INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)
Dundigal, Hyderabad -500 043

COMPUTER SCIENCE&ENG (DATA SCIENCE)

LECTURE NOTES

Course Title	OPERATING SYSTEMS				
Course Code	ACSC12				
Program	UG20				
Semester	III				
Course Type	Core				
Regulation	**IARE - UG20**				
Course Structure	**Theory**			**Practical**	
	Lectures	**Tutorials**	**Credits**	**Laboratory**	**Credits**
	2	1	3	-	-
Course Coordinator	S.Swarna Keerthi, Assistant Professor				

COURSE OBJECTIVES:

The students will try to learn:	
I	The principles of operating system, its services and functionalities with the evolution of operating systems.
II	The concepts of processes, inter-process communication, synchronization and scheduling used in process management.
III	The concepts related to memory management, paging, and segmentation including protection and security mechanisms used in computer systems.
IV	The deeper insights into the reasons for deadlock occurrences, techniques used for deadlock detection, prevention and recovery.

COURSE OUTCOMES:

After successful completion of the course, Students will be able to:

CO No.	Course Outcomes	Knowledge Level (Bloom's Taxonomy)
CO 1	**Describe the importance of computer system resources for designing operating systems security policies.**	Remember
CO 2	**Demonstrate process control blocks and threads used in scheduling the process.**	Understand
CO 3	**Classify the importance of system calls to different Applications Programming Interface in developing operating systems.**	Understand
CO 4	**Construct the critical section problem used for process synchronization.**	Apply
CO 5	**Distinguish logical and physical address space applied in process management**	Analyze
CO 6	**Construct various page replacement algorithms applied for allocation of frames.**	Apply
CO 7	**Describe the use of storage management policies with respect to different storage management technologies.**	Remember
CO 8	**Classify the different access methods used for file management systems.**	Analyze
CO 9	**Demonstrate the working of operating systems as a resource manager and file system manager used for implementing different parts of operating systems.**	Understand
CO 10	**Describe the concept of free space management to improve efficiency and performance of operating systems.**	Understand
CO 11	**Make use of various methods of handling deadlocks used for system models.**	Apply
CO 12	**Make use of access rights to implement language based protection.**	Apply

MODULE-I: INTRODUCTION

Operating systems objectives and functions: Computer system architecture, operating systems structure, operating systems operations; Evolution of operating systems: Simple batch, multi programmed, time shared, personal computer, parallel distributed systems, real time systems, special purpose systems, operating system services, user operating systems interface; Systems calls: Types of systems calls, system programs, protection and security, operating system design and implementation, operating systems structure, virtual machines.

Process concepts:

At the end of the unit students are able to:		
Course Outcomes		**Knowledge Level (Bloom's Taxonomy)**
CO1	Describe the importance of computer system resources and the role of operating systems in their management policies and algorithms	Remember
CO2	Classify the importance of system calls to various Applications Programming Interface in developing operating systems.	Understand
CO3	Demonstrate about process control blocks and threads used in scheduling the process.	Understand
CO6	Construct various page replacement algorithms applied for allocation of frames.	Apply

Operating systems objectives and functions:

Definition:

An **Operating System** acts as a communication bridge (interface) between the user andcomputer hardware.

Objectives of Operating Systems:

The purpose of an operating system is to provide a platform on which a user can execute programs in a convenient and efficient manner. An operating system is a piece of software that manages the allocation of computer hardware. The coordination of the hardware must be appropriate to ensure the correct working of the computer system and to prevent user programs from interfering with the proper working of the system.

Functions of an operating System:

Security : The operating system uses password protection to protect user data and similar other techniques. It also prevents unauthorized access to programs and user data.

Control over system performance : Monitors overall system health to help improve performance. records the response time between service requests and system response to have a complete view of the system health.

Job accounting : Operating system Keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of user.

Error detecting aids : Operating system constantly monitors the system to detect errors and avoid the malfunctioning of computer system.

Coordination between other software and users: Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.

Memory Management : The operating system manages the Primary Memory or Main Memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is a fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An Operating System performs the following activities for memory management: It keeps tracks of primary memory, i.e., which bytes of memory are used by which user program. The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used. In multi programming, the OS decides the order in which process are granted access to memory, and for how long. It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation.

Processor Management : In a multi programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called process scheduling. An Operating System performs the following activities for processor management. Keeps tracks of the status of processes. The program which perform this task is known as traffic controller. Allocates the CPU that is processor to a process. De-allocates processor when a process is no more required.

Device Management : An OS manages device communication via their respective drivers. It keeps tracks of all devices connected to system. It designates a program responsible for every device known as the Input/Output controller. It decides which process gets access to a certain device and for how long.

File Management : A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files.

1.1 Computer System Architecture:

The computer system structure consists of interrupt mechanism for I/O devices, memory unit to run programs, memory protection, and disk storage to save program and data files.A modern general purpose computer has CPU and device controllers for devices such as disk, audio devices, and video display devices connected to a common system bus. Each controller is for a specific

device only and it has a local buffer. The CPU and device controllers can run concurrently and compete for memory. A memory controller synchronizes access to shared memory.

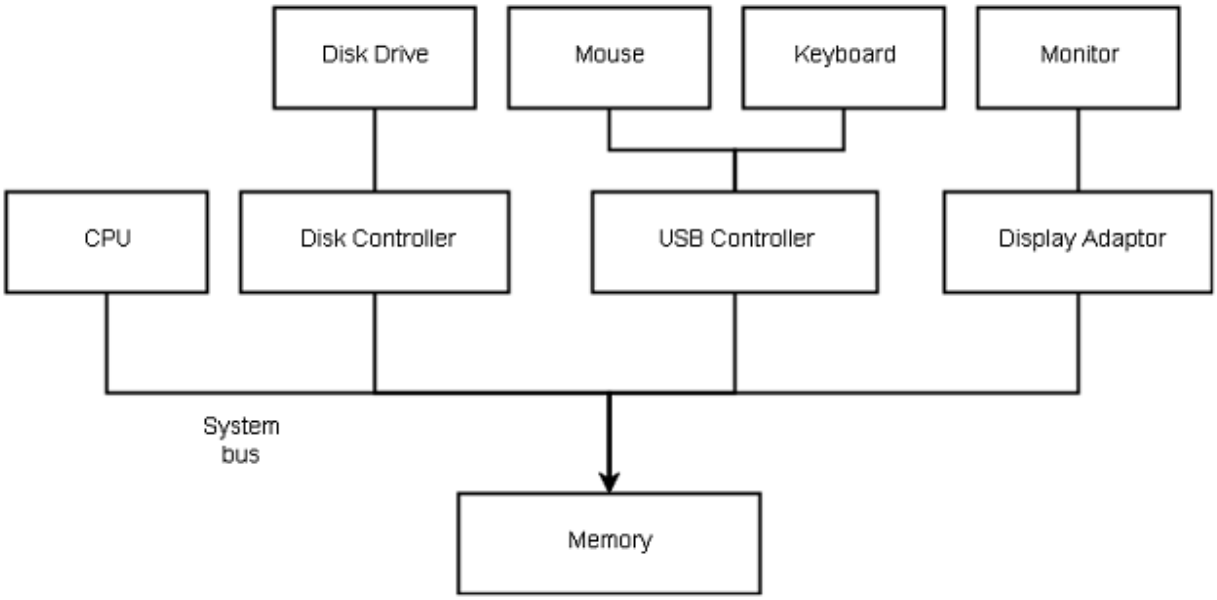


Figure 1.1 : Computer System Architecture

When a computer starts it initiates a simple program called the bootstrap program stored in a Read-Only Memory (ROM) or erasable programmable read-only memory (EPROM). The bootstrap is a simple program that initiates the system and loads the OS in memory. The OS waits for an event called Interrupt.

There are two types of interrupts : Hardware and Software interrupt. The hardware generates interrupt by signalling the CPU directly. A software interrupt is performed by a system call also known as monitor call. The CPU stops current operation and transfer the control to a fixed memory location where interrupt service routine (ISR) for the interrupt is located. Once interrupt service routine finished execution, the CPU resume the control of the interrupted operation.

1.2 Operating systems structure

An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions .

Simple Structure:

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.

An image to illustrate the structure of MS-DOS is as follows :

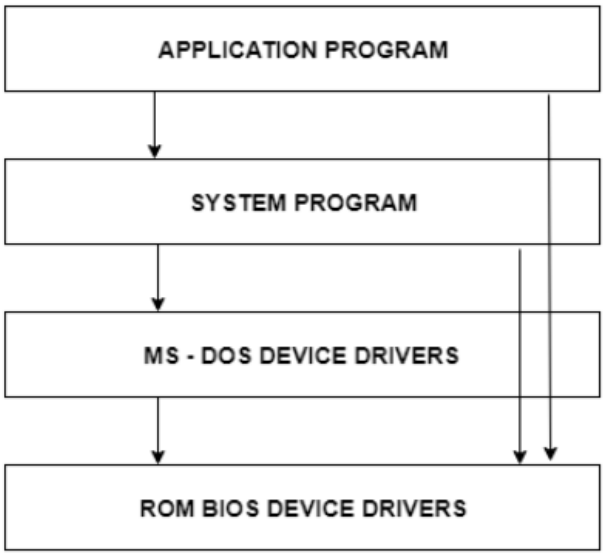


Figure 1.2a: Simple Structure

It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Layered Structure:

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows :

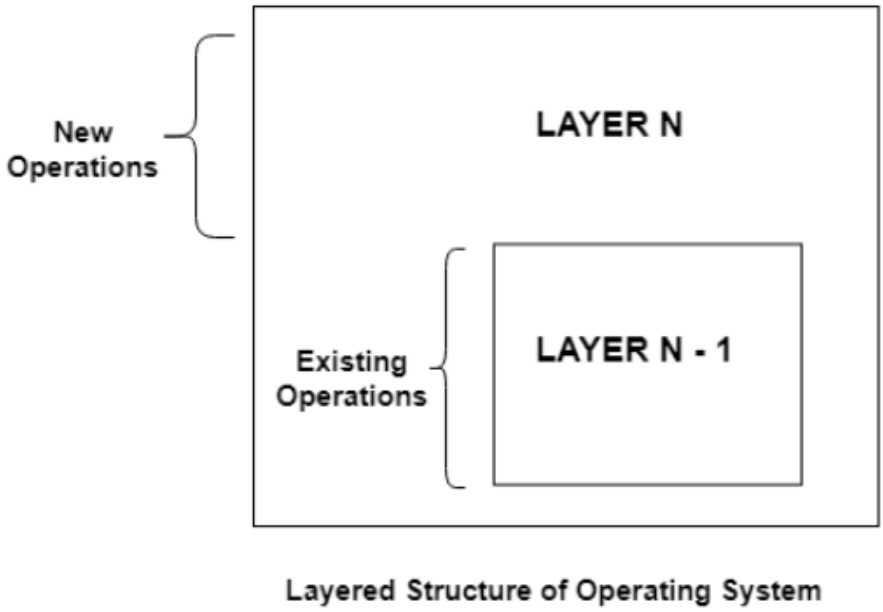


Figure 1.2b: Layered Structure

As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers. One problem with the layered structure is that each layer needs to be carefully defined.

1.3 Operating systems operations:

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.
The major operations of the operating system are process management, memory management, device management and file management.

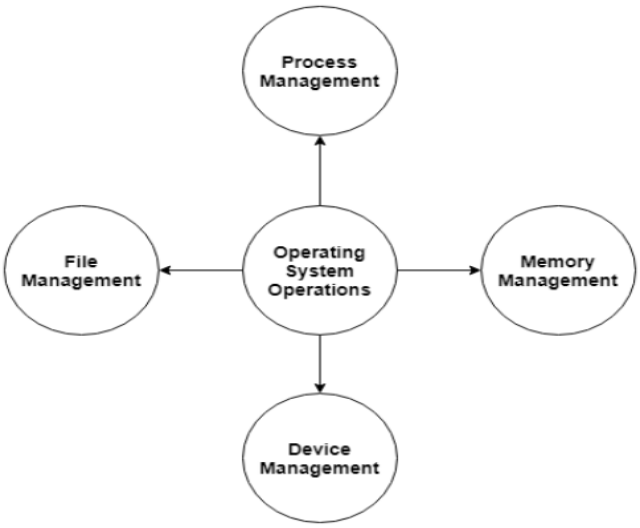


Figure 1.3: Operating System Operations

Process Management:

The operating system is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as process scheduling. The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round robin scheduling etc.

There are many scheduling queues that are used to handle processes in process management. When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the device queue.

Memory Management:

Memory management plays an important part in operating system. It deals with memory and the moving of processes from disk to primary memory for execution and back again.

The activities performed by the operating system for memory management are

- The operating system assigns memory to the processes as required. This can be done using best fit, first fit and worst fit algorithms.
- All the memory is tracked by the operating system i.e. it nodes what memory parts are in use by the processes and which are empty.
- The operating system deallocated memory from processes as required. This may happen when a process has been terminated or if it no longer needs the memory.

Device Management:

There are many I/O devices handled by the operating system such as mouse, keyboard, disk drive etc. There are different device drivers that can be connected to the operating system to

handle a specific device. The device controller is an interface between the device and the device driver. The user applications can access all the I/O devices using the device drivers, which are device specific codes.

File Management:

Files are used to provide a uniform view of data storage by the operating system. All the files are mapped onto physical devices that are usually non volatile so data is safe in the case of system failure.

The files can be accessed by the system in two ways i.e. sequential access and direct access :

- **Sequential Access**

  The information in a file is processed in order using sequential access. The files records are accessed on after another. Most of the file systems such as editors, compilers etc. use sequential access.

- **Direct Access**

  In direct access or relative access, the files can be accessed in random for read and write operations. The direct access model is based on the disk model of a file, since it allows random accesses.

1.4 Evolution of operating systems:

Operating systems have evolved from slow and expensive systems to present-day technology where computing power has reached exponential speeds and relatively inexpensive costs. In the beginning, computers were manually loaded with program code to control computer functions and process code related to business logic. This type of computing introduced problems with program scheduling and setup time. As more users demanded increased computer time and resources, computer scientists determined they needed a system to improve convenience, efficiency, and growth.

1.5 Simple Batch Systems

In this type of system, there is no direct interaction between user and the computer. The user has to submit a job (written on cards or tape) to a computer operator. Then the computer operator places a batch of several jobs on an input device. Jobs are batched together by type of languages and requirements. Then a special program, the monitor, manages the execution of each program in the batch. The monitor is always in the main memory and available for execution.

1.6 Multiprogramming Batch Systems

In this the operating system picks up and begins to execute one of the jobs from memory. Once this job needs an I/O operation, the operating system switches to another job (CPU and OS always busy). Jobs in the memory are always less than the number of jobs on disk(Job Pool). If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**. In Non-multiprogrammed system, there are moments when the CPU

sits idle and does not do any work. In a Multiprogramming system, CPU will never be idle and keeps on processing.

1.7 Time Sharing Systems

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems. In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.

1.8 Personal computer

Personal computer operating system provides a good interface to a single user. Personal computer operating systems are widely used for word processing, spreadsheets and Internet access. Personal computer operating systems are made only for personal use. We can say that your laptops, computer systems, tablets etc. are your personal computers and the operating system such as windows 7, windows 10, android, etc. are your personal computer operating system.

1.9 Parallel Distributed Systems

While both distributed computing and parallel systems are widely available these days, the main difference between these two is that a parallel computing system consists of multiple processors that communicate with each other using a shared memory, whereas a distributed computing system contains multiple processors connected by a communication network.

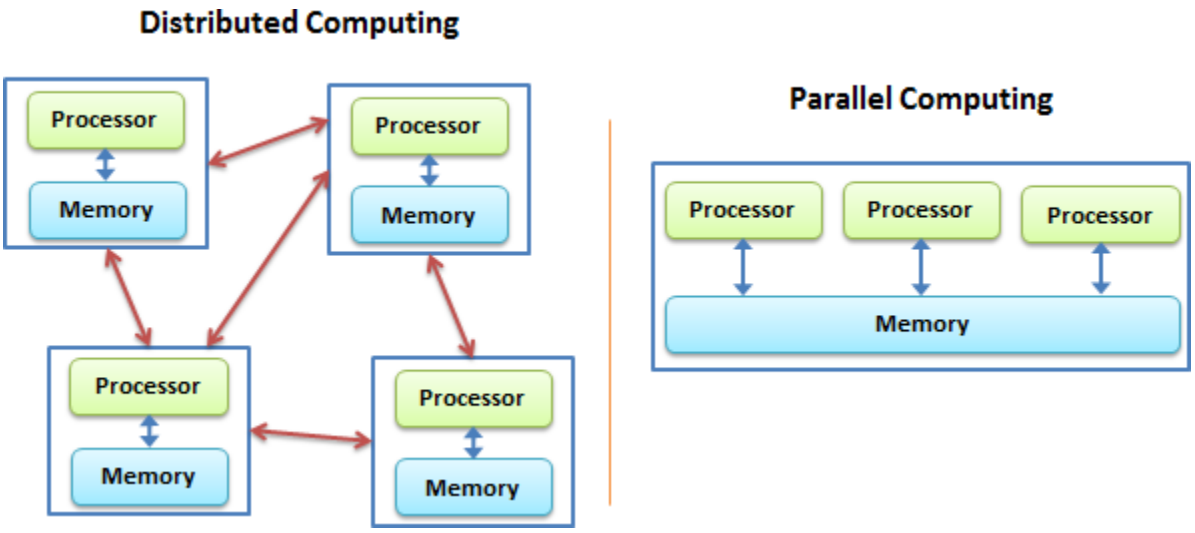


Figure 1.9: Parallel and distributed computing

In parallel computing systems, as the number of processors increases, with enough parallelism available in applications, such systems easily beat sequential systems in performance through the shared memory. In such systems, the processors can also contain their own locally allocated

memory, which is not available to any other processors. In distributed computing systems, multiple system processors can communicate with each other using messages that are sent over the network. Such systems are increasingly available these days because of the availability at low price of computer processors and the high-bandwidth links to connect them.

1.10 Real time systems

Real time system means that the system is subjected to real time, i.e., response should be guaranteed within a specified timing constraint or system should meet the specified deadline. For example: flight control system, real time monitors etc.

Types of real time systems based on timing constraints:

Hard real time system : This type of system can never miss its deadline. Missing the deadline may have disastrous consequences. The usefulness of results produced by a hard real time system decreases abruptly and may become negative if tardiness increases. Tardiness means how late a real time system completes its task with respect to its deadline. Example: Flight controller system.

Soft real time system : This type of system can miss its deadline occasionally with some acceptably low probability. Missing the deadline has no disastrous consequences. The usefulness of results produced by a soft real time system decreases gradually with increase in tardiness. Example: Telephone switches.

1.11 Special purpose systems

There are various classes of computer systems based upon their computational speed, usage and hardware. The following are some special purpose systems according to specific applications. They use:

1. Real-time embedded systems Multimedia systems
2. Handheld and portable systems
3. Handheld and portable systems

These are explained as follows below.

Real-time embedded systems Multimedia systems:

Embedded systems are small computers having a limited set of hardware like a small processor capable of processing a limited set of instructions (often called as an Application Specific Integrated Circuit (ASIC) a small memory and I/O device. These systems usually do specific tasks. Examples are microwave ovens, robots in a manufacturing unit, latest automotive engines, etc. A variety of embedded systems exist of which some are computers with standard operating systems, some have dedicated programs embedded in their limited memories and often some don't even have any software, hardware (ASIC) to do processing. Nearly all embedded systems use real-time operating systems because they are used as control devices and have rigid time requirements. Sensors are used to input data such as temperature, air pressure, etc, from the environment to the embedded system where that data is analyzed and several other controls are adjusted by the

embedded system itself to control the situation of the system.

Handheld and portable systems:

Multimedia refers to data of multiple types that includes audio and video including conventional data like text-files, word-processing documents, spreadsheets, etc. It requires that audio and video data must be processed based upon certain time restrictions. This is called streaming. It is usually 30 frames per second for a video file. Applications such as video conferencing, movies and clips downloaded over the internet, mp3, DVD, VCD playing and recording are examples of various multimedia applications. A multimedia application is usually a combination of both audio and video. The multimedia is not limited to desktop operating systems or computers but it is also becoming popular in handheld.

Handheld and portable systems:

Hand-held systems refer to small portable devices that can be carried along and are capable of performing normal operations. They are usually battery powered. Examples include Personal Digital Assistants (PDAs), mobile phones, palm-top computers, pocket-PCs etc. As they are handheld devices, their weights and sizes have certain limitations as a result they are equipped with small memories, slow processors and small display screens, etc. The physical memory capacity is very less (512 KB to 128 MB) hence the operating systems of these devices must manage the memory efficiently. As the processors are slower due to battery problems, the operating system should not burden.

1.12 Operating System Services

An operating system provides services to programs and to the users of those programs. It provides an environment for the execution of programs. The services provided by one operating system is different than other operating system.
Operating system makes the programming task easier. The common services provided by the operating system is listed below.

- Program execution
- I/O operation
- File system manipulation
- Communications
- Error detection.

Program execution: Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.

I/O operation: I/O means any file or any specific I/O device. Program may require any I/O device while running. So operating system must provide the required I/O.

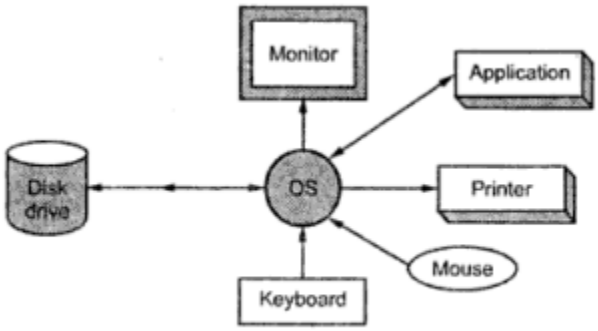
File system manipulation: Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.

Communication: Data transfer between two processes is required for some lime. The both processes are on the one computer or on different computer but connected through computer network Communication may be implemented by two methods: shared memory and message passing.

Error detection: Error may occur in CPU, in I/O device or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing.

Operating system with multiple users provide following services.

- Resource allocation
- Accounting
- Protection



View of OS with components

An operating system is a lower level of software that user programs run on. OS is built directly on the hardware interface and provides an interface between the hardware and the user program. It shares characteristics with both software and hardware. We can view an operating system as a resource allocator. OS keeps track of the status of each resource and decides who gets a resource, for how long, and when OS makes sure that different programs and users running at the same time but do not interfere with each other. It is also responsible for security, ensuring that unauthorized users do not access the system.

The primary objective of operating systems is to increase productivity of a processing resource, such as computer hardware or users. The operating system is the rust program run on a computer when the computer boots up. The services of the OS are invoked with a system call instruction that is used just like any other hardware instruction.

Name of the operating systems are: DOS, Windows 95, Windows NT/2000, UNIX, Linux etc.

1.13 Operating System Interface

User interface:
The OS provides a User Interface (UI), an environment for the user to interact with the machine. The UI is either graphical or text-based.

Graphical User Interface (GUI):
The OS on most computers and smartphones provides an environment with tiles, icons and/or menus. This type of interface is called the graphical user interface (GUI) because the user interacts with images through a mouse, keyboard or touchscreen.

Command Line Interface (CLI):
An OS also provides a method of interaction that is non-graphical, called the command line interface (CLI). This is a text-only service with feedback from the OS appearing in text. Using a CLI requires knowledge of the commands available on a particular machine.

Advantages of using the command line include:

- A **faster** way to get tasks done.
- It is **more flexible** than a GUI.
- It uses **less** memory.

1.14 System Calls

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.
System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

1.15 Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

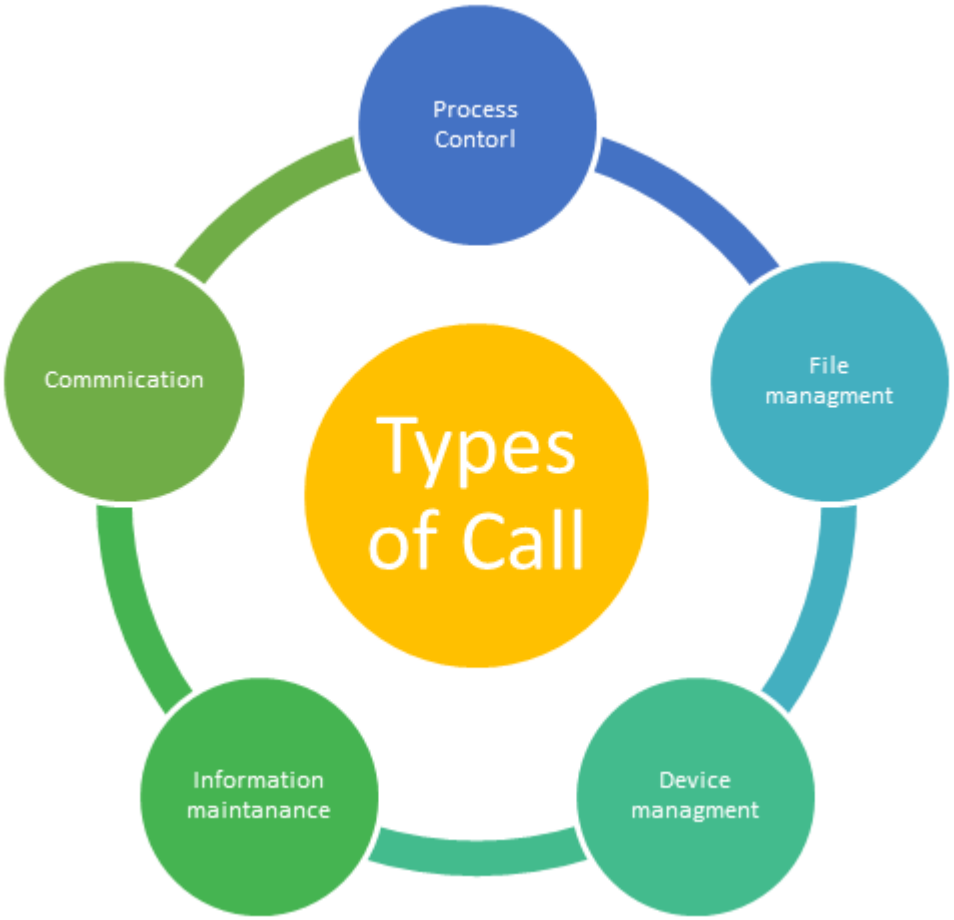


Figure 1.15: System Calls

Process Control:

This system calls perform the task of process creation, process termination, etc.
Functions:

- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signed Event
- Allocate and free memory

File Management:

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Device Management:

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
Functions:

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance:

It handles information and its transfer between the OS and the user program.

Functions:

- Get or set time and date
- Get process and device attributes

Communication:

These types of system calls are specially used for interprocess communications.

Functions:

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

1.16 System Programs

System Programming can be defined as act of building Systems Software using System Programming Languages. According to Computer Hierarchy, one which comes at last is Hardware. Then it is Operating System, System Programs, and finally Application Programs. Program Development and Execution can be done conveniently in System Programs. Some of System Programs are simply user interfaces, others are complex. It traditionally lies between user interface and system calls.

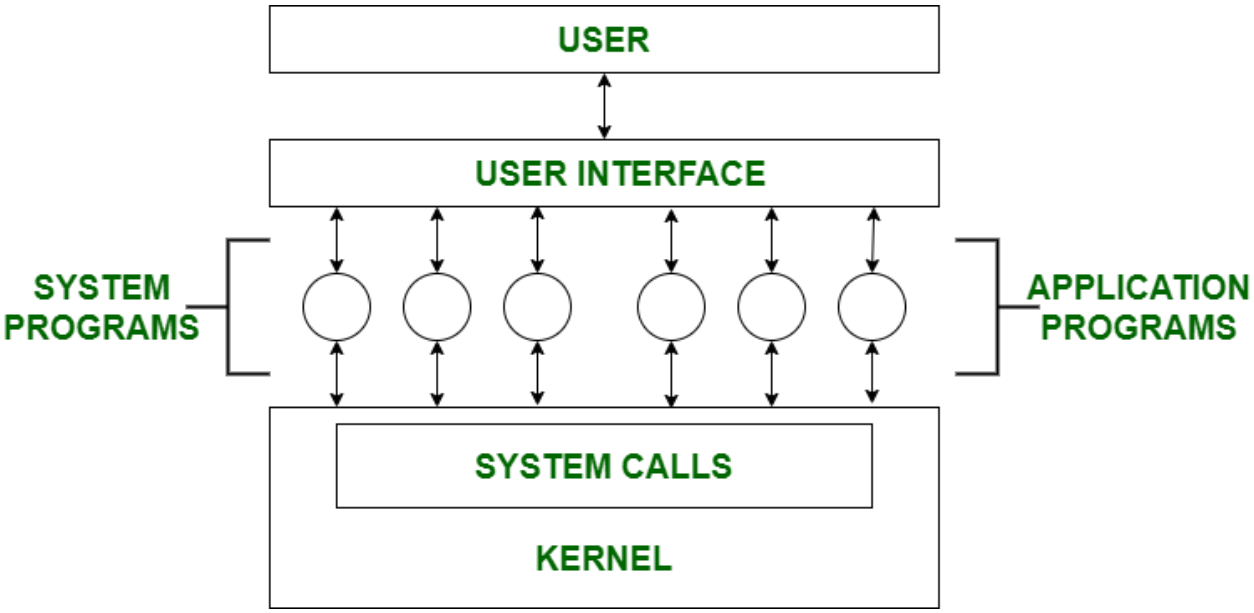


Figure 1.16 System Programs

So here, user can only view up-to-the System Programs he can't see System Calls.

System Programs can be divided into these categories.

File Management : A file is a collection of specific information stored in memory of computer system. File management is defined as process of manipulating files in computer system, it management includes process of creating, modifying and deleting files.It helps to create new files in computer system and placing them at specific locations. It helps in easily and quickly locating these files in computer system. It makes process of sharing of files among different users very easy and user friendly. It helps to stores files in separate folders known as directories. These directories help users to search file quickly or to manage files according to their types or uses. It helps user to modify data of files or to modify he name of file in directories.

Status Information : Information like date, time amount of available memory, or disk space is asked by some of users. Others providing detailed performance, logging and debugging information which is more complex. All this information is formatted and displayed on output devices or printed. Terminal or other output devices or files or a window of GUI is used for showing output of programs.

File Modification : For modifying contents of files we use this. For Files stored on disks or other storage devices we used different types of editors. For searching contents of files or perform transformations of files we use special commands.

Programming-Language support : For common programming languages we use Compilers, Assemblers, Debuggers and interpreters which are already provided to user. It provides all

support to users. We can run any programming languages. All languages of importance are already provided.

Program Loading and Execution : When program is ready after Assembling and compilation, it must be loaded into memory for execution. A loader is part of an operating system that is responsible for loading programs and libraries. It is one of essential stages for starting a program. Loaders, relocatable loaders, linkage editors and Overlay loaders are provided by system.

Communications : Virtual connections among processes, users and computer systems are provided by programs. User can send messages to other user on their screen, User can send e-mail, browsing on web pages, remote login, transformation of files from one user to another.

1.17 Protection and security

Protection and security requires that computer resources such as CPU, softwares, memory etc. are protected. This extends to the operating system as well as the data in the system. This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protected against unauthorized access, viruses, worms etc.

Protection and Security Methods:

The different methods that may provide protect and security for different computer systems are

Authentication

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- **Username/ Password**

  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.

- **User Key/ User Card**

  The users need to punch a card into the card slot or use they individual key on a keypad to access the system.

- **User Attribute Identification**

  Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

1.18 Operating System Design and Implementation

Design:

At the highest level, system design is dominated by the choice of hardware and system type. Beyond this level, the requirements can be divided into two groups: user goals, and system goals. User goals include convenience, reliability, security, and speed. System goals include ease of design, implementation, maintenance, flexibility, and efficiency.

Implementation:

At first, operating systems were written in assembly, but now C/C++ is the language commonly used. Small blocks of assembly code are still needed, especially related to some low level I/O functions in device drivers, turning interrupts on and off and the Test and Set Instruction for Synchronization Facilities. Using higher level languages allows the code to be written faster. It also makes the OS much easier to port to different hardware platforms.

1.19 Virtual Machines

Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example, VirtualBox.When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques. There are additional features of a process that cannot be provided by the hardware alone like system calls and a file system. The virtual machine approach does not provide these additional functionalities but it only provides an interface that is same as basic hardware. Each process is provided with a virtual copy of the underlying computer system.We can create a virtual machine for several reasons, all of which are fundamentally related to the ability to share the same basic hardware yet can also support different execution environments, i.e., different operating systems simultaneously.
The main drawback with the virtual-machine approach involves disk systems. Let us suppose that the physical machine has only three disk drives but wants to support seven virtual machines. Obviously, it cannot allocate a disk drive to each virtual machine, because virtual-machine software itself will need substantial disk space to provide virtual memory and spooling. The solution is to provide virtual disks. Users are thus given their own virtual machines. After which they can run any of the operating systems or software packages that are available on the underlying machine. The virtual-machine software is concerned with multi-programming multiple virtual machines onto a physical machine, but it does not need to consider any user-support software. This arrangement can provide a useful way to divide the problem of designing a multi-user interactive system, into two smaller pieces.

MODULE-II: PROCESS AND CPU SCHEDULING, PROCESS COORDINATION

Process concepts: The process, process state, process control block, threads; Process scheduling: Scheduling queues, schedulers, context switch, preemptive scheduling, dispatcher, scheduling criteria, scheduling algorithms, multiple processor scheduling; Real time scheduling; Thread scheduling; Case studies Linux windows; Process synchronization, the critical section problem; Peterson's solution, synchronization hardware, semaphores and classic problems of synchronization, monitors.

Process concepts:

At the end of the unit students are able to:		
Course Outcomes		**Knowledge Level (Bloom's Taxonomy)**
CO4	Construct the derivations , FIRST set , FOLLOW set on the context free grammar for performing the top-down and bottom up parsing methods.	Apply
CO5	Distinguish top down and bottom up parsing methods for developing parser with the parse tree representation of the input.	Analyze
CO6	Construct LEX and YACC tools for developing a scanner and a parser.	Apply

A process is the unit of work in modern time-sharing systems. A system has a collection of processes : user processes as well as system processes. All these processes can execute concurrently with the CPU multiplexed among them. This module describes what processes are, gives an introduction to process scheduling and explains the various operations that can be done on processes.

2.1 Process:

A process is a program in execution. The execution of a process progresses in a sequential fashion. A program is a passive entity while a process is an active entity. A process includes much more than just the program code. A process includes the text section, stack, data section, program counter, register contents and so on. The text section consists of the set of instructions to be executed for the process. The data section contains the values of initialized and uninitialized global variables in the program. The stack is used whenever there is a function call in the program. A layer is pushed into the stack when a function is called. The arguments to the function and the local variables used in the function are put into the layer of the stack. Once the function call returns to the calling program, the layer of the stack is popped. The text, data and stack sections comprise the address space of the process. The program counter has the address of the next instruction to be executed in the process.

2.2 Process State:

As a process executes, it changes state. The state of a process refers to what the process currently does. A process can be in one of the following states during its lifetime:
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.

The process is in the new state when it is being created. Then the process is moved to the ready state, where it waits till it is taken for execution. There can be many such processes in the ready state. One of these processes will be selected and will be given the processor, and the selected process moves to the running state. A process, while running, may have to wait for I/O or wait for any other event to take place. That process is now moved to the waiting state. After the event for which the process was waiting gets completed, the process is moved back to the ready state. Similarly, if the time-slice of a process ends while still running, the process is moved back to the ready state. Once the process completes execution, it moves to the terminated state.

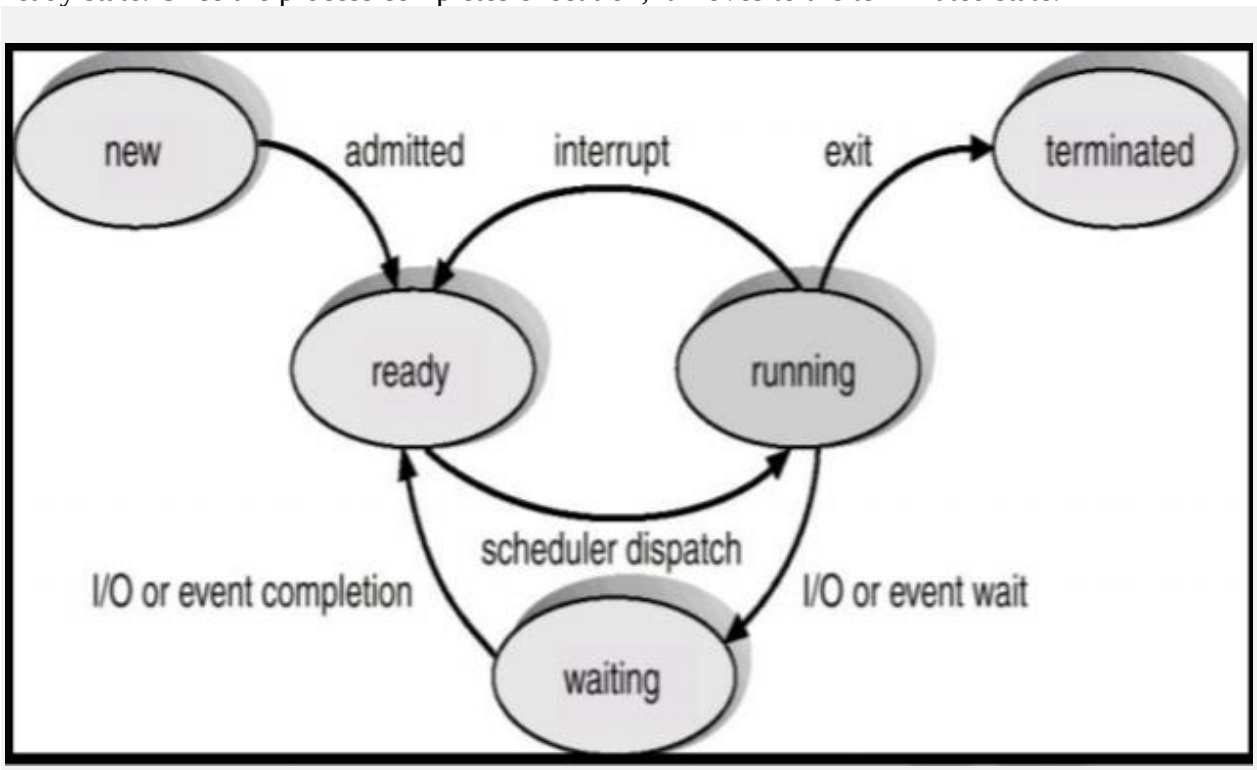


Figure 2.2: Process state diagram

2.3 Process Control Block (PCB)

The information about each process is maintained in the operating system in a process control block, which is also called a task control block. The information present in the PCB includes the following:
Process state :Current state of the process
Program counter: Address of the next instruction to be executed
- CPU registers o Accumulators, index registers, stack pointer, general purpose registers
- CPU scheduling information o Process priority, scheduling parameters
- Memory-management information o Value of base and limit registers
- Accounting information o Amount of CPU used, time limits, process numbers
- I/O status information o List of I/O devices allocated to the process, list of open files and so on.

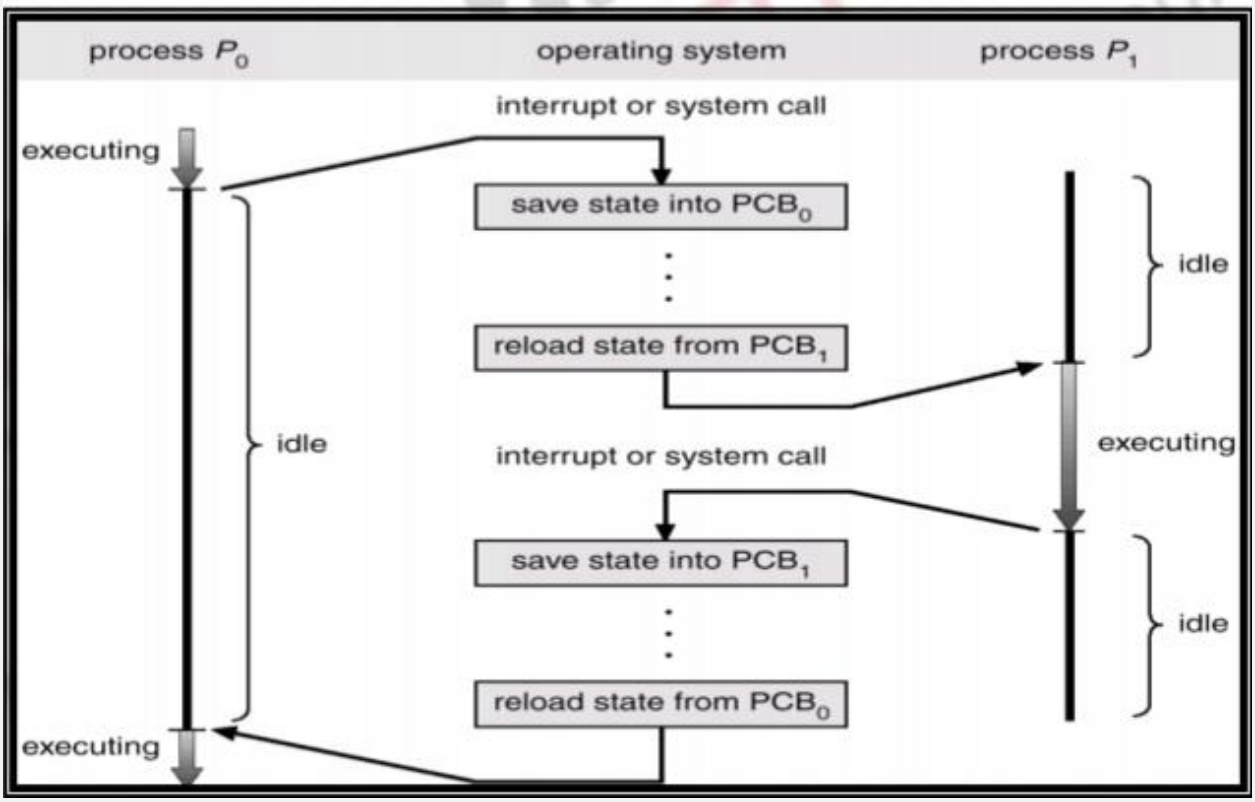


Figure 2.3: Process Control Block

2.4 Threads

A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers. Traditional (heavyweight) processes have a single **thread** of control : There is one program counter, and one sequence of instructions that can be carried out at any given time.

2.5 Process Scheduling

In multi-programmed systems, some process must run at all times, to increase CPU utilization. In time-sharing systems, processes must be switched to increase interaction between the user and the

system. If there are many processes, only one can use the CPU at a particular time and the remaining must wait during that time. When the CPU becomes free, the next process can be scheduled.

2.6 Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues :

- **Job queue**: This queue keeps all the processes in the system.
- **Ready queue** : This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** : The processes which are blocked due to unavailability of an I/O device constitute this queue.

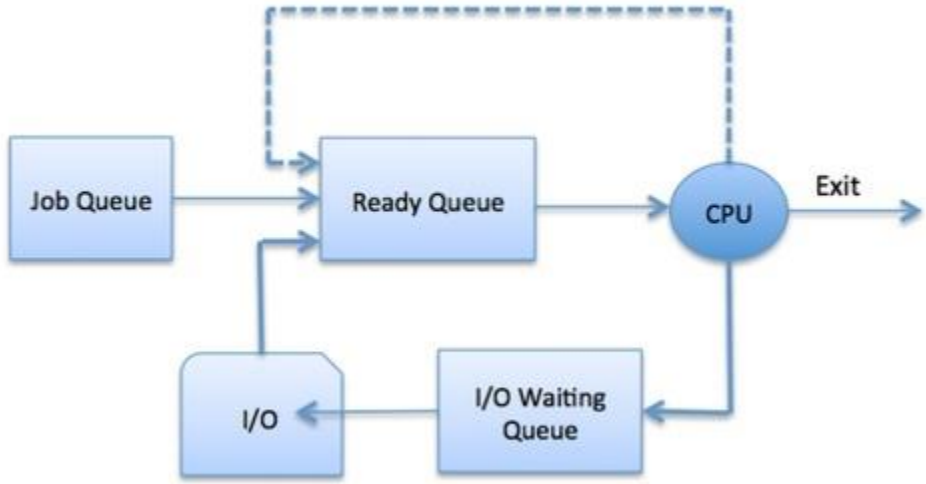


Figure 2.6: Scheduling Queues

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

2.7 Schedulers

Operating system uses various schedulers for the process scheduling described below.

1. Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool. If the job scheduler chooses more IO bound processes then all of the jobs may reside

in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution. A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time. This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

3. Medium term scheduler

Medium term scheduler takes care of the swapped out processes.If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting. Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes. It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

2.8 Context Switch

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

2.9 Premptive Scheduling

Preemptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

Algorithms that are backed by preemptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).

2.10 Dispatcher

A dispatcher is a special program which comes into play after the scheduler. When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue. The dispatcher is the module that gives a process control over the CPU after it has been 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

2.11 Scheduling Criteria

The criteria include the following:
CPU utilisation : The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

Throughput : A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.

Turnaround time : For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.

Waiting time : A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

Response time : In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time.

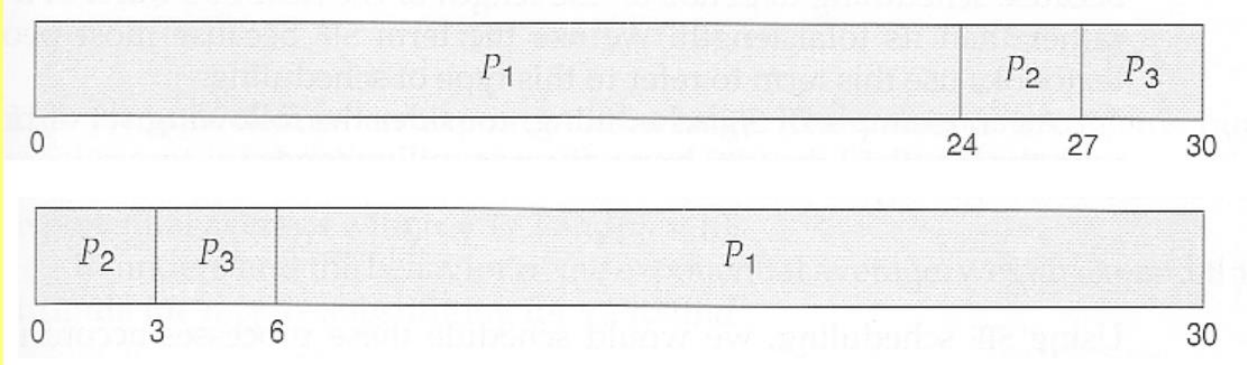
2.12 Scheduling Algorithms

First-Come First-Serve Scheduling (FCFS):

FCFS is very simple - Just a FIFO queue, like customers waiting in line at the bank or the post office or at a copying machine. Unfortunately, however, FCFS can yield some very long average wait times, particularly if the first process to get there takes a long time. For example, consider the following three processes:

Process	Burst Time
P1	24
P2	3
P3	3

In the first Gantt chart below, process P1 arrives first. The average waiting time for the three processes is (0 + 24 + 27) / 3 = 17.0 ms. In the second Gantt chart below, the same three processes have an average wait time of (0 + 3 + 6) / 3 = 3.0 ms. The total run time for the three bursts is the same, but in the second case two of the three finish much quicker, and the other process is only delayed by a short amount.

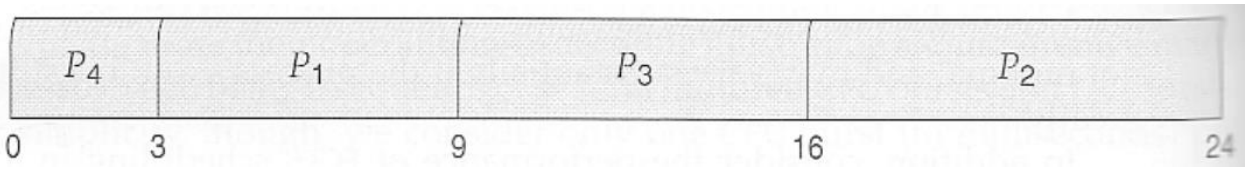


FCFS can also block the system in a busy dynamic system in another way, known as the convoy effect. When one CPU intensive process blocks the CPU, a number of I/O intensive processes can get backed up behind it, leaving the I/O devices idle. When the CPU hog finally relinquishes the CPU, then the I/O processes pass through the CPU quickly, leaving the CPU idle while everyone queues up for I/O, and then the cycle repeats itself when the CPU intensive process gets back to the ready queue.

Shortest-Job-First Scheduling (SJF)

The idea behind the SJF algorithm is to pick the quickest fastest little job that needs to be done, get it out of the way first, and then pick the next smallest fastest job to do next.For example, the Gantt chart below is based upon the following CPU burst times, (and the assumption that all jobs arrive at the same time.)

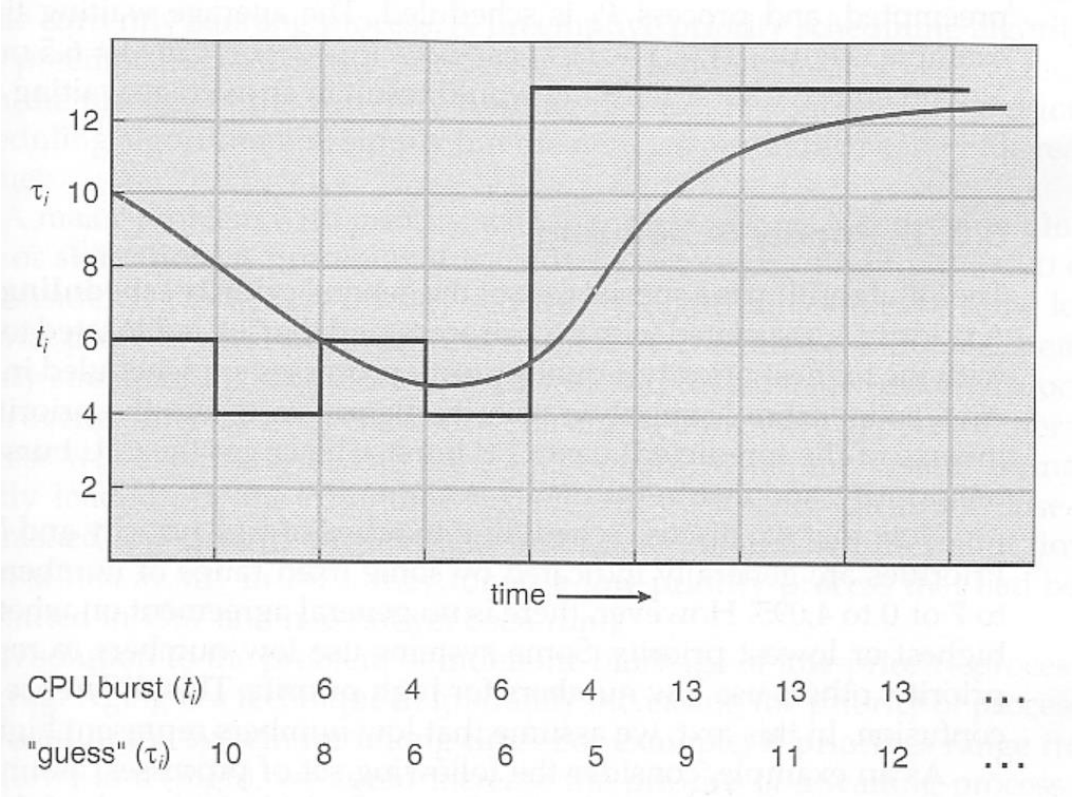
Process	Burst Time
P1	6
P2	8
P3	7
P4	3



- In the case above the average wait time is (0 + 3 + 9 + 16) / 4 = 7.0 ms, (as opposed to 10.25 ms for FCFS for the same processes.)

- SJF can be proven to be the fastest scheduling algorithm, but it suffers from one important problem: How do you know how long the next CPU burst is going to be?
    - For long-term batch jobs this can be done based upon the limits that users set for their jobs when they submit them, which encourages them to set low limits, but risks their having to re-submit the job if they set the limit too low. However that does not work for short-term CPU scheduling on an interactive system.
    - Another option would be to statistically measure the run time characteristics of jobs, particularly if the same tasks are run repeatedly and predictably. But once again that really isn't a viable option for short term CPU scheduling in the real world.
    - A more practical approach is to ***predict*** the length of the next burst, based on some historical measurement of recent burst times for this process. One simple, fast, and relatively accurate method is the ***exponential average***, which can be defined as follows. (The book uses tau and t for their variables, but those are hard to distinguish from one another and don't work well in HTML.)

    estimate[i + 1] = alpha * burst[i] + (1.0 - alpha) * estimate[i]
    The previous estimate contains the history of all previous times, and alpha serves as a weighting factor for the relative importance of recent data versus past history. If alpha is 1.0, then past history is ignored, and we assume the next burst will be the same length as the last burst. If alpha is 0.0, then all measured burst times are ignored, and we just assume a constant burst time.



SJF can be either preemptive or non-preemptive. Preemption occurs when a new process arrives in the ready queue that has a predicted burst time shorter than the time remaining in the process whose burst is currently on the CPU. Preemptive SJF is sometimes referred to as shortest remaining time first scheduling**.** For example, the following Gantt chart is based upon the following data:

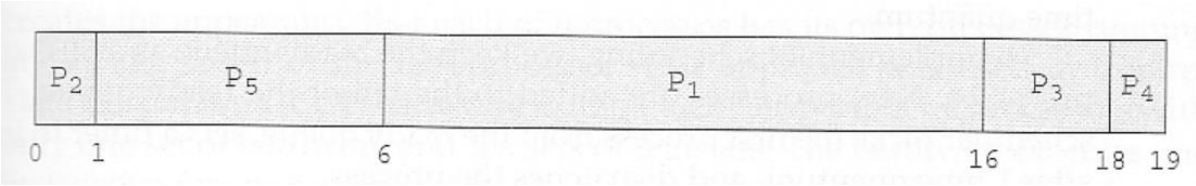
Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
p4	3	5

Priority Scheduling

Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first. (SJF uses the inverse of the next expected burst time as its priority - The smaller the expected burst, the higher the priority.) Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether "high" priorities use large numbers or small numbers. This book uses low number for high priorities, with 0 being the highest possible priority. For example, the

following Gantt chart is based upon these process burst times and priorities, and yields an average waiting time of 8.2 ms

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



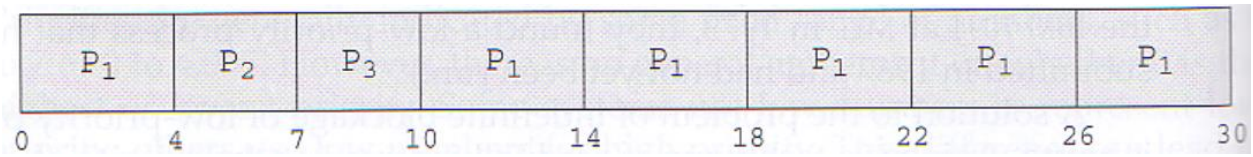
Priorities can be assigned either internally or externally. Internal priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job, fees paid, politics, etc. Priority scheduling can be either preemptive or non-preemptive. Priority scheduling can suffer from a major problem known as **indefinite blocking**, or **starvation**, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.
If this problem is allowed to occur, then processes will either run eventually when the system load lightens , or will eventually get lost when the system is shut down or crashes.
One common solution to this problem is ***aging***, in which priorities of jobs increase the longer they wait. Under this scheme a low-priority job will eventually get its priority raised high enough that it gets run.

Round Robin Scheduling
Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called **time quantum**. When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.

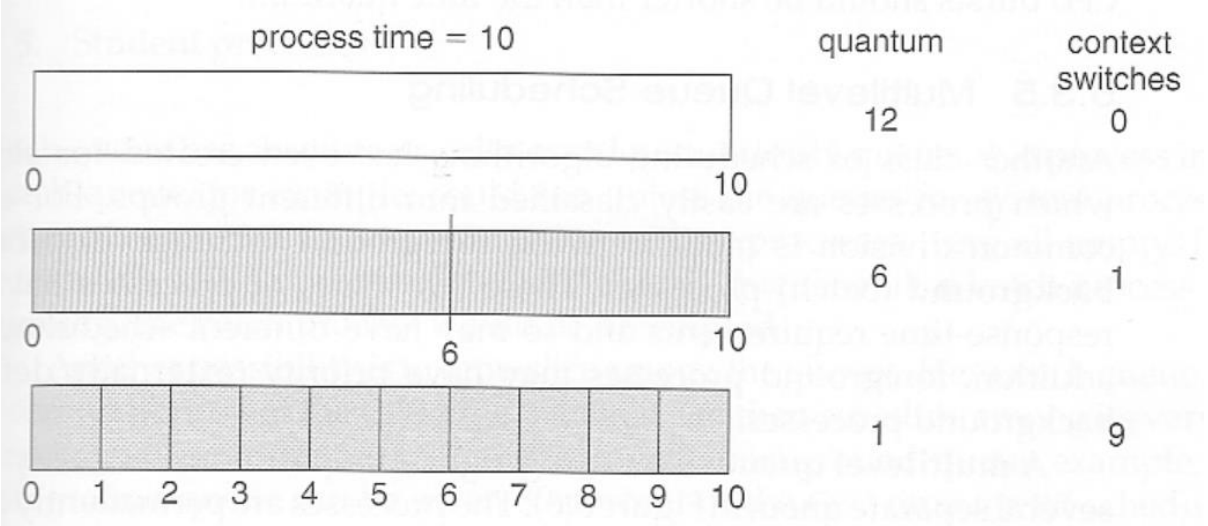
- If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
- If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.

The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on. RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms. In the following example the average wait time is 5.66 ms.

Process	Burst Time
P1	24
P2	3
P3	3



The performance of RR is sensitive to the time quantum selected. If the quantum is large enough, then RR reduces to the FCFS algorithm; If it is very small, then each process gets 1/nth of the processor time and share the CPU equally. But, a real system invokes overhead for every context switch, and the smaller the time quantum the more context switches there are. Most modern systems use time quantum between 10 and 100 milliseconds, and context switch times on the order of 10 microseconds, so the overhead is small relative to the time quantum.



Turn around time also varies with quantum time, in a non-apparent manner. In general, turnaround time is minimized if most processes finish their next cpu burst within one time quantum. For example, with three processes of 10 ms bursts each, the average turnaround time for 1 ms quantum is 29, and for 10 ms quantum it reduces to 20. However, if it is made too large, then RR just degenerates to FCFS. A rule of thumb is that 80% of CPU bursts should be smaller than the time quantum.

2.13 Multiple processor scheduling

In the multiprocessor scheduling, there are multiple CPU's which share the load so that various process run simultaneously. In general, the multiprocessor scheduling is complex as compared to single processor scheduling. In the multiprocessor scheduling, there are many processors and they

are identical and we can run any process at any time. The multiple CPU's in the system are in the close communication which shares a common bus, memory and other peripheral devices. So we can say that the system is a tightly coupled system. These systems are used when we want to process a bulk amount of data. These systems are mainly used in satellite, weather forecasting etc.

2.14 Real time scheduling

Real-time systems are systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In particular, these tasks are related to control of certain events (or) reacting to them. Real-time tasks can be classified as hard real-time tasks and soft real-time tasks. A hard real-time task must be performed at a specified time which could otherwise lead to huge losses. In soft real-time tasks, a specified deadline can be missed. This is because the task can be rescheduled (or) can be completed after the specified time.
In real-time systems, the scheduler is considered as the most important component which is typically a short-term task scheduler. The main focus of this scheduler is to reduce the response time associated with each of the associated processes instead of handling the deadline. If a preemptive scheduler is used, the real-time task needs to wait until its corresponding tasks time slice completes. In the case of a non-preemptive scheduler, even if the highest priority is allocated to the task, it needs to wait until the completion of the current task. This task can be slow (or) of the lower priority and can lead to a longer wait. A better approach is designed by combining both preemptive and non-preemptive scheduling. This can be done by introducing time-based interrupts in priority based systems which means the currently running process is interrupted on a time-based interval and if a higher priority process is present in a ready queue, it is executed by preempting the current process.

2.15 Thread Scheduling

Scheduling of threads involves two boundary scheduling: Scheduling of user level threads (ULT) to kernel level threads (KLT) via lightweight process (LWP) by the application developer. Scheduling of kernel level threads by the system scheduler to perform different unique os functions.

2.16 Case studies Linux windows

The **Linux open source operating system,** or Linux OS, is a freely attributable, cross-platform operating system based on Unix that can be installed on PCs, laptops, net-books, mobile and tablet devices, video game consoles, servers, supercomputers and more.The Linux OS is frequently packaged as a Linux distribution for both desktop and server use, and includes the Linux kernel (the core of the operating system) as well as supporting tools and libraries.

Windows Operating System, computer operating system (OS) developed by Microsoft Corporation to run personal computers (PCs). Featuring the first graphical user interface (GUI) for IBM-compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run some version of Windows. The first version of Windows, released in 1985, was simply

a GUI offered as an extension of Microsoft's existing disk operating system, or MS-DOS. Based in part on licensed concepts that Apple Inc. had used for its Macintosh System Software, Windows for the first time allowed DOS users to visually navigate a virtual desktop, opening graphical "windows" displaying the contents of electronic folders and files with the click of a mouse button, rather than typing commands and directory paths at a text prompt.

2.17 Process Synchronization

Process Synchronization means sharing system resources by processes in a 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. Process Synchronization was introduced to handle problems that arose while multiple process executions.

2.18 The critical sections problem

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

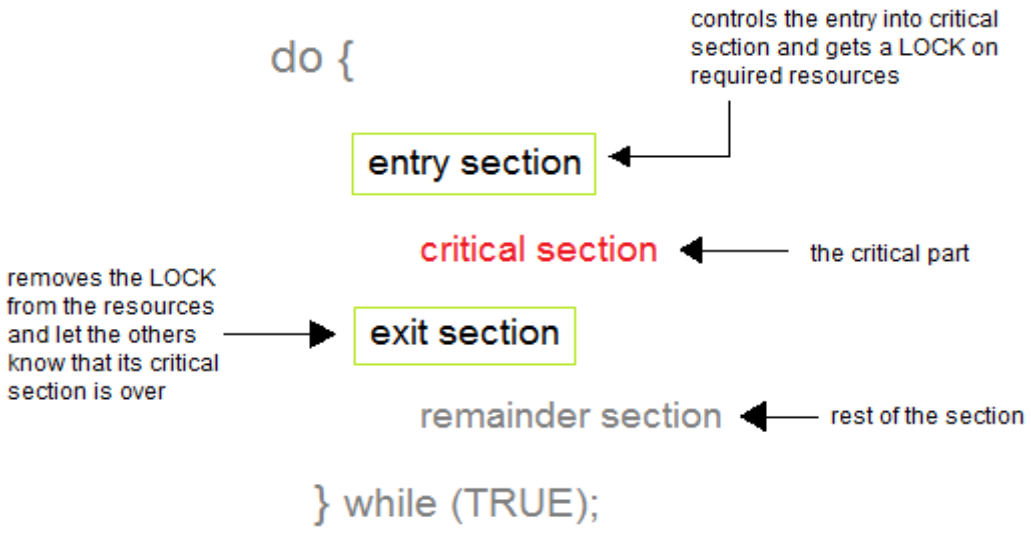


Figure 2.18:Critical section problem

Solution to Critical Section Problem

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

1. Mutual Exclusion

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

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

2.19 Petersons solution

Peterson's solution is a widely used solution to critical section problems. This algorithm was developed by a computer scientist Peterson that's why it is named as a Peterson's solution. In this solution, when a process is executing in a critical state, then the other process only executes the rest of the code, and the opposite can happen. This method also helps to make sure that only a single process runs in the critical section at a specific time.

2.20 Synchronization hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified. In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment. Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released. As the resource is locked while a process executes its critical section hence no other process can access it.

2.21 Semaphores and classic problems of Synchronization

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows −

- **Wait**

  The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
  while (S<=0);

  S--;
}
```

- **Signal**

  The signal operation increments the value of its argument S.

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

Types of Semaphores:

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows −

Counting Semaphores: These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

Binary Semaphores: The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal

operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Classic problems of Synchronization:

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

Bounded Buffer Problem

This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes .Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer 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.

Dining Philosophers Problem

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

The Readers Writers Problem

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

2.22 Monitors

Monitors are used for process synchronization. With the help of programming languages, we can use a monitor to achieve mutual exclusion among the processes. In other words, monitors are defined as the construct of programming language, which helps in controlling shared data access. The Monitor is a module or package which encapsulates shared data structure, procedures, and the synchronization between the concurrent procedure invocations.

MODULE-III: MEMORY MANAGEMENT AND VIRTUAL MEMORY

Logical and physical address space: Swapping, contiguous memory allocation, paging, structure of page table.

Segmentation: Segmentation with paging, virtual memory, demand paging; Performance of demand paging: Page replacement, page replacement algorithms, allocation of frames, thrashing

At the end of the unit students are able to:		
Course Outcomes		**Knowledge Level (Bloom's Taxonomy)**
CO6	Describe syntax directed definitions & translations for performing Semantic Analysis.	Remember
CO7	Classify the different intermediate forms for conversion of syntax translations into Intermediate Code.	Analyze

Process concepts:

3.1 Logical and physical address space:

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective. The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

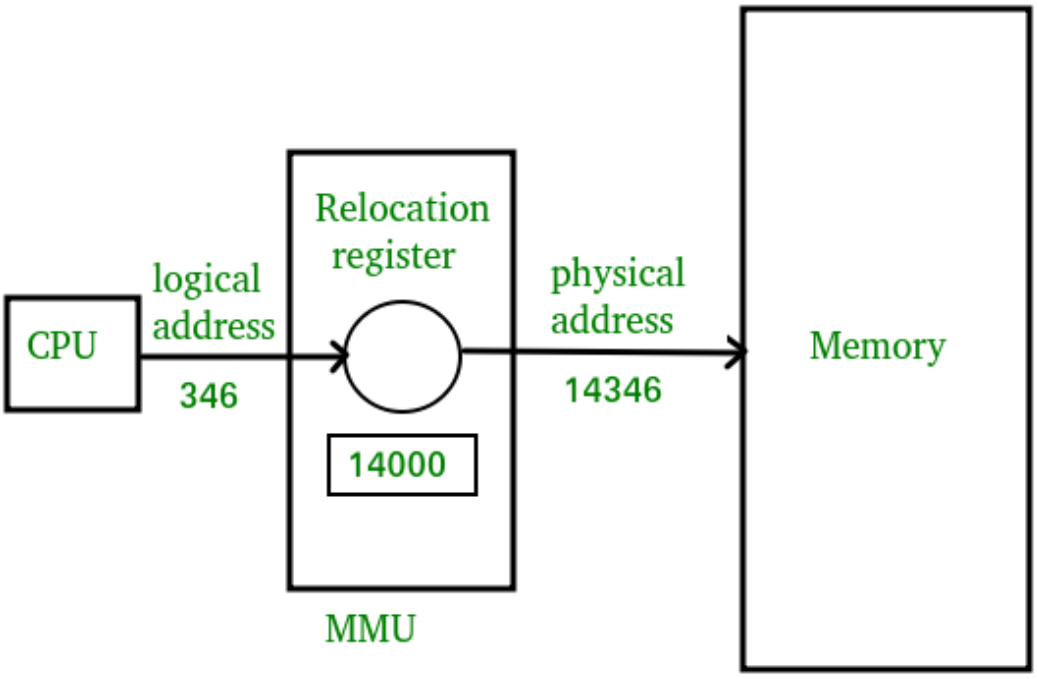


Figure 3.1: Logical and Physical Address

3.2 Swapping

Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space. The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in RAM.

3.3 Contiguous memory allocation

Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space.

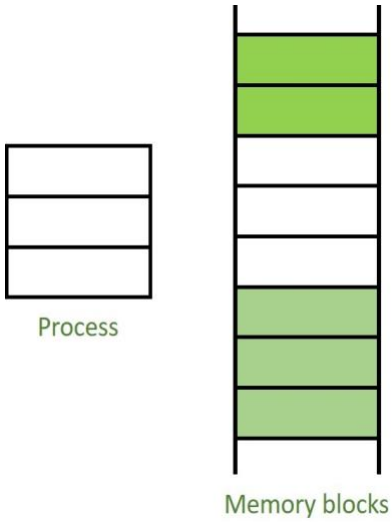


Figure 3.3 Contiguous memory allocation

The main memory is a combination of two main portions- one for the operating system and other for the user program. We can implement/achieve contiguous memory allocation by dividing the memory partitions into fixed size partitions.

3.4 Paging

Paging is a storage mechanism that allows OS to retrieve processes from the secondary storage into the main memory in the form of pages. In the Paging method, the main memory is divided into small fixed-size blocks of physical memory, which is called frames. The size of a frame should be kept the same as that of a page to have maximum utilization of the main memory and to avoid external fragmentation. Paging is used for faster access to data, and it is a logical concept.

3.5 Structure of page table

Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses. Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes. Physical addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

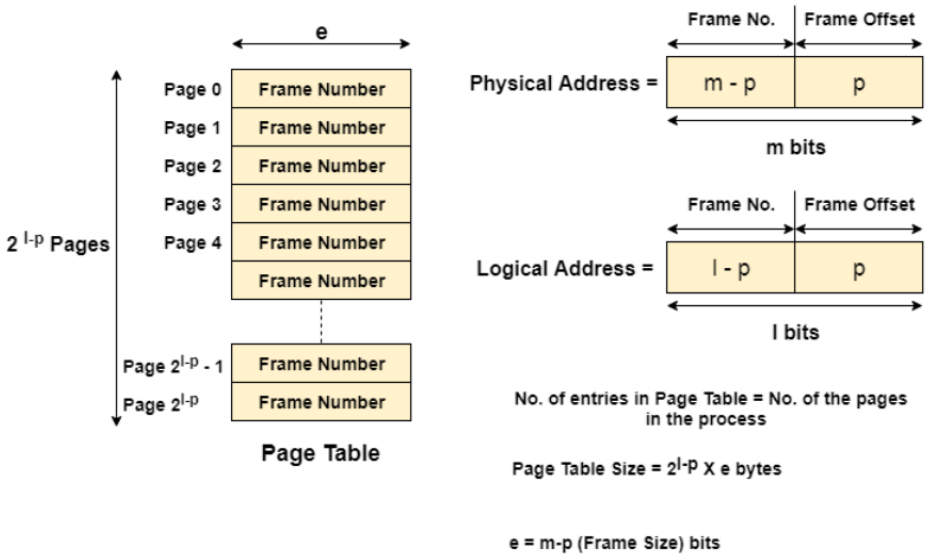


Figure 3.5: Structure of page table

The CPU always accesses the processes through their logical addresses. However, the main memory recognizes physical addresses only. In this situation, a unit named as a Memory Management Unit comes into the picture. It converts the page number of the logical address to the frame number of the physical address. The offset remains the same in both the addresses. To perform this task, the Memory Management unit needs a special kind of mapping which is done by page table. The page table stores all the Frame numbers corresponding to the page numbers of the page table.

3.6 Segmentation:

In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process. The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.
Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

3.7 Segmentation with paging

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques. In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

1. Pages are smaller than segments.

2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number : It points to the appropriate Segment Number.

Page Number : It Points to the exact page within the segment

Page Offset : Used as an offset within the page frame

Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.

3.8 Virtual Memory

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

3.9 Demand paging

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

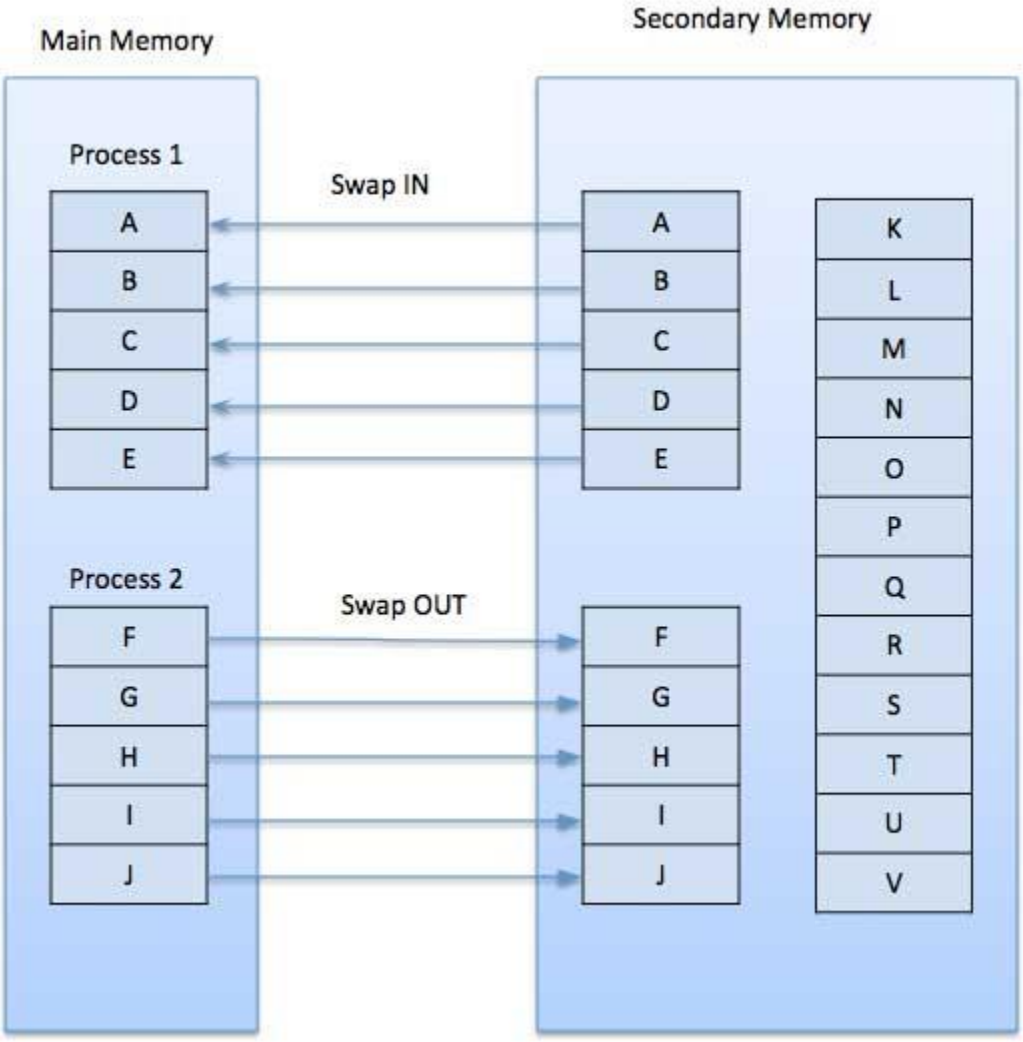


Figure 3.9: Demand Paging

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

3.10 Performance of demand paging:

Demand paging can significantly affect the performance of a computer system. To see why, let's compute the effective access time for a demand-paged memory. the memory-access time, denoted ma,ranges from 10 to 200 nanoseconds.
effective access time = $(1 - p) \times ma + p \times$ page fault time.
where p is probability of page fault.

3.11 Page replacement

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

3.12 Page replacement algorithms

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

Reference String

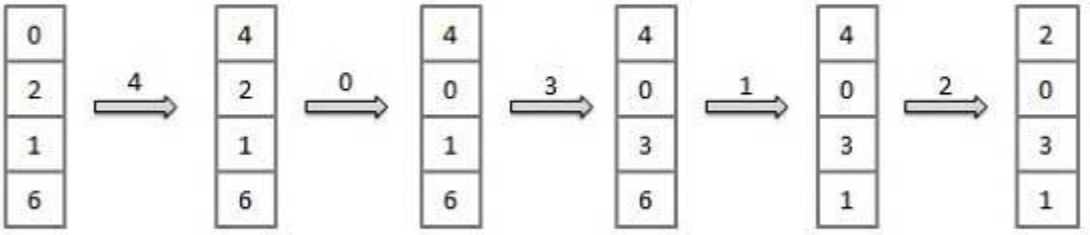
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things. For a given page size, we need to consider only the page number, not the entire address. If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault. For example, consider the following sequence of addresses :123,215,600,1234,76,96 If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

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

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

Misses : x x x x x x x x x



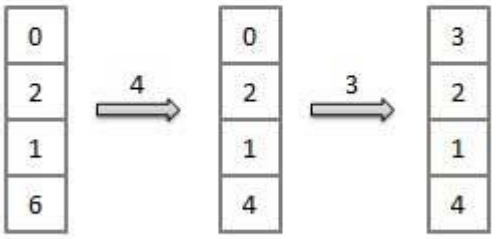
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

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

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

Misses : x x x x x x



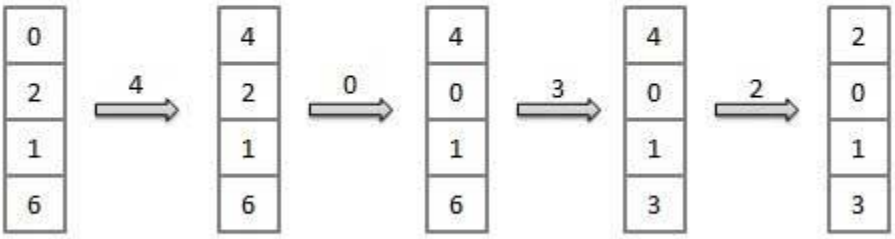
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

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

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

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm:

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

Least frequently Used(LFU) algorithm:

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

Most frequently Used(MFU) algorithm:

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

3.13 Allocation of frames

Frame allocation algorithms :

The two algorithms commonly used to allocate frames to a process are:

Equal allocation: In a system with x frames and y processes, each process gets equal number of frames, i.e. x/y. For instance, if the system has 48 frames and 9 processes, each process will get 5 frames. The three frames which are not allocated to any process can be used as a free-frame buffer pool.

Proportional allocation: Frames are allocated to each process according to the process size. For a process p_i of size s_i, the number of allocated frames is $\mathbf{a_i = (s_i/S)*m}$, where S is the sum of the sizes of all the processes and m is the number of frames in the system. For instance, in a system with 62 frames, if there is a process of 10KB and another process of 127KB, then the

first process will be allocated (10/137)*62 = 4 frames and the other process will get (127/137)*62 = 57 frames.

3.14 Thrashing

Thrashing is a computer activity that makes little or no progress. Usually, this happens either of limited resources or exhaustion of memory. It arises when a page fault occurs. Page fault arises when memory access of virtual memory space does not map to the content of RAM.
The effect of Thrashing is
1.CPU becomes idle.
2.Decreasing the utilization increases the degree of multiprogramming and hence bringing more processes at a time which in fact increases the thrashing exponentially.

MODULE-IV: FILE SYSTEM INTERFACE, MASS-STORAGE STRUCTURE

The concept of a file, access methods, directory structure, file system mounting, file sharing, protection, file system structure, file system implementation, allocation methods, free space management, directory implementation, efficiency and performance; Overview of mass storage structure: Disk structure, disk attachment, disk scheduling, disk management, swap space management; Dynamic memory allocation**:** Basic concepts; Library functions.

Process concepts:

At the end of the unit students are able to:		
Course Outcomes		**Knowledge Level (Bloom's Taxonomy)**
CO9	Demonstrate type systems for performing the static and dynamic type checking	Understand
CO10	Describe the run-time memory elements for storage allocation strategies which include procedure calls, local variable allocation, and dynamic memory allocation.	Understand

4.1 The concept of a file:

A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes. It is a method of data collection that is used as a medium for giving input and receiving output from that program. In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user. Every File has a logical location where they are located for storage and retrieval.

4.2 Access Methods:

File access is a process that determines the way that files are accessed and read into memory. Generally, a single access method is always supported by operating systems. Though there are some operating system which also supports multiple access methods.

Three file access methods are:

- Sequential access
- Direct random access
- Index sequential access

Sequential Access:

In this type of file access method, records are accessed in a certain pre-defined sequence. In the sequential access method, information stored in the file is also processed one by one. Most compilers access files using this access method.

Random Access:

The random access method is also called direct random access. This method allow accessing the record directly. Each record has its own address on which can be directly accessed for reading and writing.

Sequential Access:

This type of accessing method is based on simple sequential access. In this access method, an index is built for every file, with a direct pointer to different memory blocks. In this method, the Index is searched sequentially, and its pointer can access the file directly. Multiple levels of indexing can be used to offer greater efficiency in access. It also reduces the time needed to access a single record.

Space Allocation:

In the Operating system, files are always allocated disk spaces.

Three types of space allocation methods are:

- Linked Allocation
- Indexed Allocation
- Contiguous Allocation

Contiguous Allocation:

In this method every file users a contiguous address space on memory. Here, the OS assigns disk address is in linear order. In the contiguous allocation method, external fragmentation is the biggest issue.

Linked Allocation:

In this method, every file includes a list of links. The directory contains a link or pointer in the first block of a file. With this method, there is no external fragmentation This File allocation method is used for sequential access files. This method is not ideal for a direct access file.

Indexed Allocation:

In this method, Directory comprises the addresses of index blocks of the specific files. An index block is created, having all the pointers for specific files. All files should have individual index blocks to store the addresses for disk space.

4.3 Directory Structure

A single directory may or may not contain multiple files. It can also have sub-directories inside the main directory. Information about files is maintained by Directories. In Windows OS, it is called folders.

Following is the information which is maintained in a directory:

- **Name** The name which is displayed to the user.
- **Type**: Type of the directory.
- **Position**: Current next-read/write pointers.
- **Location**: Location on the device where the file header is stored.
- **Size** : Number of bytes, block, and words in the file.

- **Protection**: Access control on read/write/execute/delete.
- **Usage**: Time of creation, access, modification

4.4 File system mounting

Operating system organises files on a disk using filesystem. And these filesystems differ with operating systems. We have greater number of filesystems available for linux. One of the great benefits of linux is that we can access data on many different file systems even if these filesystems are from different operating system. In order to access the file system in the linux first we need to mount it. Mounting refers to making a group of files in a file system structure accessible to user or group of users. It is done by attaching a root directory from one file system to that of another. This ensures that the other directory or device appears as a directory or subdirectory of that system.

4.5 File sharing

File sharing is the public or private sharing of computer data or space in a network with various levels of access privilege. While files can easily be shared outside a network (for example, simply by handing or mailing someone your file on a diskette), the term *file sharing* almost always means sharing files in a network, even if in a small local area network. File sharing allows a number of people to use the same file or file by some combination of being able to read or view it, write to or modify it, copy it, or print it. Typically, a file sharing system has one or more administrators. Users may all have the same or may have different levels of access privilege. File sharing can also mean having an allocated amount of personal file storage in a common file system.

4.6 Protection

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

Need of Protection:

- To prevent the access of unauthorized users and
- To ensure that each active programs or processes in the system uses resources only as the stated policy,
- To improve reliability by detecting latent errors.
-

4.7 File system structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file. Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

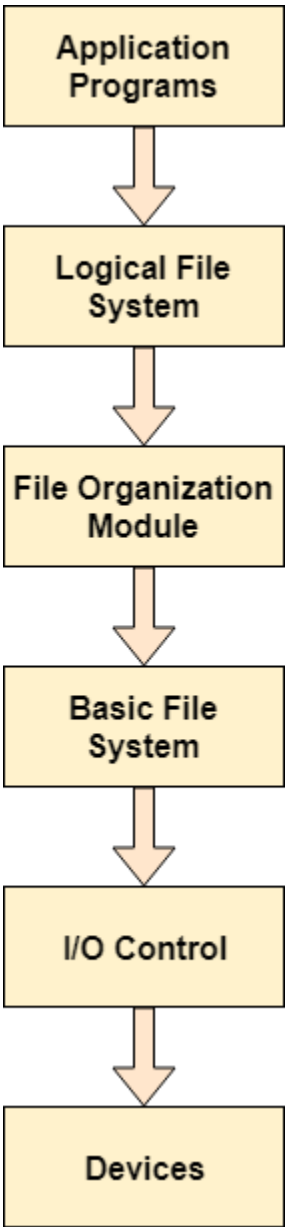


Figure 4.7: File System Structure

4.8 File system implementation

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

File system organized in many layers :

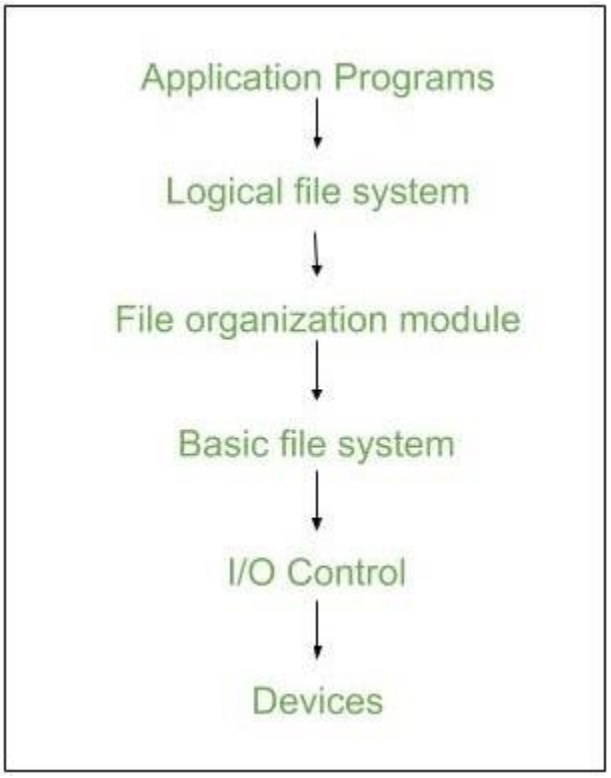


Figure 4.8 File System Implementation

I/O Control level: Device drivers acts as interface between devices and Os, they help to transfer data between disk and main memory. It takes block number a input and as output it gives low level hardware specific instruction.

Basic file system : It Issues general commands to device driver to read and write physical blocks on disk. It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

File organization Module: It has information about files, location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.

Logical file system : It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

4.9 Allocation Methods

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

- Contiguous Allocation
- Linked Allocation

- Indexed Allocation

1. Contiguous Allocation:

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

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

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

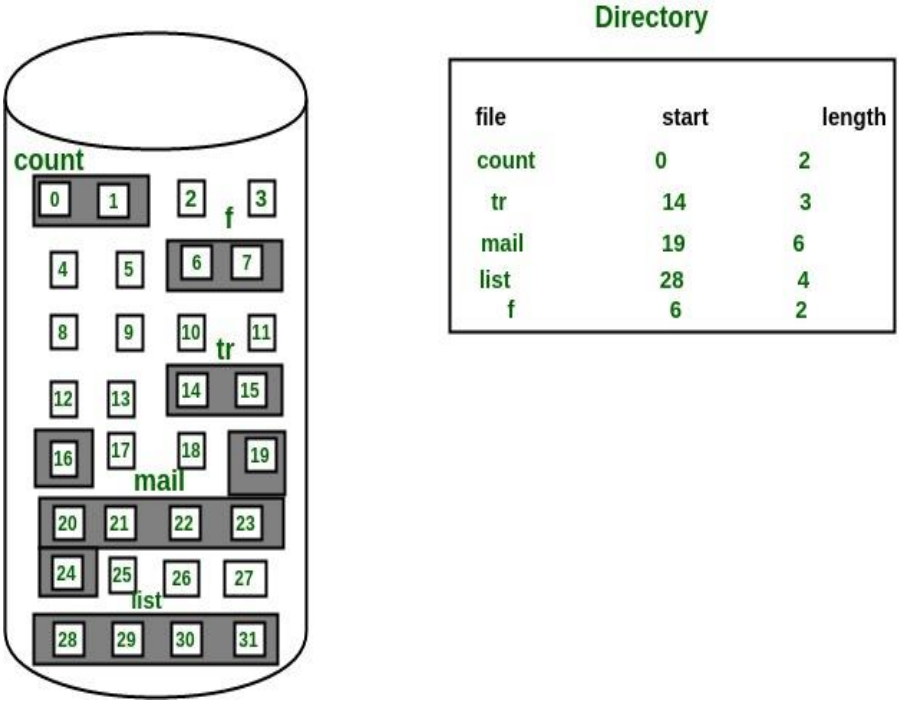


Figure 4.9a :Contiguous Allocation

Advantages:

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

Disadvantages:

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

2. Linked List Allocation
In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.
The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

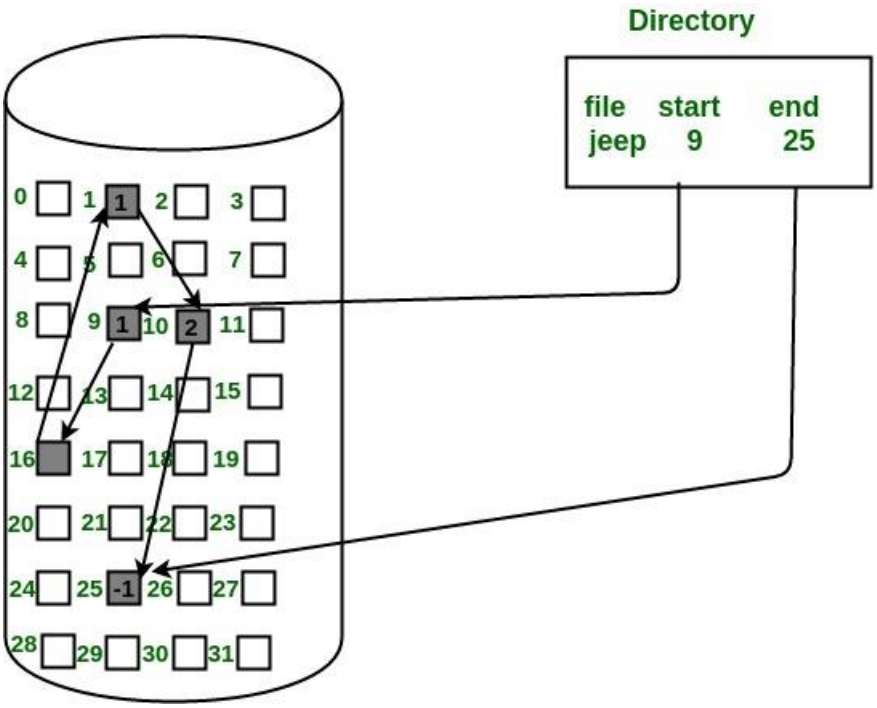


Figure 4.9b: Linked list Allocation

Advantages:
This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory. This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.
Disadvantages:
Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower. It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers. Pointers required in the linked allocation incur some extra overhead.

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

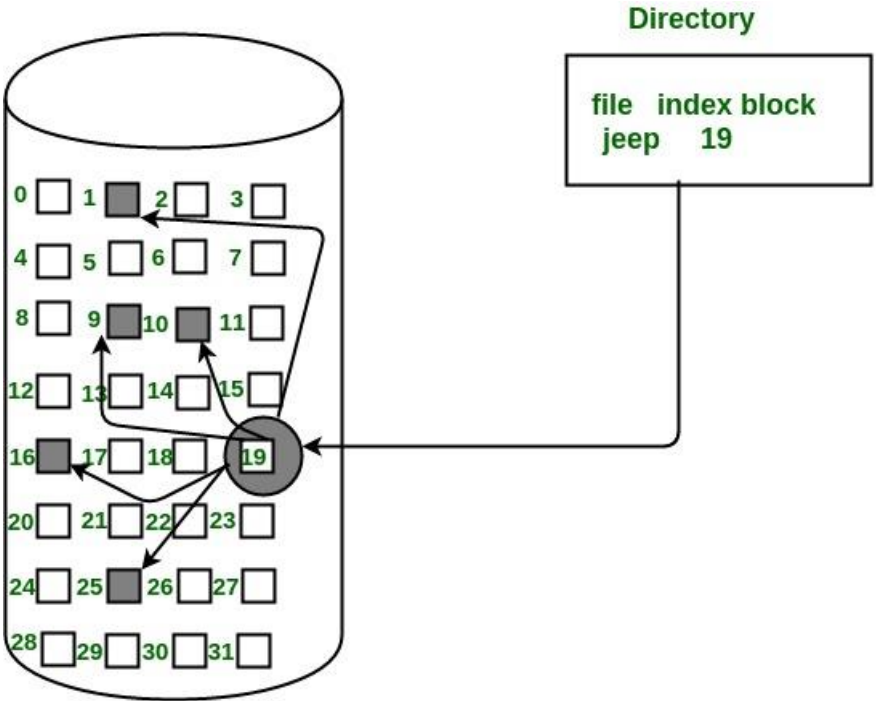


Figure 4.9c :Indexed Allocation

Advantages:
This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks. It overcomes the problem of external fragmentation.
Disadvantages:
The pointer overhead for indexed allocation is greater than linked allocation.
For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.
For files that are very large, single index block may not be able to hold all the pointers.
contain the address of the blocks containing the file data.

4.10 Free Space Management

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

Bitmap or Bit vector :
A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1

indicates a free block. The given instance of disk blocks on the disk can be represented by a bitmap of 16 bits as: **0000111000000110**.

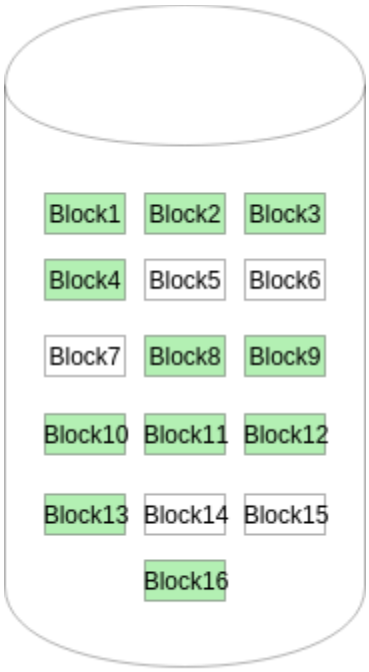


Figure 4.10a: Bitmap Vector

Advantages :
Simple to understand.. Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.
The block number can be calculated as:
(number of bits per word) *(number of 0-values words) + offset of bit first bit 1 in the non-zero word .we scan the bitmap sequentially for the first non-zero word.
The first group of 8 bits (00001110) constitute a non-zero word since all bits are not 0. After the non-0 word is found, we look for the first 1 bit. This is the 5th bit of the non-zero word. So, offset = 5.
Therefore, the first free block number = 8*0+5 = 5.
Linked List : In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.The free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.
A drawback of this method is the I/O required for free space list traversal.
Grouping : This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks.

An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

Counting : This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.

Every entry in the list would contain:

- Address of first free disk block
- A number n

4.11 Directory Implementation

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system. The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used in these days.

1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics

When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

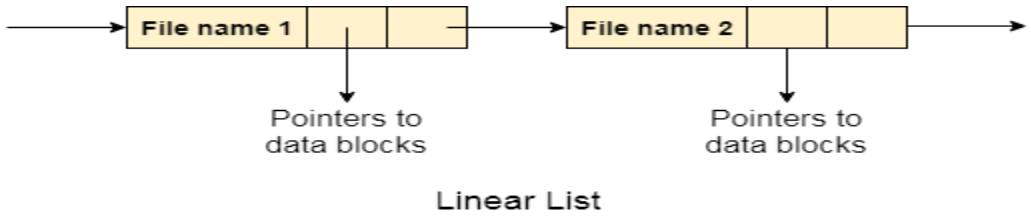


Figure 4.11a: Linear list

2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists. A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory. Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are

checked using the key and if an entry found then the corresponding file will be fetched using the value.

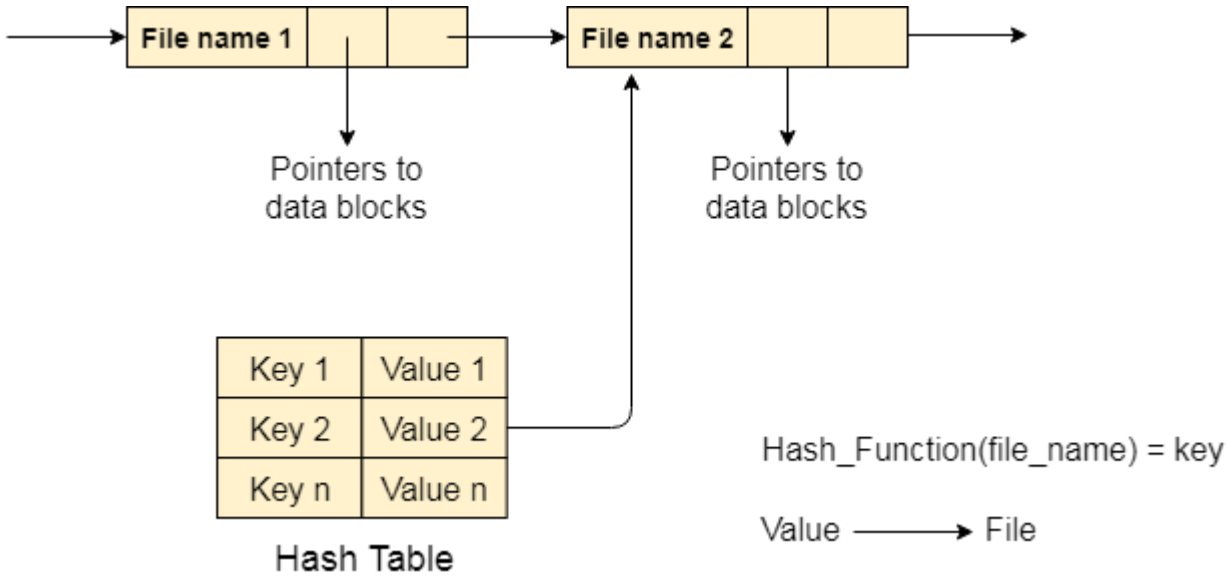


Figure 4.11b: Hash Table

4.12 Efficiency and Performance

Efficiency

UNIX pre-allocates inodes, which occupies space even before any files are created. UNIX also distributes inodes across the disk, and tries to store data files near their inode, to reduce the distance of disk seeks between the inodes and the data. Some systems use variable size clusters depending on the file size. The more data that is stored in a directory (e.g. last access time), the more often the directory blocks have to be re-written. Kernel table sizes used to be fixed, and could only be changed by rebuilding the kernels. Modern tables are dynamically allocated, but that requires more complicated algorithms for accessing them.

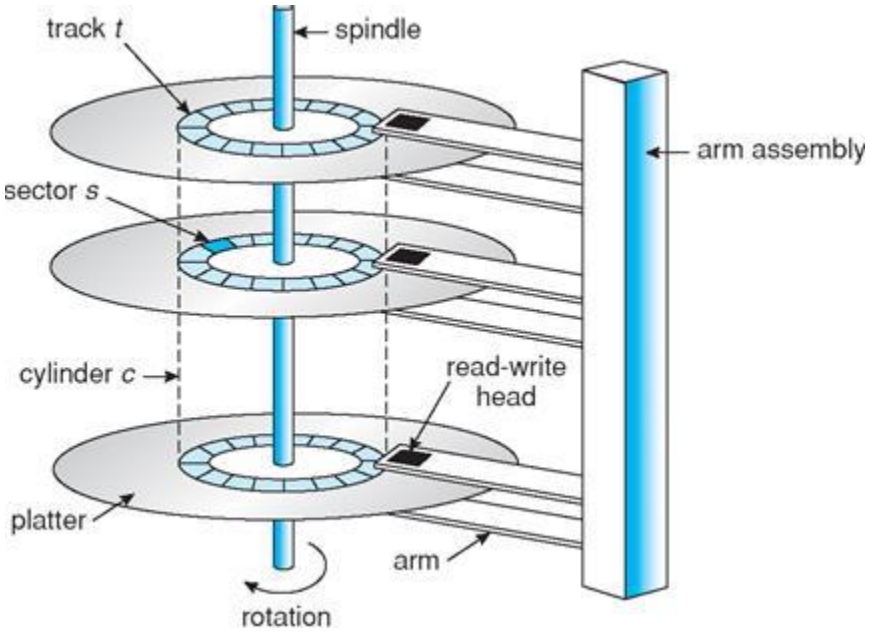
Performance

Disk controllers generally include on-board caching. When a seek is requested, the heads are moved into place, and then an entire track is read, starting from whatever sector is currently under the heads (reducing latency.) The requested sector is returned and the unrequested portion of the track is cached in the disk's electronics. Some OSes cache disk blocks they expect to need again in a buffer cache. A page cache connected to the virtual memory system is actually more efficient as memory addresses do not need to be converted to disk block addresses and back again. Some

systems (Solaris, Linux, Windows 2000, NT, XP) use page caching for both process pages and file data in a unified virtual memory.

.

4.13 Overview of mass storage structure



Magnetic disks provide bulk of secondary storage of modern computers:

Drives rotate at 60 to 200 times per second. Transfer rate is the rate at which data flow between drive and computer. Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency). Head crash results from disk head making contact with the disk surface. Disks can be removable. Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. Mapping proceeds in order through that track, then the rest of the tracks in that cylinder and then through the rest of the cylinders from outermost to innermost.

4.14 Disk Scheduling

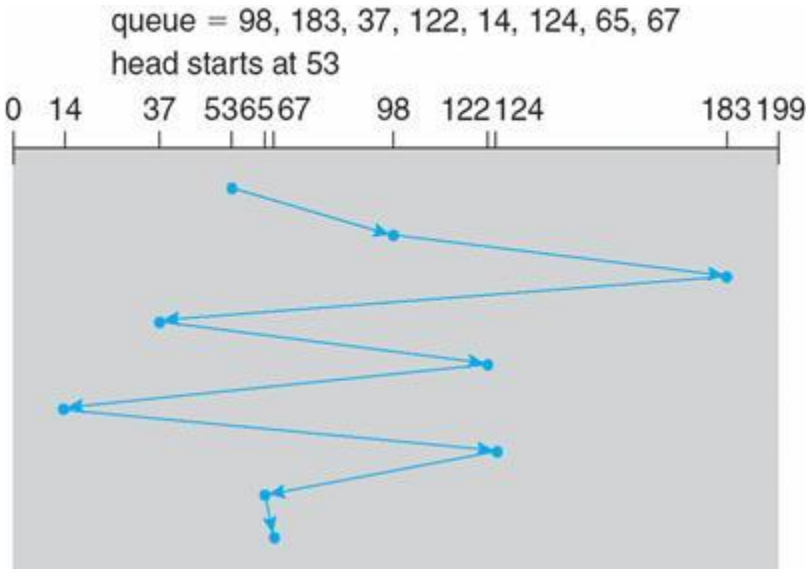
The operating system is responsible for using hardware efficiently for the disk drives, this means having a fast access time and disk bandwidth. n Access time has two major components F Seek time is the time for the disk are to move the heads to the cylinder containing the desired sector. F Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head. n Minimize seek time n Seek time » seek distance n Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

4.15 Disk Scheduling Algorithms

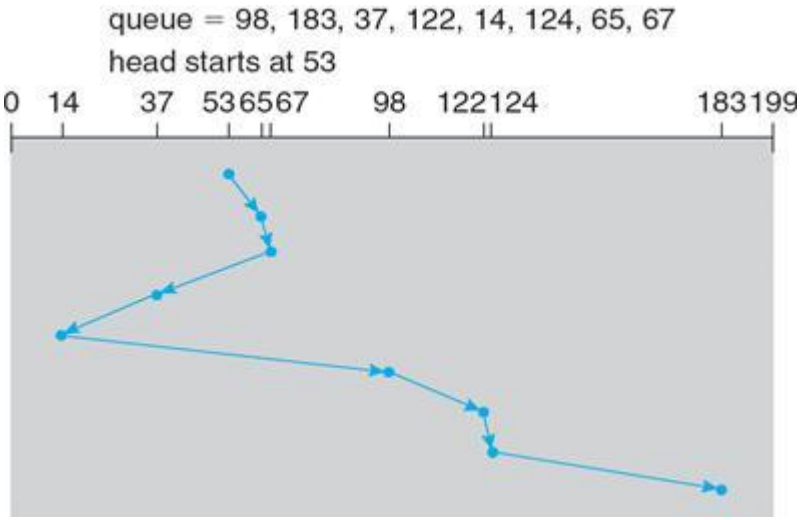
The operating system is responsible for using hardware efficiently. For the disk drives, this means having a fast access time & disk bandwidth.
Access time has two major components: Seek time is the time for the disk to move the heads to the cylinder containing the desired sector. Rotational latency time waiting for the disk to rotate the desired sector to the disk head
Disk bandwidth is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer.

Several algorithms exist to schedule the servicing of disk I/O requests.
We illustrate them with a Request Queue (cylinder range 0-199): 98, 183, 37, 122, 14, 124, 65, 67. Head pointer: cylinder 53

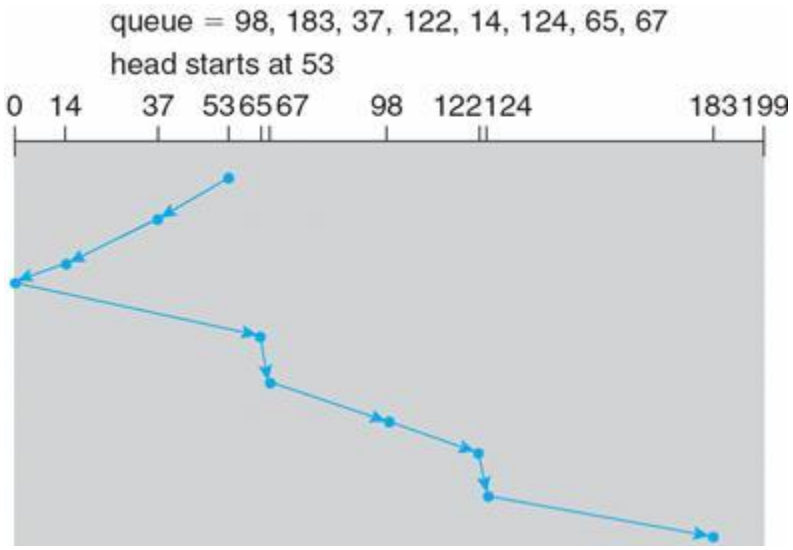
FCFS: Illustration shows total head movement of 640 cylinders



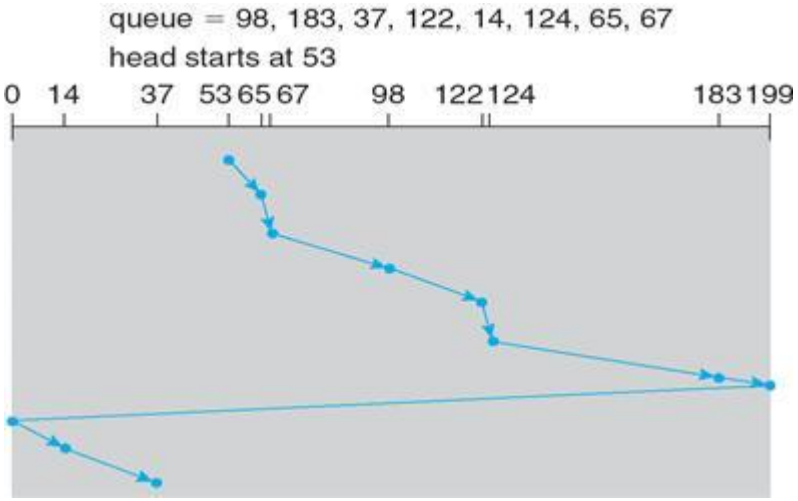
SSTF: Selects the request with the minimum seek time from the current head position. SSTF scheduling may cause starvation of some requests. Illustration shows total head movement off 236 cylinders



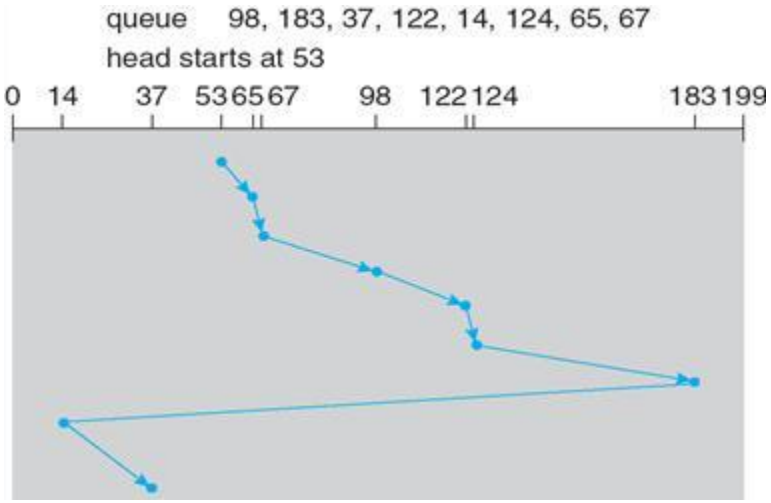
SCAN : The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. SCAN algorithm sometimes called the elevator algorithm. Illustration shows total head movement of 208 cylinders



C-SCAN: Provides a more uniform wait time than SCAN. The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. Treats the cylinders as a Circular list that wraps around from the last cylinder to the first one



C-LOOK: Version of C-SCAN. Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



4.16 Disk structure

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

4.17 Disk Attachment

Disks may be attached one of two ways: 1. Host attached via an I/O port 2. Network attached via a network connection.

4.18 Disk Management

Low-level formatting, or physical formatting: Dividing a disk into sectors that the disk controller can read and write. n To use a disk to hold files, the operating system still needs to record its own data structures on the disk. F Partition the disk into one or more groups of cylinders. F Logical formatting or "making a file system". n Boot block initializes system. F The bootstrap is stored in ROM. F Bootstrap loader program. n Methods such as sector sparing used to handle bad blocks.

4.19 Swap Space Management

Swap-space :Virtual memory uses disk space as an extension of main memory. n Swap-space can be carved out of the normal file system,or, more commonly, it can be in a separate disk partition. n Swap-space management F 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment. F Kernel uses swap maps to track swap-space use. F Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

4.20 Dynamic Memory Allocation

Dynamic memory allocation is when an executing program requests that the operating system give it a block of main memory. The program then uses this memory for some purpose. Usually the purpose is to add a node to a data structure. In object oriented languages, dynamic memory allocation is used to get the memory for a new object.

The memory comes from above the static part of the data segment. Programs may request memory and may also return previously dynamically allocated memory. Memory may be returned whenever it is no longer needed. Memory can be returned in any order without any relation to the order in which it was allocated. The heap may develop "holes" where previously allocated memory has been returned between blocks of memory still in use.

A new dynamic request for memory might return a range of addresses out of one of the holes. But it might not use up all the hole, so further dynamic requests might be satisfied out of the original hole.

If too many small holes develop, memory is wasted because the total memory used by the holes may be large, but the holes cannot be used to satisfy dynamic requests. This situation is called **memory fragmentation**. Keeping track of allocated and deallocated memory is complicated. A modern operating system does all this.

4.21 Library Functions

A library function is linked to the user program and executes in user space while a system call is not linked to a user program and executes in kernel space. A library function execution time is counted in user level time while a system call execution time is counted as a part of system time.

MODULE-V: DEADLOCKS, PROTECTION

System model: Deadlock characterization, methods of handling deadlocks, deadlock prevention, dead lock avoidance, dead lock detection and recovery form deadlock system protection, goals of protection, principles of protection, domain of protection, access matrix, implementation of access matrix, access control, revocation of access rights, capability based systems, language based protection.

At the end of the unit students are able to:		
Course Outcomes		**Knowledge Level (Bloom's Taxonomy)**
CO4	Construct the derivations , FIRST set , FOLLOW set on the context free grammar for performing the top-down and bottom up parsing methods.	Apply
CO5	Distinguish top down and bottom up parsing methods for developing parser with the parse tree representation of the input.	Analyze
CO6	Construct LEX and YACC tools for developing a scanner and a parser.	Apply

Process concepts:

5.1 System Model

For the purposes of deadlock discussion, a system can be modelled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs. Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc. By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "color inkjet printers".
Some categories may have a single resource. In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:

Request : If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ), malloc( ), new( ), and request( ).

Use : The process uses the resource, e.g. prints to the printer or reads from the file.
Release : The process relinquishes the resource. so that it becomes available for other processes.

For example, close( ), free( ), delete( ), and release( ).

For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or wait( ) and signal( ) calls, (i.e. binary or counting semaphores.) A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress.)

5.2 Deadlock Characterization

Mutual exclusion: only one process at a time can use a resource
Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes
No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task
Circular wait: there exists a set {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and P0 is waiting for a resource that is held by P0.

5.2 Methods of handling deadlocks

There are three ways of handling deadlocks:

Deadlock prevention or avoidance : Do not allow the system to get into a deadlocked state.

Deadlock detection and recovery : Abort a process or prempt some resources when deadlocks are detected.

Ignore the problem all together : If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or prempting resources, neither of which is an attractive alternative. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

5.3 Deadlock Prevention

Deadlocks can be prevented by preventing at least one of the four required conditions.

Mutual Exclusion:

Shared resources such as read-only files do not lead to deadlocks. Unfortunately some resources, such as printers and tape drives, require exclusive access by a single process.

Hold and Wait:

To prevent this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others. There are several possibilities for this:

- Require that all processes request all resources at one time. This can be wasteful of system resources if a process needs one resource early in its execution and doesn't need some other resource until much later.
- Require that processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request. This can be a problem if a process has partially completed an operation using a resource and then fails to get it re-allocated after releasing it.
- Either of the methods described above can lead to starvation if a process requires one or more popular resources.

No Preemption:

Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.

- One approach is that if a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, (preempted), forcing this process to re-acquire the old resources along with the new resources in a single request, similar to the previous discussion.
- Another approach is that when a resource is requested and not available, then the system looks to see what other processes currently have those resources *and* are themselves blocked waiting for some other resource. If such a process is found, then some of their resources may get preempted and added to the list of resources for which the process is waiting.
- Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.

Circular Wait:

One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order. In other words, in order to request resource Rj, a process must first release all Ri such that i >= j. One big challenge in this scheme is determining the relative ordering of the different resources

5.4 Dead lock avoidance

The general idea behind deadlock avoidance is to prevent deadlocks from ever happening, by preventing at least one of the aforementioned conditions. This requires more information about each process, AND tends to lead to low device utilization. (I.e. it is a conservative approach.)In some algorithms the scheduler only needs to know the *maximum* number of each resource that a process might potentially use. In more complex algorithms the scheduler can also take advantage of the *schedule* of exactly what resources may be needed in what order. When a scheduler sees that starting a process or granting resource requests may lead to future deadlocks, then that process is just not started or the request is not granted. A resource allocation **state** is defined by the number of available and allocated resources, and the maximum requirements of all processes in the system.
Safe State:
A state is **safe** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state. More formally, a state is safe if there exists a **safe sequence** of processes { P0, P1, P2, ..., PN } such that all of the resource requests for Pi can be granted using the resources currently allocated to Pi and all processes Pj where j < i. (I.e. if all the processes prior to Pi finish and free up their resources, then Pi will be able to finish also, using the resources that they have freed up.) If a safe sequence does not exist, then the system is in an unsafe state, which **MAY** lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks.)

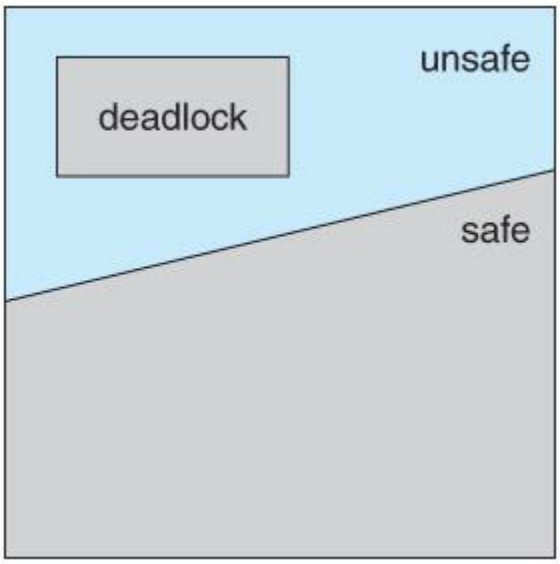


Figure 5.4a : Safe, unsafe, and deadlocked state spaces.

Resource-Allocation Graph Algorithm:

- If resource categories have only single instances of their resources, then deadlock states can be detected by cycles in the resource-allocation graphs.
- In this case, unsafe states can be recognized and avoided by augmenting the resource-allocation graph with ***claim edges***, noted by dashed lines, which point from a process to a resource that it may request in the future.

- In order for this technique to work, all claim edges must be added to the graph for any particular process before that process is allowed to request any resources. (Alternatively, processes may only make requests for resources for which they have already established claim edges, and claim edges cannot be added to any process that is currently holding resources.)
- When a process makes a request, the claim edge Pi->Rj is converted to a request edge. Similarly when a resource is released, the assignment reverts back to a claim edge.
- This approach works by denying requests that would produce cycles in the resource-allocation graph, taking claim edges into effect.
- Consider for example what happens when process P2 requests resource R2:

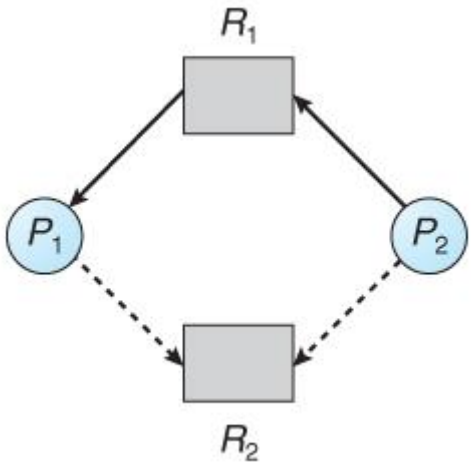


Figure 5.4b : Resource allocation graph for deadlock avoidance

- The resulting resource-allocation graph would have a cycle in it, and so the request cannot be granted.

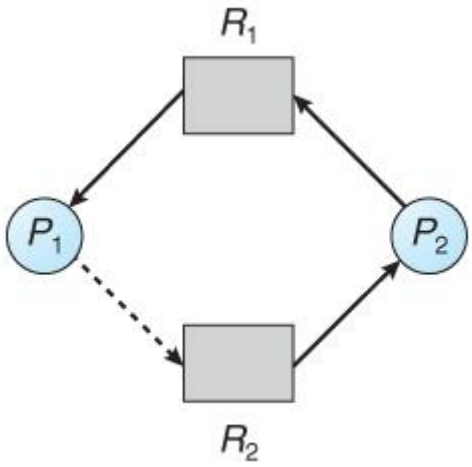


Figure 5.4c : An unsafe state in a resource allocation graph

Banker's Algorithm:

- For resource categories that contain more than one instance the resource-allocation graph method does not work, and more complex (and less efficient) methods must be chosen.
- The Banker's Algorithm gets its name because it is a method that bankers could use to assure that when they lend out resources they will still be able to satisfy all their clients. (A banker won't loan out a little money to start building a house unless they are assured that they will later be able to loan out the rest of the money to finish the house.)
- When a process starts up, it must state in advance the maximum allocation of resources it may request, up to the amount available on the system.
- When a request is made, the scheduler determines whether granting the request would leave the system in a safe state. If not, then the process must wait until the request can be granted safely.
- The banker's algorithm relies on several key data structures: (where n is the number of processes and m is the number of resource categories.)
    - Available[m] indicates how many resources are currently available of each type.
    - Max[n][m] indicates the maximum demand of each process of each resource.
    - Allocation[n][m] indicates the number of each resource category allocated to each process.
    - Need[n][m] indicates the remaining resources needed of each type for each process. (Note that Need[i][j] = Max[i][j] - Allocation[i][j] for all i, j.)
- For simplification of discussions, we make the following notations / observations:
    - One row of the Need vector, Need[i], can be treated as a vector corresponding to the needs of process i, and similarly for Allocation and Max.
    - A vector X is considered to be <= a vector Y if X[i] <= Y[i] for all i.

Safety Algorithm:

- In order to apply the Banker's algorithm, we first need an algorithm for determining whether or not a particular state is safe.
- This algorithm determines if the current state of a system is safe, according to the following steps:
    1. Let Work and Finish be vectors of length m and n respectively.
        - Work is a working copy of the available resources, which will be modified during the analysis.
        - Finish is a vector of booleans indicating whether a particular process can finish. (or has finished so far in the analysis.)
        - Initialize Work to Available, and Finish to false for all elements.
    2. Find an i such that both (A) Finish[i] == false, and (B) Need[i] < Work. This process has not finished, but could with the given available working set. If no such i exists, go to step 4.
    3. Set Work = Work + Allocation[i], and set Finish[i] to true. This corresponds to process i finishing up and releasing its resources back into the work pool. Then loop back to step 2.
    4. If finish[i] == true for all i, then the state is a safe state, because a safe sequence has been found.

Resource-Request Algorithm (The Bankers Algorithm):

- Now that we have a tool for determining if a particular state is safe or not, we are now ready to look at the Banker's algorithm itself.
- This algorithm determines if a new request is safe, and grants it only if it is safe to do so.
- When a request is made (that does not exceed currently available resources), pretend it has been granted, and then see if the resulting state is a safe one. If so, grant the request, and if not, deny the request, as follows:
    1. Let Request[n][m] indicate the number of resources of each type currently requested by processes. If Request[i] > Need[i] for any process i, raise an error condition.
    2. If Request[i] > Available for any process i, then that process must wait for resources to become available. Otherwise the process can continue to step 3.
    3. Check to see if the request can be granted safely, by pretending it has been granted and then seeing if the resulting state is safe. If so, grant the request, and if not, then the process must wait until its request can be granted safely.The procedure for granting a request (or pretending to for testing purposes) is:
        - Available = Available - Request
        - Allocation = Allocation + Request
        - Need = Need – Request

An Illustrative Example

- Consider the following situation:

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- And now consider what happens if process P1 requests 1 instance of A and 2 instances of C. (Request[1] = (1, 0, 2))

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

5.5 Dead lock detection and recovery form deadlock system protection

If deadlocks are not avoided, then another approach is to detect when they have occurred and recover somehow. In addition to the performance hit of constantly checking for deadlocks, a policy algorithm must be in place for recovering from deadlocks, and there is potential for lost work when processes must be aborted or have their resources preempted.

Single Instance of Each Resource Type

- If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph**.**
- A wait-for graph can be constructed from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
- An arc from Pi to Pj in a wait-for graph indicates that process Pi is waiting for a resource that process Pj is currently holding.

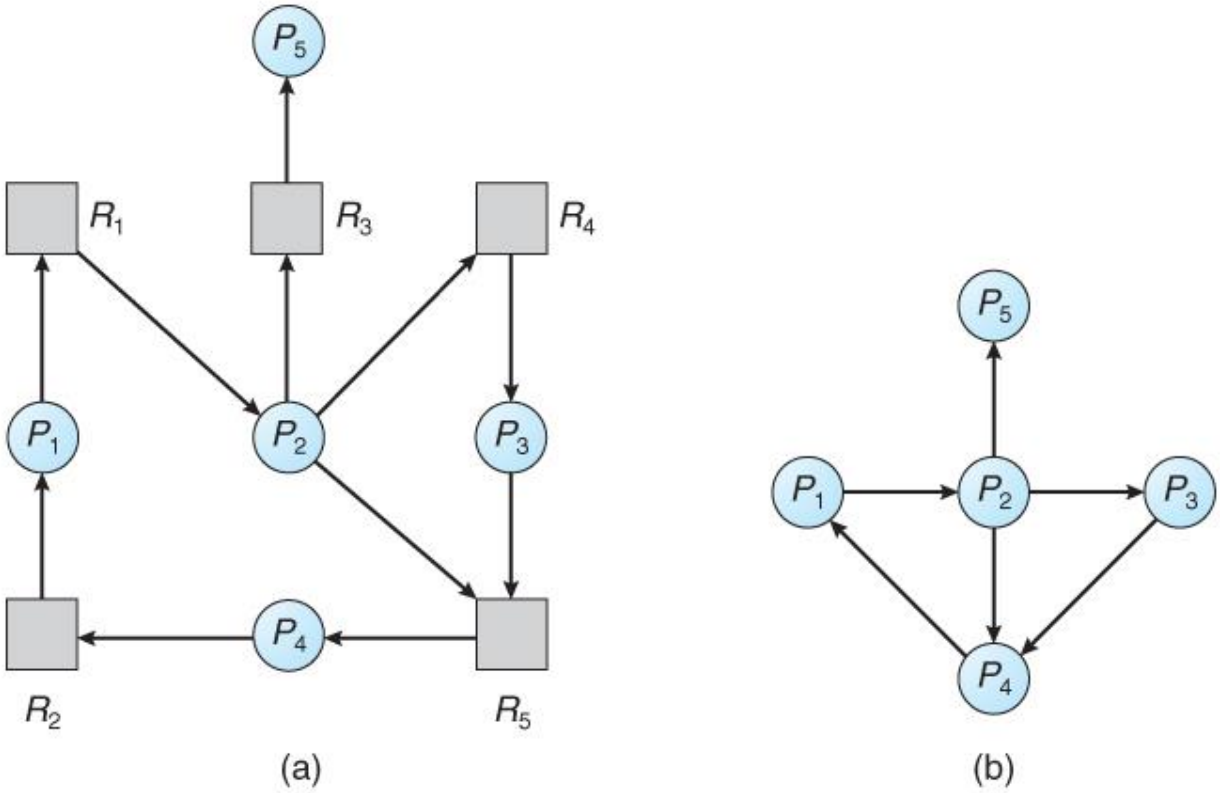


Figure 5.5 - (a) Resource allocation graph. (b) Corresponding wait-for graph

- As before, cycles in the wait-for graph indicate deadlocks.
- This algorithm must maintain the wait-for graph, and periodically search it for cycles.

Several Instances of a Resource Type

- The detection algorithm outlined here is essentially the same as the Banker's algorithm, with two subtle differences:
    - In step 1, the Banker's Algorithm sets Finish[i] to false for all i. The algorithm presented here sets Finish[i] to false only if Allocation[i] is not zero. If the currently allocated resources for this process are zero, the algorithm sets Finish[i] to true. This is essentially assuming that IF all of the other processes can finish, then this process can finish also. Furthermore, this algorithm is specifically looking for which processes are involved in a deadlock situation, and a process that does not have any resources allocated cannot be involved in a deadlock, and so can be removed from any further consideration.
    - Steps 2 and 3 are unchanged
    - In step 4, the basic Banker's Algorithm says that if Finish[i] == true for all i, that there is no deadlock. This algorithm is more specific, by stating that if Finish[i] == false for any process Pi, then that process is specifically involved in the deadlock which has been detected.

- (Note: An alternative method was presented above, in which Finish held integers instead of booleans. This vector would be initialized to all zeros, and then filled with increasing integers as processes are detected which can finish. If any processes are left at zero when the algorithm completes, then there is a deadlock, and if not, then the integers in finish describe a safe sequence. To modify this algorithm to match this section of the text, processes with allocation = zero could be filled in with N, N - 1, N - 2, etc. in step 1, and any processes left with Finish = 0 in step 4 are the deadlocked processes.)
- Consider, for example, the following state, and determine if it is currently deadlocked:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

  - Now suppose that process P2 makes a request for an additional instance of type C, yielding the state shown below. Is the system now deadlocked?

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Detection-Algorithm Usage:

- When should the deadlock detection be done? Frequently, or infrequently?
- The answer may depend on how frequently deadlocks are expected to occur, as well as the possible consequences of not catching them immediately. (If deadlocks are not removed immediately when they occur, then more and more processes can "back up" behind the deadlock, making the eventual task of unblocking the system more difficult and possibly damaging to more processes.)

- There are two obvious approaches, each with trade-offs:
    1. Do deadlock detection after every resource allocation which cannot be immediately granted. This has the advantage of detecting the deadlock right away, while the minimum number of processes are involved in the deadlock. (One might consider that the process whose request triggered the deadlock condition is the "cause" of the deadlock, but realistically all of the processes in the cycle are equally responsible for the resulting deadlock.) The down side of this approach is the extensive overhead and performance hit caused by checking for deadlocks so frequently.
    2. Do deadlock detection only when there is some clue that a deadlock may have occurred, such as when CPU utilization reduces to 40% or some other magic number. The advantage is that deadlock detection is done much less frequently, but the down side is that it becomes impossible to detect the processes involved in the original deadlock, and so deadlock recovery can be more complicated and damaging to more processes.
    3. (As I write this, a third alternative comes to mind: Keep a historical log of resource allocations, since that last known time of no deadlocks. Do deadlock checks periodically (once an hour or when CPU usage is low?), and then use the historical log to trace through and determine when the deadlock occurred and what processes caused the initial deadlock. Unfortunately I'm not certain that breaking the original deadlock would then free up the resulting log jam.)

5.6 Goals of Protection

Obviously to prevent malicious misuse of the system by users or programs. See chapter 15 for a more thorough coverage of this goal. To ensure that each shared resource is used only in accordance with system *policies,* which may be set either by system designers or by system administrators. To ensure that errant programs cause the minimal amount of damage possible. Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

5.7 Principles of protection

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

5.8 Domain of Protection

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

Domain Structure:

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

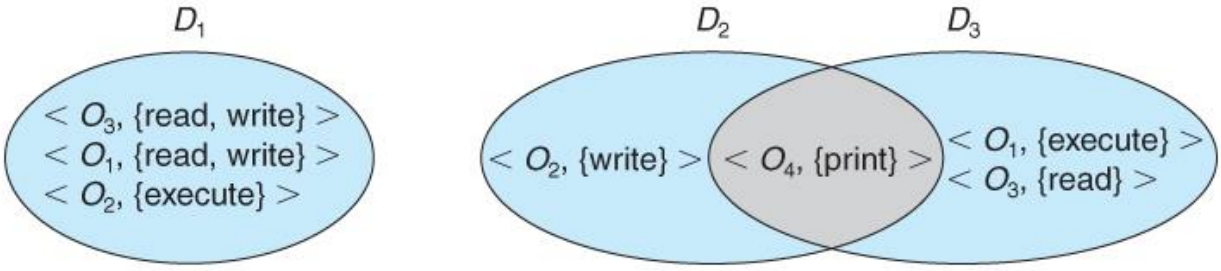


Figure 5.8 : System with three protection domains.

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

5.9 Access Matrix

The model of protection that we have been discussing can be viewed as an access matrix**,** in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure 5.9a : Access matrix.

Domain switching can be easily supported under this model, simply by providing "switch" access to other domains:

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure 5.9b : Access matrix of Figure 5.9a with domains as objects.

The ability to **copy** rights is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e. for the same object. There are two important variations:

- If the asterisk is removed from the original access right, then the right is **transferred,** rather than being copied. This may be termed a **transfer** right as opposed to a **copy** right.
- If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propagated further. That is the new domain does not also receive the right to copy the access.

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Figure 5.9c : Access matrix with *copy* rights.

- The owner right adds the privilege of adding new rights or removing existing ones:

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Figure 5.9d : Access matrix with *owner* rights.

- Copy and owner rights only allow the modification of rights within a column. The addition of ***control rights***, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains. For example in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Figure 14.7 - Modified access matrix of Figure 14.4

5.10 Implementation of Access Matrix

Global Table:

The simplest approach is one big global table with < domain, object, rights > entries. Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques.) There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

Access Lists for Objects:

Each column of the table can be kept as a list of the access rights for that particular object, discarding blank entries. For efficiency a separate list of default access rights can also be kept, and checked first.

Capability Lists for Domains:

In a similar fashion, each row of the table can be kept as a list of the capabilities of that domain. Capability lists are associated with each domain, but not directly accessible by the domain or any user process.

A Lock-Key Mechanism:

Each resource has a list of unique bit patterns, termed locks. Each domain has its own list of unique bit patterns, termed keys. Access is granted if one of the domain's keys fits one of the resource's locks. Again, a process is not allowed to modify its own keys.

Comparison:

Each of the methods here has certain advantages or disadvantages, depending on the particular situation and task at hand. Many systems employ some combination of the listed methods.

5.11 Revocation of Access Rights:

- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.
- With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:

- Reacquisition - Capabilities are periodically revoked from each domain, which must then re-acquire them.
- Back-pointers - A list of pointers is maintained from each object to each capability which is held for that object.
- Indirection - Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.
- Keys - A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
    - A master key is associated with each object.
    - When a capability is created, its key is set to the object's master key.
    - As long as the capability's key matches the object's key, then the capabilities remain valid.
    - The object master key can be changed with the set-key command, thereby invalidating all current capabilities.
    - More flexibility can be added to this scheme by implementing a ***list*** of keys for each object, possibly in a global table.

5.11 Capability-Based Systems

An Example: Hydra

- Hydra is a capability-based system that includes both system-defined rights and user-defined rights. The interpretation of user-defined rights is up to the specific user programs, but the OS provides support for protecting access to those rights, whatever they may be
- Operations on objects are defined procedurally, and those procedures are themselves protected objects, accessed indirectly through capabilities.
- The names of user-defined procedures must be identified to the protection system if it is to deal with user-defined rights.
- When an object is created, the names of operations defined on that object become auxiliary rights, described in a capability for an instance of the type. For a process to act on an object, the capabilities it holds for that object must contain the name of the operation being invoked. This allows access to be controlled on an instance-by-instance and process-by-process basis.
- Hydra also allows rights amplification**,** in which a process is deemed to be trustworthy, and thereby allowed to act on any object corresponding to its parameters.
- Programmers can make direct use of the Hydra protection system, using suitable libraries which are documented in appropriate reference manuals.

5.13Language-Based Protection

As systems have developed, protection systems have become more powerful, and also more specific and specialized. To refine protection even further requires putting protection capabilities into the hands of individual programmers, so that protection policies can be implemented on the application level, i.e. to protect resources in ways that are known to the specific applications but not to the more general operating system.