hold for a deadlock to occur and these four conditions are not completely independent.

4.3 METHODS FOR HANDLING DEADLOCKS

Handling deadlock problem using one of the following methods:

• Using a protocol to prevent or avoid deadlocks, ensuring that the system will ***never*** enter a deadlocked state.

• allowing the system to enter a deadlocked state, detect it, and recover.

• Ignoring the deadlock problem completely and pretending as if deadlocks had never occurred in the system. It is then up to the application developer to write programs that handle deadlocks.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme. **Deadlock prevention** provides a set of methods to ensure that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

Deadlock avoidance requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime. With this additional knowledge, the operating system can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

4.3.1 Deadlock Prevention

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

are the methods employed to prevent deadlock.

Mutual Exclusion: The mutual exclusion condition must hold indicating that at least one resource is non-sharable. A process never needs to wait for a sharable resource. Sharable resource, do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are sharable resources which attempted by many processes to open a read-only file simultaneously, they can be granted simultaneous access to the file. A mutex lock is a non-sharable resource that cannot be simultaneously shared by several processes.

Hold and Wait

In order to prevent from the occurrence of the hold-and-wait condition in a system, it must be ensured that, a process requesting for a resource, must not hold any other resources. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated. This can be implemented by two methods:

- Ensuring that system calls requesting resources for a process precede all other system calls.
- Allowing a process to request resources only when it has none.

  Disadvantages:
- Resource utilization may be low.
- Starvation is possible making at least one of the resources that a process needs is always allocated to some other process.

No Preemption

A necessary condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, the following methods can be adopted.

Method 1:

If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources the process is currently holding are implicitly released. These preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only if it can regain its old resources, as well as the new ones that it has requested.

Method 2:

If a process requests some resources, check for its availability and allocate it if available. If not, check if it is allocated to some other process that is waiting for additional resources. If so, preempt the desired resources from the waiting process and allocate them to the requesting process.

If the resources are neither available nor held by a waiting process, the requesting process must wait. A process can be restarted only when it is allocated the requested new resources and recovers any resources that were preempted while it was waiting.

Circular Wait

To ensure that circular wait condition never holds, enforce a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration.

Ensuring that resources are acquired in the proper order is the responsibility of application developers. Software can also be used to verify that locks are acquired in the proper order and to give appropriate warnings when locks are acquired out of order.

Low device utilization and reduced system throughput are the major drawback of deadlock prevention methods.

4.3.2 Deadlock Avoidance

In order to avoid deadlock the system must consider the following:

- the resources currently available,
- the resources currently allocated to each process,
- the future requests and releases of each process.

System models require a priori information about the number of resources it may need. With this prior information, it is possible to construct an algorithm that ensures that the system will never enter a deadlocked state.

The resource allocation ***state*** is defined by the number of available and allocated resources and the maximum demands of the processes. A state space can be in one of the following states:

- deadlocked state
- safe state
- unsafe state

A state is *safe* if the system can allocate resources to each process in some order and still avoid a deadlock. A deadlock-avoidance algorithm dynamically examines the resource-allocation state to avoid deadlock. Two important deadlock avoidance algorithms are

- Resource allocation Graph algorithm
- Bankers algorithm

4.4 RECOVERY FROM DEADLOCK

The two methods employed for recovering from deadlocked state are,

- Process termination
- Resource Preemption

Process termination: This techniques adopts one of following the strategies:

__a. Abort all deadlocked processes__: Will break the deadlock cycle, with high computational overhead. The processes that are computed for a long time, as well as the results of these partial computations are discarded which needs recomputation at a later stage.

b**. **Abort one process at a time until the deadlock cycle is eliminated: After aborting each process, the deadlock-detection algorithm is invoked to find out if there are other processes in deadlock state.

The factors that affect in choosing the process for aborting includes:

- the number of processes to be terminated
- the type of the process (interactive / batch)
- priority of the process
- the time needed to complete the chosen task for
- the types of resources used by the process
- the required resources for the process to complete it

Resource Preemption: This method adopts the following strategies:

a. Selecting a victim: the process must be chosen considering parameters like, the number of resources a deadlocked process is holding and the amount of time the process has consumed so far.

b. Rollback: To recover the process with pre-empted resources, roll back the process to some safe state and restart it from that state. Rolling back the process only as far as necessary requires the system to keep information about the state of all running processes. Therefore a total rollback will be best method that will abort the process and then restart it.

c. Starvation: Resources should not always be preempted from the same process. If done so, that process will never complete its task, leading to a starvation situation, To avoid such a situation, it must be ensured that a process can be picked as a victim few times only. Including the number of rollbacks is a solution.

4.5 MEMORY MANAGEMENT STRATEGIES

Memory is most important for operating modern computer system. Memory consists of array of bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter.

The general-purpose storage that the CPU can access directly are:

- Main memory
- Registers.

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

Any instructions or data for execution must be in main memory or register. If the data are not in memory, it must be moved before CPU execution. Main memory is accessed through a transaction on the memory bus. Registers that are built into the CPU are generally accessible within one cycle of the CPU clock.

Cache is a fast memory that lies between the CPU and main memory, which is present on the CPU chip for fast access.

- Each process must have a separate memory space to protect the processes from each other.
- The memory spaces have distinct legal addresses that the process may access.
- The base register and the limit register are the two registers which protects the address space.
- The base and limit registers can be loaded only by the operating system,
- The base register holds the smallest legal physical memory address
- The limit register specifies the size of the range. For example, if the base register holds 3020 and the limit register is 1200, then the program can legally access all addresses from 3020 through 4219.

- Any attempt by a program executing in *user mode* to access operating-system memory results in a fatal error thus preventing a user program from modifying the content of the operating system.
- The operating system can change the value of the registers but prevents user programs from changing its contents.
- An unrestricted access if given to the operating system, executing in kernel mode, to OS memory and users' memory.
- Virtual address is also known as Logical address and is generated by the CPU.
- Physical address is the address that actually exists on memory.

Dynamic Loading: All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory. If a certain part or routine of the program alone is loaded into the main memory when it is called by the program, it is called as Dynamic Loading.

Dynamic Linking*:* If one program is dependent on other program, instead of loading all the dependent programs, CPU can link the dependent programs dynamically to the main executing program when it is required. This mechanism is called as Dynamic Linking.

Swapping: If there is not enough main memory space to hold all the currently active processes in a timesharing system, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

Address Binding: A program on a disk is in the form of a binary executable file. For execution, the program must be brought into memory and placed within a process. An input queue is formed if the processes on the disk are waiting to be brought into memory for execution.

Single task Procedure: It selects one of the processes from the input queue and loads that process into memory. During execution of process, it accesses instructions and data from memory. When the process terminates, its memory space is declared available.

Normally, a user process may reside in any part of the physical memory starting from the first address in address space of the computer say 00000.

Address representation: An address may be represented in different ways. Addresses in the source program are generally symbolic.

- A compiler binds these symbolic addresses to relocatable addresses.
- The linkage editor or loader in turn binds the relocatable addresses to absolute addresses.
- Each binding is a mapping from one address space to another.
- During compile time Absolute code is generated
- During load time relocatable code is generated.

4.6 CONTIGUOUS MEMORY ALLOCATION

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

Memory protection, Memory allocation, Fragmentation, Segmentation and Paging are some of the important concepts considered in Contiguous memory allocation.

Memory Protection

- Memory protection controls memory access rights on a computer. It prevents a process from accessing memory that has not been allocated to it. It also prevents a bug within a process from affecting other processes, or the operating system itself. The relocation and limit registers protect both the operating system and the other users programs and data from being modified by this running process.
- The CPU scheduler selects a process for execution
- The dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- Every address generated by a CPU is checked against the relocation and limit registers

The advantage of using relocation-register scheme is, it provides an effective way to allow the operating system's size to change dynamically.

Transient operating-system code: The operating system contains code and buffer space for device drivers which is not always kept in memory. As needed, it enters and leaves the memory changing the size of the operating system during program execution.

Memory Allocation

Memory allocation is a process by which computer programs are assigned memory or space. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate. The three types of memory allocation are:

a. First Fit: The first hole that is big enough is allocated to program. Searching can start from the beginning, or at the location where the previous first-fit search ended. Searching is stopped as soon as a free hole that is large enough is found

b. Best Fit: The smallest hole that is big enough is allocated to program. smallest hole that is big enough. If the entire list, is ordered by size searching is easier. This method produces the many smallest leftover holes after all allocations.

c. Worst Fit: The largest hole that is big enough is allocated to program. The entire list, must be searched, it is not sorted by size. This method produces the largest leftover hole, which may be better than a best-fit method.

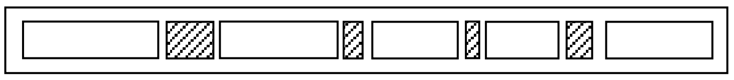
Fragmentation

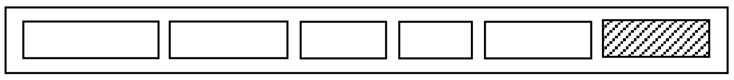
Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is a situation where these small blocks of available memory cannot be used for any further memory allocation.

External fragmentation: When processes are loaded and removed from the memory. There might exist small fragments. The existing free holes are non contiguous i.e. the memory is fragmented into large number of small holes also known as External Fragmentation.

Internal fragmentation: If the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as Internal fragmentation i.e. the memory that is internal to a partition but is of no use.

Compaction: It is a solution to the problem of external fragmentation. Shuffling the memory contents to place all free memory together in one large block is the technique applied in compaction. The simplest compaction algorithm moves all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory which involves lot of overhead.


Fragmented memory before compaction


After Compaction

Segmentation

Segmentation is a memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

Paging

Paging is a solution to fragmentation problem that allows the physical address space of a process to be non-contagious. The physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

Page Table

A Page Table is the data structure that stores the mapping between *virtual address* and *physical addresses.* It is used by a virtual memory system in a computer operating system.

Segmentation with Paging

In this scheme each segment is divided into pages and each segment is maintained in a page table. It combines the features of both paging and segmentation for further improvement. It is also known as 'Page the Elements' scheme. In this scheme, the logical address is divided into following three parts:

- Segment numbers(S)

- Page number (P)
- The displacement or offset number (D)

CPU Scheduling

CPU scheduling process allows one process to use the CPU while the execution of another process is on hold state / waiting state due to unavailability of resource. It makes use of CPU fully. The aim of CPU scheduling is to make the system fast and efficient.

Scheduling Criteria

Some of the CPU scheduling criteria are listed below:

- **CPU utilization :** To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time. Considering a real system, CPU usage should range from 40% to 90% for a lightly loaded to heavily loaded system respectively.
- **Throughput:** It is the total number of processes completed per unit time i.e., the total amount of work done per unit time which may range widely depending on the specific processes.
- **Turnaround time**: It is the amount of time taken to execute a particular process, i.e. The duration from time of submission of the process to the time of completion of the process.
- **Waiting time**: The amount of time a process has been waiting in the ready queue to get control on the CPU.
- **Load average**: The average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- **Response time**: The duration between the submission of a request and the first response produced.

Normally, in an efficient system, CPU utilization and Throughput must be maximized and other factors must be reduced for proper optimization.

4.7 SEGMENTATION

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

- When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.
- Segments are of variable-length where as in paging pages are of fixed size.
- A program segment contains the program's main function, utility functions, data structures etc.,
- Segments are numbered and are referred to by a segment number, rather than by a segment name. The logical address consists of segment-number and the offset. The segment number is used as an index to the segment table. The offset ***d*** of the logical address must be between 0 and the segment limit.

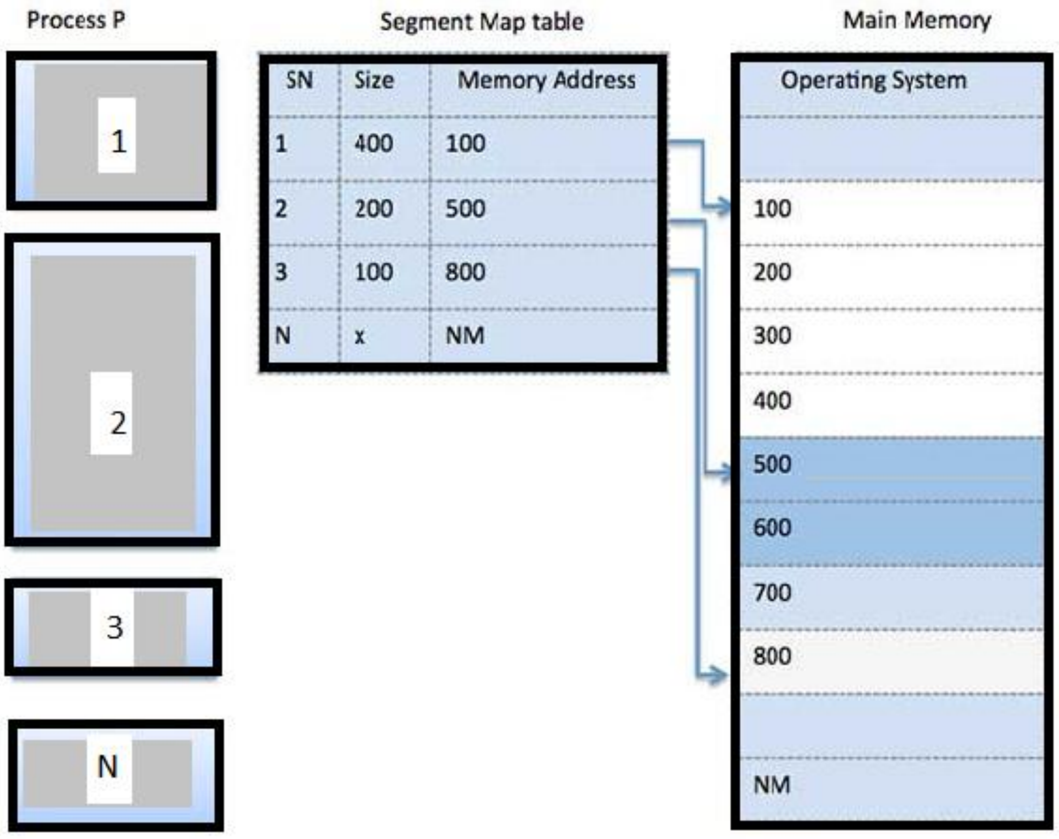


Fig 4.1 Segmentation

Segment map table

The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. (Fig 5.1) For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

- The segment table is an array of base–limit register pairs.
- Mapping the user-defined addresses into physical address is done with the help of the segment table.
- Each entry in the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.
- The correct offset is added to the segment base to produce the address in physical memory of the desired byte.

  Advantage: Segmentation permits the physical address space of a process to be noncontiguous.

  Disadvantage: Has External fragmentation

4.8 PAGING

Pages: Paging is a memory management technique in which process address space is broken into blocks of the same size called pages. The size of the pages is in power of 2, between 512 bytes and 8192 bytes. The size of the process is measured in the number of pages.

Frames: Main memory is divided into small fixed-sized blocks of physical memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

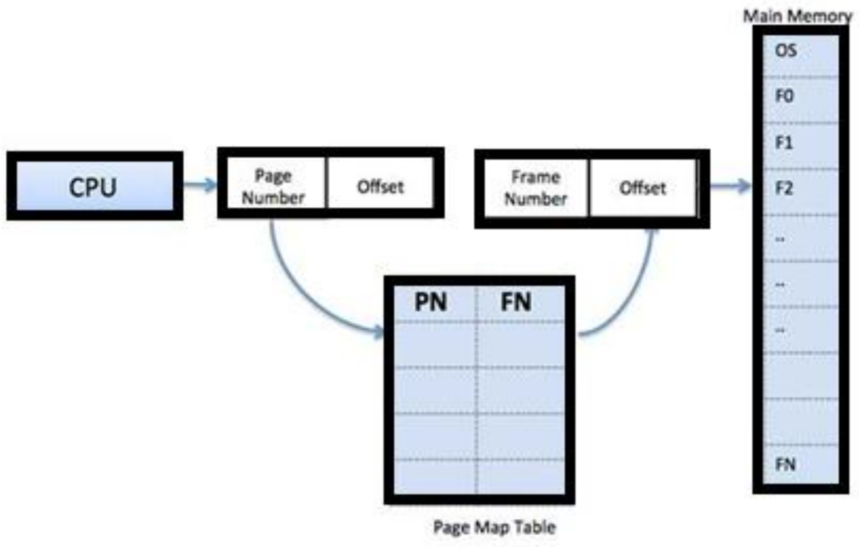
Implementation: The ides behind is - breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source. When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program. The backing store is divided into fixed-sized blocks that are the same size as the memory frames.

Page address is called logical address and represented by page number and the offset.

Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame number + page offset



Advantages:

- The logical address space is now totally separate from the physical address space, so a process can have a logical 64-bit address space even though the system has less than 2^{64} bytes of physical memory.
- Paging reduces external fragmentation,
- Paging is simple to implement
- Swapping is easy since the page size and frame size are equal.

Disadvantages:

- Page table requires extra memory space
- Has internal fragmentation.

* * *

OS UNIT V

5.1 VIRTUAL MEMORY MANAGEMENT

Virtual memory is a technique that allows the execution of processes that are not stored completely in memory. The programs can be larger than physical memory. Virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory from physical memory. There exist no limitations on memory-storage. Virtual memory allows processes to share files easily and to implement the concept of shared memory.

Virtual memory provides an efficient mechanism for process creation. Most processes never need all their pages at once. For example, Error handling code is not needed unless that specific error occurs. Similarly, Arrays are often over-sized and normally, only a small fraction of the arrays are actually used in practice. Under such circumstances, the concept of virtual memory can be applied in which virtual memory space is more than actual physical memory.

Virtual Memory can be defined as an extra space as a section of the hard disk, where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into it. Memory protection is achieved by translating the virtual address to a physical address. It is commonly implemented by the following methods:

- demand paging
- segmentation
- demand segmentation

Advantages:

- Programs can be larger than physical memory
- It allows us to extend the use of physical memory by using disk.
- Programs occupy very less space on actual physical memory.

The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory. (Figure 5.1) A process begins at a certain logical address and exists in contiguous memory.

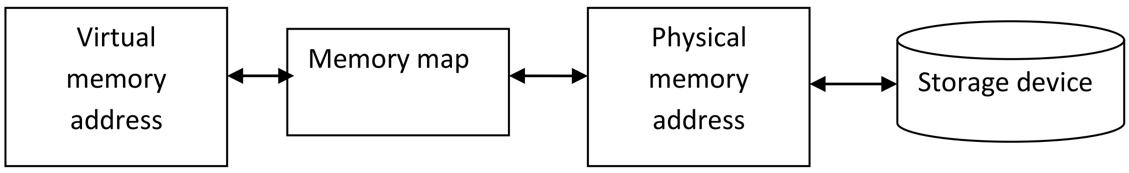


Fig 5.1 Virtual memory allocation

The memory management unit (MMU) maps the logical pages to physical page frames in memory. In addition to separating logical memory from physical memory, virtual memory allows files and memory to be shared by two or more processes through page sharing. System libraries can be shared by several processes through mapping of the shared object into a virtual address space. A library is mapped read-only into the space of each process that is linked with it. Processes can communicate through the use of shared memory. Virtual memory allows one process to create a region of memory that it can be shared with another process.

5.2 DEMAND PAGING

In demand paging method, the processes reside in secondary memory and pages are loaded into main memory only on demand, and not in advance. Pages that are never accessed are thus never loaded into physical memory.(Figure 5.2)

A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory. During a process execution, swapping into the memory is performed. When a process is to be swapped in, instead of swapping in a whole process, the pager brings only those pages into memory. Thus, it avoids reading into memory pages that will not be used thereby decreasing the swapping time and the amount of physical memory needed.

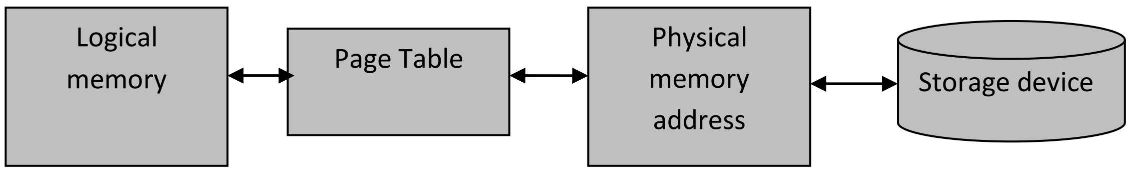


Fig 5.2 Demand Paging

The page table has ***valid*** – ***invalid*** bit indicating if a page is in memory or in disk respectively.

Page Fault: During program execution, if the program references a page which is not available in the main memory, a page fault occurs. Page fault is an invalid memory reference. Processor transfers control from the program to the operating system to demand the page back into the memory. The procedure for handling page fault is,

- Check an internal table for the process to determine whether the reference was a valid or an invalid memory access.
    - If the reference is invalid, terminate the process.
    - If it is valid but not in memory, page it in as described below.
- Find a free frame
    - Schedule a disk operation to read the desired page into the newly allocated frame.
- When the disk read is complete, modify the internal table and the page table to indicate that the page is now in memory.
- Restart the instruction that was interrupted by the trap.
- The process can now access the page.

During page fault, the interrupted process's state of registers, condition code, instruction counter are saved. Restarting the process in ***exactly*** the same place and state is made possible.

Pure demand paging: This scheme never bring a page into memory until it is required.

The hardware to support demand paging are,

- ***Page table*** – marks an entry invalid through a ***valid–invalid*** bit or a special value of protection bits.
- ***Secondary memory*** - holds the pages that are not present in main memory. It is known as the swap device, and the section of disk used for this purpose is known as **swap space**.

Efficiency: Demand paging can significantly affect the performance of a computer system.

If there are no page faults, then

the effective access time = the memory access time.

If, a page fault occurs, we must first read the relevant page from disk and then access the desired word. The effective access time can be calculated as,

effective access time = (1 − p) × ma + p × page fault time

where ***p – the probability of page fault and,***

ma – the memory access time

Advantages

- Efficient use of memory.
- Multiprogramming capability.

Disadvantage : Processor overhead for handling page interrupts are greater

5.3 PAGE REPLACEMENT ALGORITHMS

In demand paging, when a process requests for more pages and free memory is not available to bring them in, the pages that are not being used currently are identified and moved to the disk to get free frames. This technique is called Page Replacement.

Page replacement takes the following approach.

- If no frame is free, find one that is not currently being used and free it.
- Before freeing a frame do the following
    - writing its contents to swap space
    - changing the page table and other tables to indicate that the page is no longer in memory
- Use the freed frame to hold the page for which the process faulted.
- Modify the page-fault service routine to include page replacement:

Modify bit / dirty bit: Each page or frame has a modify bit associated with it in the hardware. The modify bit for a page is set by the hardware whenever any byte in the page is written into, indicating that the page has been modified.

Before selecting a page for replacement, examine its modify bit.

Bit is set - If the bit is set, it indicates that the page has been modified. If so, write the page to the disk.

Bit not set- If the modify bit is not set, the page has not been modified since it was read into memory and need not write the memory page to the disk. This scheme can significantly reduce the time required to service a page fault, since it reduces I/O time by one-half if the page has not been modified.

A frame-allocation algorithm and a page-replacement algorithm are needed to implement demand paging. Frame allocation decides, the number of frames to be allocated to each process. Page replacement decides, the frame to be selected for replacement.

Page Replacement Algorithms

Page replacement algorithms are the techniques using which an Operating System decides which memory pages are to be swapped out to get free frames for a new page to enter into memory. The fewer the page faults the better the algorithm.

Some the page replacement algorithms are,

- FIFO algorithm
- Least Recently Used (LRU) algorithm
- Page Buffering algorithm

• Most frequently Used(MFU) algorithm

First In First Out (FIFO) algorithm

Page replacement is based on First In first Out. The Oldest page in main memory is selected for replacement. (Figure 5.3) Using a queue, it replace pages from the tail and add new pages at the head. This method can be used for small systems.

Example :

Number of frames = 3

Reference string = **9 0 1 6 0 3 0 4 6 3 0 3 6 1 6 0 1 9 0 1**

9	**9**	**9**	**6**
	0	**0**	**0**
		1	**1**

6	**6**	**4**	**4**	**4**	**0**
3	**3**	**3**	**6**	**6**	**6**
1	**0**	**0**	**0**	**3**	**3**

0	**0**
1	**1**
3	**6**

9	**9**	**9**
1	**0**	**0**
6	**6**	**1**

Fig 5.3 FIFO Page replacement algorithm

Optimal Page algorithm

This algorithm follows the method of replacing the page that will not be used for the longest period of time. An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. (Figure 5.4)

Example:

Number of frames = 3

Reference string = **9 0 1 6 0 3 0 4 6 3 0 3 6 1 6 0 1 9 0 1**

9	**9**	**9**	**6**
	0	**0**	**0**
		1	**1**

6
0
3

6
4
3

6
0
3

6
0
1

9
0
1

Fig 5.4 FIFO Page replacement algorithm

Least Recently Used (LRU) algorithm

Page which has not been used for the longest time in main memory is the one which will be selected for replacement. It maintains a list and replaces the pages by looking back into time. (Figure 5.3)

Example:

Number of frames = 3

Reference string = **9 0 1 6 0 3 0 4 6 3 0 3 6 1 6 0 1 9 0 1**

9	**9**	**9**	**6**	**6**	**4**	**4**	**4**	**0**	**1**	**1**	**1**
	0	**0**	**0**	**0**	**0**	**0**	**3**	**3**	**3**	**0**	**0**
		1	**1**	**3**	**3**	**6**	**6**	**6**	**6**	**6**	**9**

Fig 5.5 LRU Page replacement algorithm

Page Buffering algorithm

This algorithm maintains a pool of free frames. During page fault, it writes the new page in the frame of free pool, marks the page table and restarts the process. Then it selects a page to be replaced, sends it out and places the frame holding replaced page in free pool.

Least frequently Used (LFU) algorithm

This algorithm maintains the record of the frequency of page usage. The page with the smallest count is the one which will be selected for replacement. The drawback of this method is, if a page is used heavily during the initial phase of a process, but then is never used again efficiency decreases.

Most frequently Used (MFU) algorithm

This algorithm also maintains the record of the frequency of page usage. This algorithm is based on the principle that the page with the smallest count was probably the latest entered page and is yet to be used. The page with the maximum count is selected for replacement.

5.4 FILE SYSTEM

A file is a named collection of related information. A file has sequence of bits, bytes, lines or records defined by the creator of the file. Various types are:

- ***Text file:*** Sequence of characters organized into lines.
- ***Source file:*** Sequence of procedures and functions.
- ***Object file:*** Sequence of bytes organized into blocks that a machine can understand
- ***Directory files:*** has list of file names and other information related to files.

The file system provides the mechanism for on-line storage and access to file contents, that includes data and programs. The file system resides permanently on secondary storage, which is designed to hold a volume of data permanently. I/O transfers between memory and disk are performed in units of blocks. Each block has one or more sectors.

The important characteristics of disk storage are,

- A disk can be rewritten in same place; reading, modifying and writing a block back into the same place is possible.
- A disk can access directly the block of information sequentially or randomly is possible
- Switching from one file to another can be carried out easily by moving the read–write heads.

It provide efficient and convenient access to the disk for file related operations like storing, locating and retrieving. To perform this, the file structure must be designed considering the following:

- defining a file and its attributes,
- the operations allowed on a file,
- the directory structure for organizing files.
- mapping the logical file system onto the physical secondary-storage devices.

The file system itself is generally composed of many different levels.

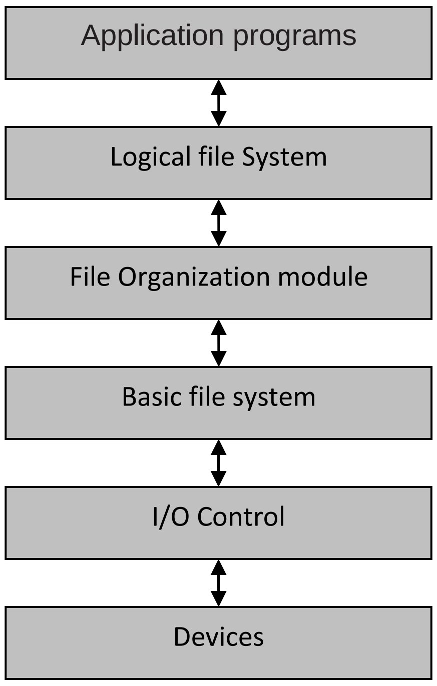


Fig 5.6 Layered structure

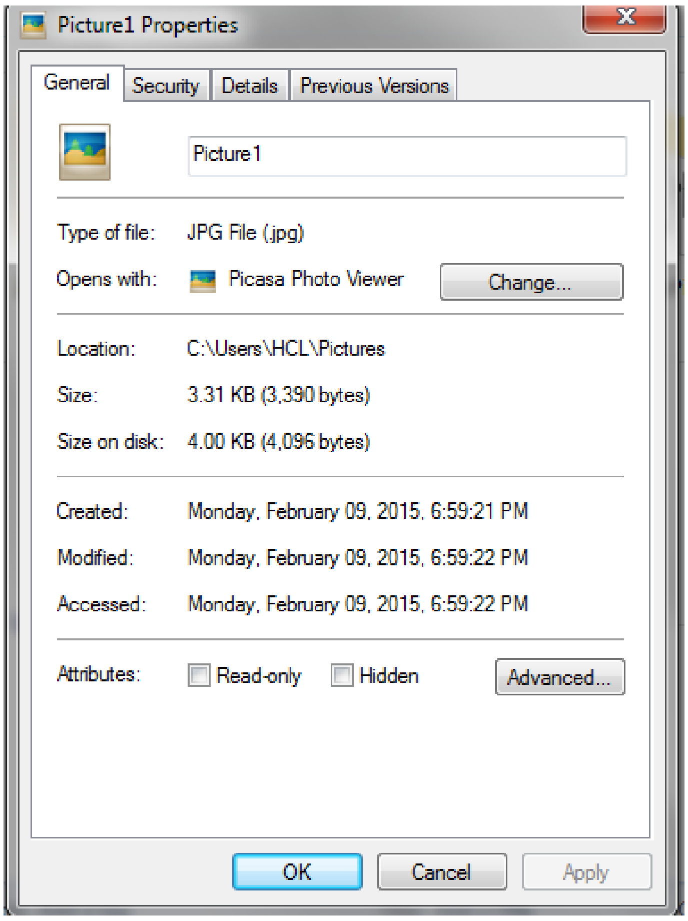
The layered design structure is shown in Figure 5.6 The features of lower levels are used to create new features for higher levels.

When operating system defines different file structures, it also contains the code to support these file structure. The structure of the file should be according to a required format that the operating system can understand. Unix, MS-DOS operating systems support minimum number of file structure.

File Attributes: Important attributes of a file are:

- *Name*: the file name.
- *Identifier:* a unique number in file system.
- *Type:* the type of file.
- *Location:* Pointer to file location on device.
- *Size:* Size of the file.
- *Protection:* controls the file with reading, writing, executing permission to user.

The various properties of a file is shown by Windows OS as depicted in Fig 5.7


Picture1 Properties

General | Security | Details | Previous Versions

Picture1

Type of file: JPG File (.jpg)

Opens with: Picasa Photo Viewer

Location: C:\Users\HCL\Pictures

Size: 3.31 KB (3,390 bytes)

Size on disk: 4.00 KB (4,096 bytes)

Created: Monday, February 09, 2015, 6:59:21 PM

Modified: Monday, February 09, 2015, 6:59:22 PM

Accessed: Monday, February 09, 2015, 6:59:22 PM

Attributes: ☐ Read-only ☐ Hidden

Fig 5.7 File information on Windows OS

Common File operations are:

- Creating a file

- Writing a file
- Reading a file
- Deleting a file

Information associated with an open file are,

• ***File pointer:*** Points to the current file position. This pointer is unique to each process operating on the file.

• ***File-open count***: Since, multiple processes would have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open count tracks the number of opens and closes for a file.

• ***Disk location of the file:*** The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

• ***Access rights:*** Each process opens a file in an access mode which is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

Some systems provide the following ***file locking*** mechanisms***:***

- ***Shared lock:*** reading alone.
- ***Exclusive lock*** : reading and writing.
- ***Advisory lock*** : informational only, and not enforced.
- ***Mandatory lock*** : is enforced.

5.4.1 File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. File pointers records the current position in the file, for the next read or write operation. Several ways to access files are:

- ***Sequential access:*** The information in the file are accessed in some sequential order normally, one record after the other. Example : Magnetic Tape

  Reads and writes are important operations on a file. In sequential access, a read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.

  In similar way, the write operation appends to the end of the file and advances to the end of the new file.

  - ***Direct/Random access:*** A file is made up of fixed-length logical records that allow programs to read and write records quickly in any order. The direct-access method is based on a disk model of a file. Disks allow random access to any file block. Each record has its own address on the file and is directly accessed for reading or writing. There is no restriction on reading / writing order. Direct-access files are widely used for immediate access to volume of information.

Block number: For the direct-access method, the file operations uses the block number to locate the exact position. Example: In ***read(n)***,operation, n is the block number. The block number provided by the user to the operating system is normally a ***relative block number*** which is an index relative to the beginning of the file.

- ***Indexed sequential access:*** An index is created for each file which contains pointers to various blocks. The Index is searched sequentially and its pointer is used to access the file directly. It constructs an index for the file. To find a record in the file, initially the index is searched and then the pointer is used to access the file directly and to find the desired record. For large files, the index file itself may become too large to be kept in memory. Under such situation, an index for the index file can be created. The primary index file contains pointers to secondary index files, which point to the actual data items.

5.4.2. File Space Allocation

Files are allocated disk spaces by the operating system by one the following methods:

Contiguous Allocation:

- Each file occupies a neighbouring address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.

Linked Allocation

- Each file has a list of links to disk blocks.
- Directory points to first block of a file.
- Effectively used in sequential access file.

Indexed Allocation

- Each file has its own index block which stores the addresses of disk space occupied by the file.
- An index block is created consisting of all pointers to files.
- Directory contains the addresses of index blocks of files.

An indexed access scheme can be easily built on top of a direct access system. Very large files may require a multi-tiered indexing scheme, i.e. indexes of indexes.

5.5 DIRECTORY AND DISK STRUCTURE

All Information about files is maintained by Directories. A directory may contain multiple files or directories inside them. Directories are also called as folders. When the root of one file system is in the existing tree of another file system it is called as ***Mounting.*** A directory maintains the following information:

- **Name** : The name of the directory.
- **Type** : Type of the directory.
- **Location** : Device and location on the device where the file header is located.
- **Size** : the size of each file in the directory..
- **Position** : Current next-read/next-write pointers.
- **Protection** : Access control on read/write/execute/delete.
- **Usage** : Time of creation, access, modification etc.

A physical disk can be divided into multiple partitions, each of which can have its own file system. (Figure 5.8) Multiple physical disks can be combined into one ***volume***, i.e. a larger virtual disk, with its own file system spanning the physical disks.

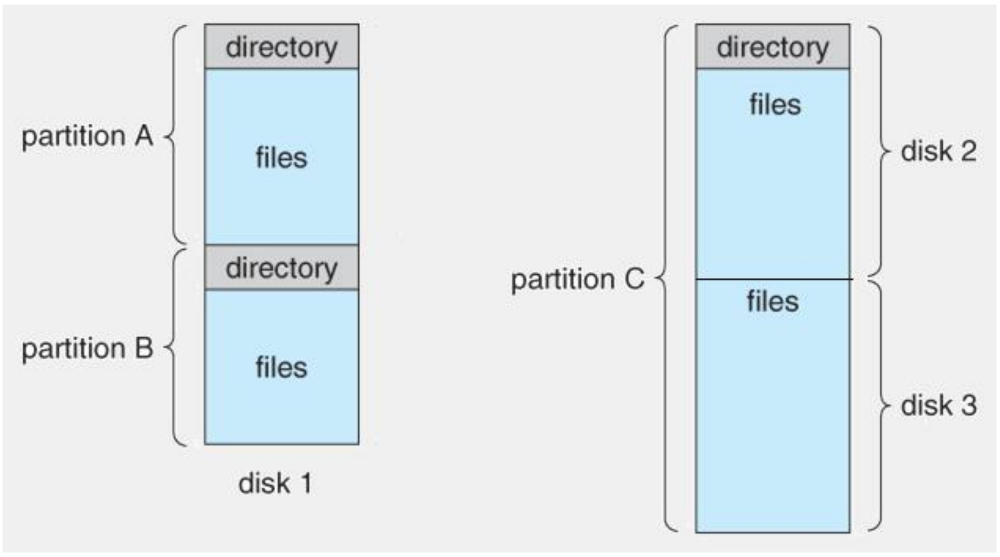


Fig. 5.8 - File-System Organization.

Disk Partitioning is useful for the following reasons:

- limits the size of individual file systems
- puts multiple file-system types on the same device
- leaves part of the device available for other uses

- A file system can be created on each of these parts of the disk.

Volume:

- Any entity containing a file system is commonly known as a volume.
- The volume may be a subset of a device, a whole device, or multiple devices linked together.
- Each volume can be thought of as a virtual disk.
- Volumes can store multiple operating systems, allowing a system to boot and run more than one operating system.
- Each volume consisting of a file system must contain information about the files which is kept in entries in a volume table of contents also called as device directory or directory. It records the information such as name, location, size, and type—for all files on that volume.

Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries. The directory can be organized in many ways to perform the following operations:
- Search for a file - Searching the directory structure to find the entry for a particular file or to find all files whose names match a particular pattern.
- Create a file - New files need to be created and added to the directory.
- Delete a file - The file that is no longer needed, must be removed from the directory.
- List a directory - Listing the files in a directory and the contents of the directory entry for each file in the list.
- Rename a file - Changing the name of the file.
- Traverse the file system- Accessing every directory and every file within a directory structure.

Single-Level Directory

- Simple to implement, but each file must have a unique name. All files are contained in the same directory, which is easy to support. When the number of files increases or

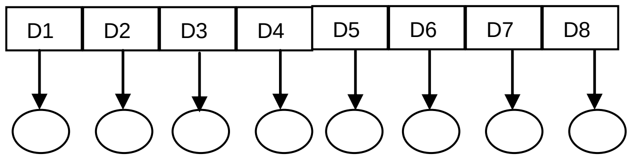


Fig 5.9 Single level directory and files

when the system has more than one user it becomes difficult. Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases. (Figure 5.9)

Two-Level Directory

- The structure of two level directory is shown in Figure 5.10.
- Two level directory has
    - Master file directory
    - User file directory.
- Each user gets their own directory space.

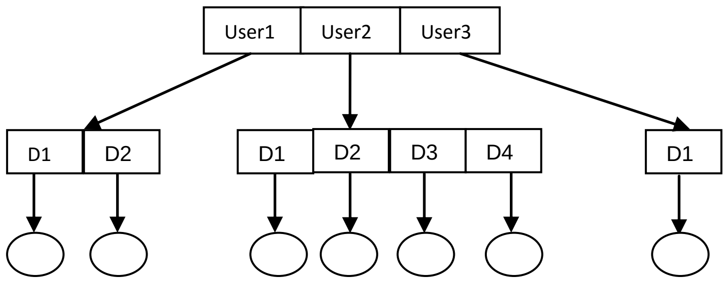


Fig 5.10 Two level directory

- File names need to be unique only within a given user's directory.
- A master file directory is used to keep track of each user’s directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system executable files.
- Accessing other directories by users need permission
- A ***search path*** is the list of directories in which to search for executable programs, and can be set uniquely for each user.

Tree structured directory

The directory structure can be extended to a tree of arbitrary height. This allows users to create their own subdirectories and to organize their files accordingly.

- A tree has a root directory, and every file in the system has a unique path name.
- A directory / subdirectory contains a set of files or subdirectories.
- All directories have the same internal format. A status bit in each directory entry determines if it a file (0) or as a subdirectory (1).

- Normally each process has a current directory. When reference is made to a file, the current directory is searched.
- The file that is not in the current directory, specify the path name or change the current directory to be the directory holding that file.
- Path names can be of two types: absolute and relative.
- An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A relative path name defines a path from the current directory.

5.6 FILE SYSTEM IMPLEMENTATION

The file system contains the following information:

- The technique of booting an operating system
- the total number of blocks
- the number and location of free blocks
- the directory structure, and individual files.

A boot control block (per volume): It is the first block of a volume. It contains information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty.

A volume control block (per volume) contains the following volume / partition details:

- the number of blocks in the partition
- the size of the blocks
- a free-block count and free-block pointers
- a free-FCB (File Control Block) count and FCB pointers.

File Control Block: A directory structure (per file system) is used to organize the files. A FCB has a unique identifier number to allow association with a directory entry. A typical FCB is shown in Figure 5.11

file permissions
file dates - create, access, write
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Fig. 5.11 File-Control Block.

FCB functions under various situations:

- To create a new file, a new FCB is allocated. The system then reads the appropriate directory into memory, updates it with the new file name and FCB, and writes it back to the disk.
- When a file is accessed during a program, the open( ) system call reads in the FCB information from disk, and stores it in the system-wide open file table. An entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned by the open( ) system call. UNIX refers to this index as a ***file descriptor***, and Windows refers to it as a ***file handle***.
- If another process already has a file open when a new request comes in for the same file, and it is sharable, then a counter in the system-wide table is incremented and the per-process table is adjusted to point to the existing entry in the system-wide table.
- When a file is closed, the per-process table entry is freed, and the counter in the system-wide table is decremented. If that counter reaches zero, then the system wide table is also freed. Any data currently stored in memory cache for this file is written out to disk if necessary

Virtual File Systems (VFS)

- It provides a common interface to different file system types.
- It provides for a unique identifier for files across the entire space, including all file systems of different types.
- The VFS in Linux is based upon the following four key object types:
    - ***inode*** - representing an individual file
    - ***file*** - representing an open file.
    - ***superblock*** - representing a filesystem.
    - ***dentry*** - representing a directory entry.

Directory Implementation

Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space. The two important types of directory structures are Linear List and Hash Table.

Linear List

- A linear list is the simplest and easiest directory structure to set up.
- Finding a file in a directory requires only a linear search.

- Deleting a file can be done by moving all entries, by marking an entry as deleted with a flag bit. Moving the last entry into the newly vacant position is another method followed during deletion.
- Complex insertions and deletions are performed for Sorting the list quickly.
- A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.

Hash Table

- A hash table can also be used to speed up searches.
- Hash tables are generally implemented ***in addition to*** a linear or other structure
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- It can greatly decrease the directory search time.
- Insertion and deletion are performed directly
- The Hash table has a fixed size and the hash function depends on that size.

5.7 ALLOCATION METHODS

- There are three major methods of storing files on disks:
    - Contiguous allocation
    - Linked allocation
    - Indexed allocation.

Contiguous Allocation

- ***Contiguous Allocation* -** all blocks of a file be kept together continuously.
- Reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.Hence, the performance for reading operation is very fast
- Storage allocation involves the allocation of contiguous blocks of memory like first fit, best fit, fragmentation problems, etc.
- Problems can arise when files grow, or if the exact size of a file is unknown at creation time:
    - ***Over-estimation*** of the file's final size increases external fragmentation and wastes disk space.

    - ***Under-estimation*** may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.
    - If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.
- ***Extents:*** allocating file space in large contiguous chunks is called as ***extents.*** When a file outgrows its original extent, then an additional one is allocated. An extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.

Linked Allocation

- Disk files can be stored as linked lists, where the storage space is connected by each link.
- Linked allocation involves no external fragmentation
- It does not require the file sizes to be known well in advance
- It allows files to grow dynamically at any time.
- Linked allocation is only efficient for sequential access files.
- Allocating ***clusters*** of blocks reduces the space wasted by pointers, but it leads to internal fragmentation.
- If a pointer is lost or damaged this methods becomes unreliable.
- Doubly linked list is more reliable, but has additional overhead and wasted space.

The ***File Allocation Table, FAT,*** used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. The benefit of this approach is that the FAT table can be cached in memory, greatly improving random access speeds.

Indexed Allocation

- ***Indexed Allocation*** combines all of the indexes for accessing each file into a common block for that file. Some disk space is wasted because an entire index block must be allocated for each file, regardless of how many data blocks the file contains.
- Several approaches for determining the block size are,
    - **Linked Scheme -** An index block is one disk block, which can be read and written in a single disk operation. The first index block contains some

header information, the first N block addresses, and if necessary a pointer to additional linked index blocks.

- **Multi-Level Index -** The first index block contains a set of pointers to secondary index blocks, which in turn contain pointers to the actual data blocks.
- **Combined Scheme -** This is the scheme used in UNIX inodes, in which the first twelve data block pointers are stored directly in the inode, and then singly, doubly, and triply indirect pointers provide access to more data blocks as needed. For both small files and huge files, the data blocks are readily accessible using a relatively small number of disk accesses.

5.8 MASS-STORAGE STRUCTURE

Storage devices

Magnetic storage devices include magnetic disk such as floppy disk, hard disk, removable hard disk and magnetic tape. Magnetic storage devices record data as magnetic fields. The secondary storages are also called as external or auxiliary memory or backing storage. These devices are used to hold mass information which can be desirably transferred at any time. Two types of secondary memories, which have larger capacity are, Magnetic bubble memory and Charge coupled memory.

The advantage of magnetic bubble memory is that the large amount of data can be stored in a less area. They are considerably faster than semiconductor memories. Charge coupled devices (CCD) are volatile memory. In CCD’s presence or absence of electric charge represents a bit.

There exist different types of secondary storage devices, each of them suitable for a specific purpose. They mainly differ in the following aspects: Technology used to store data, Portability of storage device and Access time.

Magnetic Disks

- One or more ***platters*** in the form of dsks covered with magnetic media. ***Hard disk*** platters are made of rigid metal. Each platter has two working ***surfaces.*** The magnetic medium of the disk (hard disk or floppy) is pre-written with empty *tracks*, which are further divided into *sector*. The collection of all tracks that are the same distance from the edge of the platter, i.e., all tracks immediately above one another is called a ***cylinder***. One of many concentric rings, that are encoded on the disk during formatting are called tracks. A segment of one of the concentric tracks, encoded on a disk during formatting are called sectors. *(Figure 5.12)*

In addition, an empty index is also placed on the disk. The data is stored in sectors and tracks and the PC locates specific data with the help of the index. The placing of tracks, dividing them into sectors, and placing of an empty index is called *formatting* of the disk. Without formatting, the disk will not accept any data.

The basic unit of data storage in magnetic disk is called cluster. A cluster includes two or more sectors. The smallest space that a file can occupy is a cluster. Depending on the size of the file, it can take, track of the clusters, that hold the contents of each file.

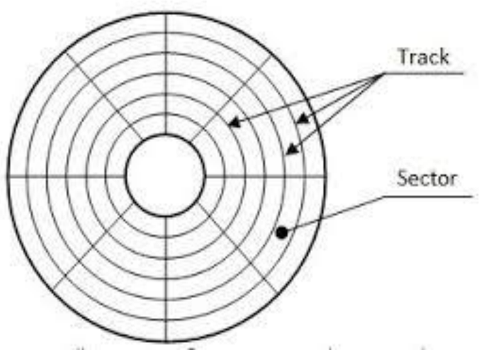


Figure 5.12 Track, Sector

- Each track is further divided into ***sectors,*** traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. The data on a hard drive is read by read-write ***heads.*** The standard configuration uses one head per surface, each on a separate ***arm***, and controlled by a common ***arm assembly*** which moves all heads simultaneously from one cylinder to another.
- The storage capacity of a traditional disk drive depends on the following:
  - the number of working surfaces
  - the number of tracks per surface
  - the number of sectors per track
  - the number of bytes per sector.

A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located. In operation the disk rotates at high speed, such as 7200rpm. The rate at which data can be transferred from the disk to the computer is composed of several steps:

- The **transfer rate** is the rate at which data flow between the drive and the computer.
- The **positioning time**, or **random-access time**, consists of two parts:
    - ***Seek time:*** the time necessary to move the disk arm to the desired cylinder,
    - ***rotational latency:*** the time necessary for the desired sector to rotate to the disk head.

Typical disks can transfer several megabytes of data per second, and they have seek times and rotational latencies of several milliseconds.

- If the disk heads accidentally contact the disk, then a ***head crash*** occurs, which may or may not permanently damage the disk or even destroy it completely.
- Disk drives are connected to the computer via a cable known as the ***I/O Bus.*** Some of the common Bus types are:
    - Enhanced Integrated Drive Electronics (EIDE)
    - Advanced Technology Attachment (ATA)
    - Serial ATA (SATA)
    - Universal Serial Bus (USB)
    - Fiber Channel (FC)
    - Small Computer Systems Interface (SCSI)
- The ***host controller*** is at the computer end of the I/O bus, and the ***disk controller*** is built into the disk itself. The CPU issues commands to the host controller via I/O ports. Data is transferred between the magnetic surface and onboard ***cache*** by the disk controller, and then the data is transferred from that cache to the host controller and the motherboard memory at electronic speeds.

Hard disks are designed to store very high volume of data. Hard disks can store gigabytes to terabyte of data and they form an integral part of the computer. Most operating systems are stored in hard disks and all contents (application software, documents, images, music files etc) in the computer are also stored in the hard disk.

The data is stored on a metal platter in the form of magnetic spots. The metal platter is sealed inside a disk drive that protects the platter and as well enables reading and writing to the disk. Hard disk may contain several platters forming hard disk packs that increase the storage capacity. These provide fast access for both reading and writing. Four common hard disk storage technologies are:

- ATA (Advanced Technology Attachment)
- FireWire
- SCSI(Small Computer System Interface)
- RAID (Redundant Array of Inexpensive Disks) and Fiber Channel

iVDR hard disk

iVDR is an acronym of **i**nformation **V**ersatile **D**evice for **R**emovable usage, which means a portable information device designed for various applications. Cartridge dimensions, mechanical and electrical interface specifications and recording method have been established as "iVDR standards." iVDR hard disks are compatible with various kinds of equipment.
Some standards of iVDR types are: 1 TB, 500 GB and 320 GB. 1 TB is equivalent to about 214 DVD discs (single-sided single-layer type = 4.7 GB per disc). The capacity of hard disks is increasing by 40 percent approximately every year.

Magnetic Tape

Magnetic tapes are very similar to the tapes that are used in radio cassette player. It provides a very effective means for backing up of large amount of data. Tape drives are well suited for backing up a systems entire hard disk. Capacities of tapes can be as high as 100 GB and more, tape offers an inexpensive way to store a lot of data on a single cassette.

Special type of tape drive uses digital audiotape (DAT) to achieve high storage capacities. DAT drives typically have multiple read and write heads built into a small wheel or cylinder that spins near the tape at high speed. The main limitation is that data stored is in a sequential mode, and hence needs long access time.

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives, but today are used primarily for backups. Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives. Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

Disk Structure

- The traditional ***head-sector-cylinder***, HSC numbers are mapped to linear block addresses by numbering the first sector on the first head on the outermost track as sector 0. Numbering proceeds with the rest of the sectors on that same track, and then the rest of the tracks on the same cylinder before proceeding through the rest of the cylinders to the center of the disk. In modern practice these linear block addresses are used in place of the HSC numbers for a variety of reasons:
- The linear length of tracks near the outer edge of the disk is much longer than for those tracks located near the center, and therefore it is possible to squeeze many more sectors onto outer tracks than onto inner ones.
- All disks have some bad sectors. A few spare sectors can be used in place of the bad ones. The mapping of spare sectors to bad sectors in managed internally to the disk controller.
- Modern Hard drives have thousands of cylinders, and hundreds of sectors per track on their outermost tracks. These numbers exceed the range of HSC numbers on traditional ones.
- The limit on how closely packed individual bits can be placed on a physical media, is growing increasingly due to technological advances.

Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:

- ***Constant Linear Velocity (CLV):*** the density of bits is uniform from cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders. This is the approach used by modern CDs and DVDs.
- ***Constant Angular Velocity (CAV):*** the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. Hard disks have a constant number of sectors per track on all cylinders.

Disk Attachment

Disk drives can be attached either directly to a particular host (a local disk) or to a network. The two attachment methods are,

- Host attached storage
- Network attached storage

Host-Attached Storage: The most common interfaces are IDE or ATA, each of which allow up to two drives per host controller. High end workstations or other systems that needs larger number of disks use SCSI disks:

- The SCSI standard supports up to 16 ***targets*** on each SCSI bus, one of which is generally the host adapter and the other 15 of which can be disk or tape drives.
- A SCSI supports up to 8 ***units*** within each target. These would generally be used for accessing individual disks within a RAID array.
- The SCSI standard also supports multiple host adapters in a single computer, i.e. multiple SCSI busses.
- SCSI cables may be either 50 or 68 conductors. SCSI devices may be external as well as internal.

- ***Fibre channel*** (FC) is a high-speed serial architecture that can operate over optical fiber or four-conductor copper wires, and has two variants:
- A large switched fabric having a 24-bit address space that allows for multiple devices and multiple hosts to interconnect, forming the basis for the ***storage-area networks, SANs,***
- The ***arbitrated loop, FC-AL,*** that can address up to 126 devices like drives and controllers.

Network-Attached Storage

- Network attached storage connects storage devices to computers using a ***remote procedure call (RPC)*** interface, with NFS file system mounts. (Figure 5.13)
- Allows several computers in a group for shared storage.
- Can be implemented using ISCSI cabling, allowing long-distance remote access to shared files.
- Allows computers to easily share data storage, but it is less efficient than standard host-attached storage.

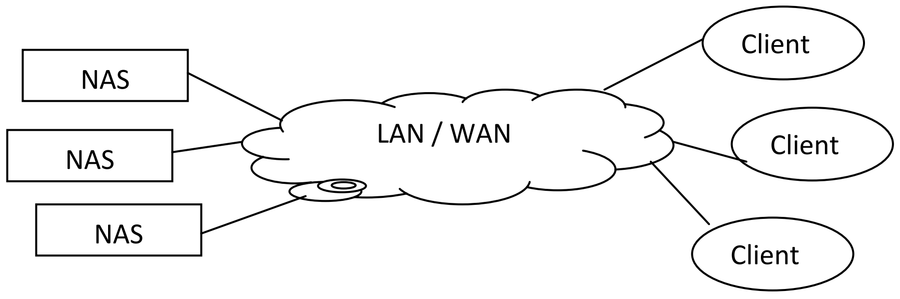


Fig 5.13 Network Attached Storage

DISK SCHEDULING

The disk transfer speeds are limited primarily by ***seek times*** and ***rotational latency.*** When multiple requests are to be processed there is also some inherent delay in waiting for other requests to be processed.

- ***Bandwidth*** is measured by the amount of data transferred to the total amount of time from the first request being made to the last transfer being completed, for a series of disk requests.
- Disk requests include the following:
    - disk address
    - memory address
    - number of sectors to transfer
    - reading or writing request

If the desired disk drive and controller are available, the request is serviced immediately. If not, any new requests for service will be placed in the queue of pending requests for that drive.

In a multiprogramming system with many processes, the disk queue may have many pending requests. The operating system chooses pending request for service next using Disk Scheduling algorithms. Some of the algorithms are,

- FCFS Scheduling
- SSTF Scheduling - shortest-seek-time-first (SSTF) algorithm.
- SCAN Scheduling
- C-SCAN Scheduling
- LOOK Scheduling

FCFS Scheduling

First-Come First-Serve is a simple scheduling algorithm. It does not provide quick service and is not very efficient. Consider in the following sequence the wild swing from cylinder 132 to 16 and then back to 134. The head movement is very high thereby decreasing the performance. Example:

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

Movement 53 → 98→183 → 37 → 122 → 14 → 124 → 65 → 67

SSTF Scheduling

- ***Shortest Seek Time First*** scheduling is more efficient. It tries to service all the requests close to the current head position before moving the head far away to service other requests.
- SSTF selects the request with the least seek time from the current head position.
- SSTF reduces the total head movement.
- May lead to starvation if a constant stream of requests arrives for the same general area of the disk. Example:

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

Movement 53 → 65 →67 → 37 → 14 → 98 → 122 → 124 → 183

- This scheduling method results in a total head movement of only 236 cylinders

SCAN Scheduling

- The ***SCAN*** algorithm, also called as the ***elevator*** algorithm moves back and forth from one end of the disk to the other, similar to an elevator processing requests in a multistorey building.
- If a request arrives just ahead of the moving head then it will be processed right away, but if it arrives just after the head has passed, then it will have to wait for the head to pass going the other way on the return trip. This leads to a wide variation in access times.

Example:

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

Movement 53 → 37 →14→(0) → 65 → 67 → 98 → 122 → 124 → 183

C-SCAN Scheduling

The ***Circular-SCAN*** algorithm improves upon SCAN by treating all requests in a circular queue fashion - Once the head reaches the end of the disk, it returns to the other end without processing any requests, and then starts again from the beginning of the disk:

Example:

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

Movement 53 → 65 → 67 → 98 → 122 → 124 → 183 → (199) → (0) 14 → 37

LOOK Scheduling

LOOK scheduling improves upon SCAN by looking ahead at the queue of pending requests, and not moving the heads any farther towards the end of the disk than is necessary.

Example:

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

Movement 53 → 65 → 67 → 98 → 122 → 124 → 183 → 14 → 37

The Selection of a Disk-Scheduling Algorithm can be based on the load. For slightly larger loads, SSTF offers better performance than FCFS, but may lead to starvation when loads become heavy enough. For busier systems, SCAN and LOOK algorithms eliminate starvation problems.

* * *

Reference:

1. Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Eighth Edition ", Chapter 11