GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

OPERATING SYSTEMS

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

PEO1 : Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2 : Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3 : Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON OPERATING SYSTEMS

Class & Sem :II B.Tech. – II Semester Year : 2018-19

Branch : CSE Credits : 3

==========

1. Brief History and Scope of the Subject

- Computer operating systems (OS) provide a set of functions needed and used by most application programs on a computer, and the links needed to control and synchronize computer hardware. On the first computers, with no operating system, every program needed the full hardware specification to run correctly and perform standard tasks, and its own drivers for peripheral devices like printers and punched paper card readers.
- Operating systems are the software that makes the hardware usable. Hardware provides "raw computing power." Operating system makes the computing power conveniently available to users, by managing the hardware carefully to achieve good performance.
- Operating systems can also be considered to be managers of the resources. An operating system determines which computer resources will be utilized for solving which problem and the order in which they will be used. In general, an operating system has three principal types of functions.
- Allocation and assignment of system resources such as input/output devices, software, central processing unit, etc.
- Scheduling: This function coordinates resources and jobs and follows certain given priority.
- Monitoring: This function monitors and keeps track of the activities in the computer system. It maintains logs of job operation, notifies end-users or computer operators of any abnormal terminations or error conditions. This function also contains security monitoring features such as any authorized attempt to access the system as well as ensures that all the security safeguards are in place .
- Throughout the history of computers, the operating system has continually evolved as the needs of the users and the capabilities of the computer systems have changed.

2. Pre-Requisites

Basic knowledge of system programs and application programs

3. Course Objectives:

- To impart the concepts of process, memory and file management techniques.
- To familiarize with the deadlock handling techniques.

4. Course Outcomes:

Upon successful completion of the course, the students will be able to

- describe the role, functions and structures of operating systems.
- evaluate the performance of CPU scheduling algorithms by calculating average waiting time and turnaround time.
- compare and contrast memory management schemes for efficient utilization of memory.
- apply deadlock prevention, avoidance and recovery techniques to keep the system in safe state.
- determine seek time of disk scheduling algorithms.
- develop software or hardware based solutions for critical section problems.
- analyze files and directory structures and implementations.

5. Program Outcomes:

Graduates of the Computer Science and Engineering Program will have Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and wit society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

6. Mapping of Course Outcomes with Program Outcomes:

	1	2	3	4	5	6	7	8	9	10	11	12
CO1	M	M										M
CO2	M	M										M

CO3	M	M										M
CO4	M	M										M
CO5	M	M										M
CO6	M	M										M
CO7	M	M										M

7. Prescribed Text Books

i. Abraham Silberschatz, Peter B, Galvin, Greg Gagne, Operating System Principles, John Wiley, 7th edition.

ii. Stallings, Operating Systems - Internal and Design Principles, Pearson education, 6th edition–2005.

8. Reference Text Books

i. D. M. Dhamdhere, Operating systems- A Concept based Approach, TMH, 2nd edition.

ii. Andrew S Tanenbaum, Modern Operating Systems, PHI, 3rd edition.

9. URLs and Other E-Learning Resources

http://www.nptel.iitm.ac.in/video.php?subjectId=112106134

http://www.preservearticles.com/2012051832397/5-important-limitation-of-operations-research.html

http://www.nptel.iitm.ac.in/video.php?subjectId=112106134

http://personal.maths.surrey.ac.uk/st/J.F/chapter7.pdf

http://nptel.iitm.ac.in/syllabus/syllabus_pdf/111107064.pdf

http://nptel.iitm.ac.in/courses/110106045/

http://nptel.iitm.ac.in/syllabus/109103021/

http://www.me.utexas.edu/~jensen/ORMM/supplements/units/game_theory/game_thry.pdf

http://nptel.iitm.ac.in/video.php?subjectId=112106131

10. Digital Learning Materials:

http://www.scribd.com/doc/39223153/Replacement-Models-Operation-Research#download

http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IIT-ROORKEE/INDUSTRIAL-ENGINERRING/part3/inventory/lecture2.htm

http://www.eolss.net/sample-chapters/c02/E6-05-05-05.pdf

11. Lecture Schedule / Lesson Plan(3+1*)

Topic	No. of Periods	
	Theory	Tutorial
UNIT- 1: INRODUCTION		
Operating system operations	**1**	**2**
Operating system services	**2**	
System calls	**1**	
Types of system calls	**2**	
Operating –system structure	**2**	
UNIT-II: Process Management		
Process, Process state, Process control block (PCB)	**1**	**3**
Process scheduling	**1**	
Scheduling queues	**1**	
Schedulers	**1**	
Context switch	**1**	
Scheduling criteria	**1**	
Scheduling algorithms	**3**	
Operations on processes	**2**	
Inter process communication	**2**	
UNIT – III: Memory Management Strategies		
Swapping	**1**	**2**
Contiguous memory allocation	**1**	
Paging	**3**	
Segmentation	**1**	
Virtual-Memory Management	**1**	
Demand paging	**1**	
Page replacement Algorithms	**2**	
Allocation of Frames	**1**	
Thrashing	**1**	

UNIT - IV : Deadlocks and Mass-storage structure		
System model, Deadlock characterization	**1**	**2**
Methods for handling deadlocks:	**1**	
deadlock- prevention, Avoidance	**1**	
Detection, recovery	**1**	
Mass-storage structure:	**1**	
Overview, Disk Scheduling	**2**	
Disk Management	**2**	
UNIT - V: Synchronization		
The critical section problem	**1**	**2**
Peterson's solution	**1**	
Synchronization hardware	**1**	
Semaphores	**2**	
Classic problems of synchronization	**2**	
Monitors	**2**	
UNIT-VI: File system Interface		
Concept of a file	**1**	**2**
Access methods	**1**	
Directory structure	**1**	
File system mounting	**1**	
Files sharing and protection	**1**	
Total No. of periods	**56**	**13**

12. Seminar Topics

CPU Scheduling
Deadlocks
Disk Scheduling

OPERATING SYSTEMS

Unit – 1

Introduction

Objectives:

- To introduce the basic concepts and functions of various operating systems

Syllabus: Introduction

Operating system operations, Operating system services, System calls, Types of system calls, Operating –system structure.

Outcomes:

Students will be able to

- Understand the structure of Operating System
- Know the Services provided by Operating System
- Identify various System calls

LEARNING MATERIAL
UNIT-I

Definitions:

- An Operating System is a program that acts as an intermediary between a user of a computer and the computer hardware.
- An operating system is software that manages the computer hardware.
- The operating system controls the hardware and coordinates its use among the various application programs for the various users.
- An operating system is similar to a ***government***. Like a government, it performs no useful function by itself. It simply provides an *environment* within which other programs can do useful work.
- An operating system is a control program. A manages the execution of user programs to prevent errors and improper use of the computer.
- An operating system is a control program. A manages the execution of user programs to prevent errors and improper use of the computer.
- It is a set of programs previously written and stored in the memory of the computer.

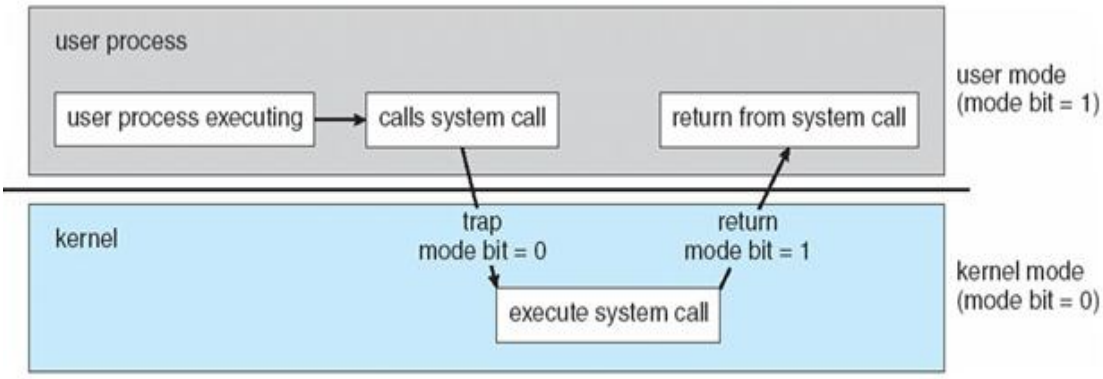
1.1 OPERATING SYSTEM OPERATIONS

- Modern operating systems are **interrupt driven.**
- If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system is waiting for something to happen.
- Events are almost always signaled by the occurrence of an interrupt or a trap.
- A **trap (or** an **exception)** is a software-generated interrupt caused either by an error or by a specific request from a user program.

- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- If the operating system and the users share the hardware and software resources of the computer system.
    - With sharing, many processes could be adversely affected by a bug in one program.
    - For example, if a process gets stuck in an infinite loop, this loop could prevent the correct operation of many other processes.
    - One erroneous program might modify another program, the data of another program, or even the operating system itself.
- A properly designed operating system must ensure that an incorrect program cannot cause other programs to execute incorrectly.

Dual-Mode Operation:

- To ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.
- There are two **modes** of operation:
    - **User mode** and
    - **Kernel mode (supervisor mode, system mode, or privileged mode).**
- A bit, called the **mode bit,** is added to the hardware of the computer to indicate the current mode: **kernel (0) or user (1)**.
- When the computer system is executing on behalf of a user application, the system is in user mode.
- When a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.



- At system boot time, the hardware starts in **kernel mode**.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode
- Whenever the operating system gains control of the computer, it is in kernel mode.
- The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
- We accomplish this protection by designating some of the machine instructions that may cause harm as **privileged instructions.**
- The hardware allows privileged instructions to be executed only in kernel mode.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.

Timer:

- We must prevent a user program from getting stuck in an infinite loop or not calling system services and never returning control to the operating system.
- To accomplish this goal, we can use a timer.

- A timer can be set to interrupt the computer after a specified period.
- There are two types of timers
  - **Fixed timer**
  - **Variable timer**
- A **variable timer** is generally implemented by a fixed-rate clock and a counter.
- The operating system sets the counter.
  - Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs
- Before turning over control to the user, the operating system ensures that the timer is set to interrupt.
- If the timer interrupts, control transfers automatically to the operating system.

1.2 OPERATING SYSTEM SERVICES

- An Operating System provides an environment for the execution of programs.
- It provides certain services to programs and to the users of those programs.
- The specific services provided are differ from one operating system to another.
- One set of operating system services provides functions that are helpful to the user.

1. **User Interface**
   - Almost all operating systems have a user interface. This interface can take several forms
     - **Command Line Interface:** Command Line Interface which uses text commands and a method for entering them.

- **Batch Interface:** Batch Interface in which commands and directives to control those commands are entered into files, and those are executed.
- **Graphical User Interface:** This interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

2. **Program execution**
   - The system must be able to load a program into memory and to run that program.
   - The program must be able to end its execution, either normally or abnormally.
3. **I/O operation**
   - A running program may require I/O, which may involve file or an I/O device.
   - For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.
4. **File System manipulation**
   - The file System is of particular interest obviously, programs need to read and write files and directories.
   - They also need to create and delete them by name, search for a given file, and list file information.
   - Some program includes permissions management to allow or deny access to files or directories based on file ownership.
5. **Communication**
   - There are many circumstances in which one process needs to exchange information with another process.

- Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.
- Communication may be implemented via shared memory or through message passing

6. **Error Detection:**
   - The operating system needs to be constantly aware of possible errors.
   - Error may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices.
   - Debugging facilities can greatly enhance the user's and programmer's abilities to use the system efficiently.
7. **Resource Allocation**
   - When there are multiple users or multiple jobs running at a same time, resources must be allocated to each of them.
   - Many different types of resources are managed by the operating system.
   - Some may have special allocation code, whereas others may have much more general request and release code.
8. **Accounting**
   - We want to keep track of which users use how much and what kinds of computer resources.
   - This record keeping may be used for accounting so that user billed or simply for accumulating usage statistics.
9. **Protection and security**
   - The owners of information stored in a multiuser or networked computer system may want to control use of that information.

- When several separate processes execute concurrently, it should not be possible for one process to interface with the others or with the operating system itself.
- Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important.

1.3 SYSTEM CALLS

- System calls provide an interface to the services made available by an operating system.
- These system calls are generally available as routines written in C and C++.
- Certain low level tasks are written in assembly-language instructions.
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.
- Three most common APIs are
  - Win32 API for Windows
  - POSIX* API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

Below is a sequence of system calls to copy the contents of one file to another file:

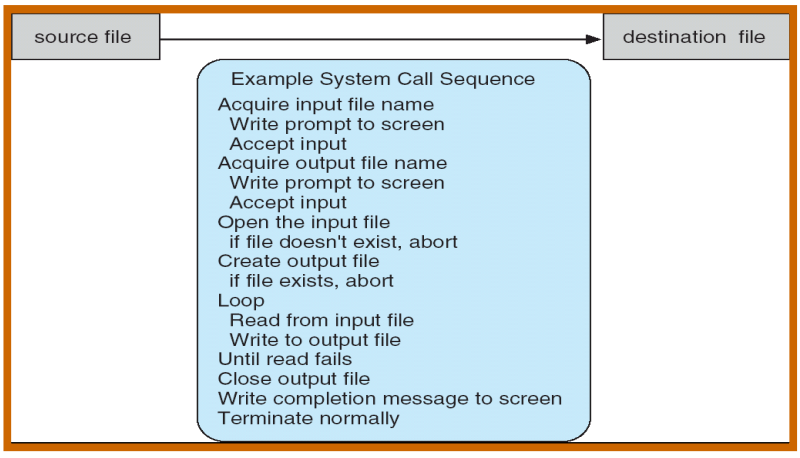


Fig: **Example of a System Call**

- Some system calls are generally in assembly language. The assembler converts assembly language to machine language and thus system calls are executed.
- Typically, there is a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

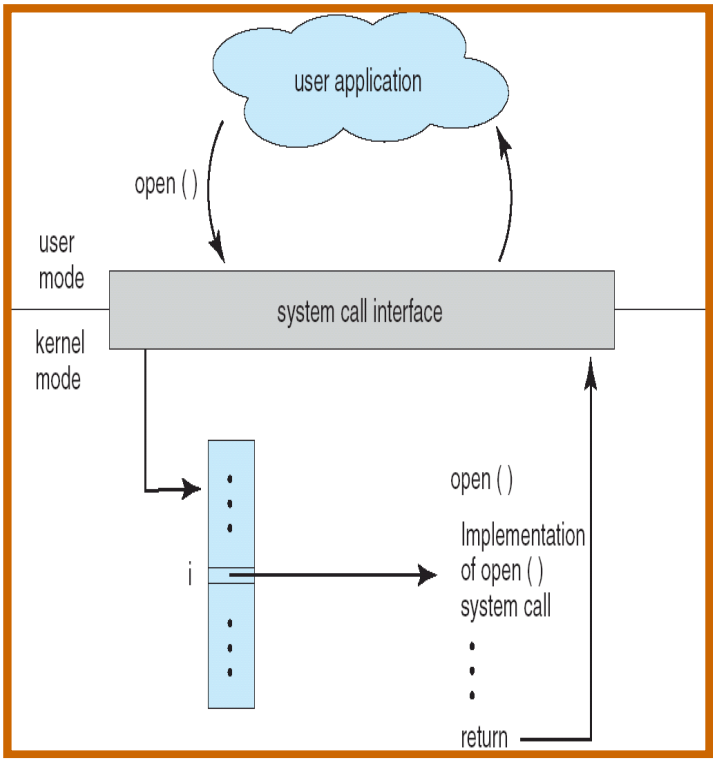


Fig: **The handling of a user application invoking the open() system call**

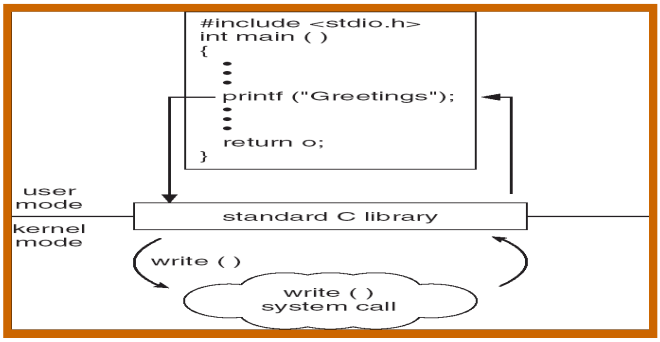


Fig: **C program invoking printf() library call, which calls the write() system call**

System Call Parameter Passing:

- Three general methods used to pass parameters to the OS
  1. Simplest: pass the parameters in *registers*
     - In some cases, may be more parameters than registers
  2. Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
     - This approach taken by Linux and Solaris
  3. Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

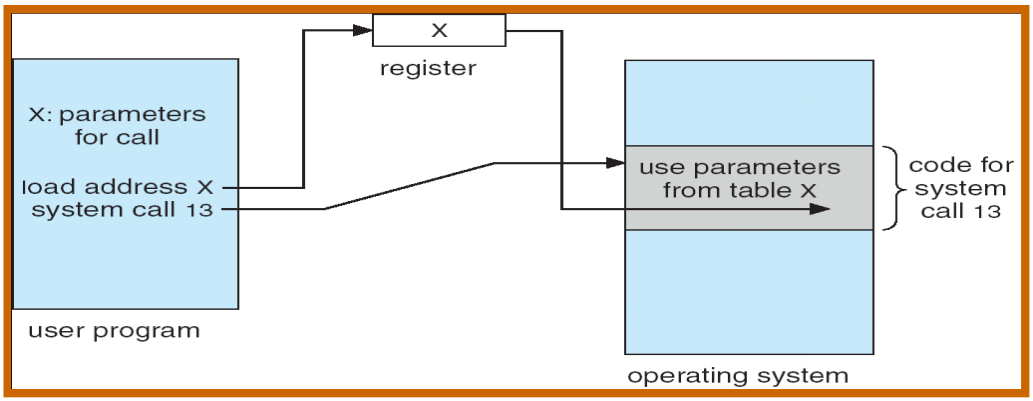


Fig: **Passing of parameters as a table**

1.4 Types of System Calls

- System calls can be grouped into five major categories:
  1. Process control
     - Load, execute, end, abort, create process, get/set process attributes, wait for time/signal, allocate/free memory
  2. File management (manipulation)
     - Create/delete/open/close/read/write a file, get/set file attributes
  3. Device management
     - Request/release device, read/write data, get/set attributes
  4. Information maintenance
     - Get/set time or date, get/set system data, get/set attributes for process/file/device
  5. Communications
     - Create/delete connection, send/receive messages, attach/detach devices

1. Process control:

- **end, abort:** The process end is used normally for the process to be end and abort is used when the errors occur in the process.
- **load, execute:** To bring the job from secondary memory to main memory, the system call load is used and to execute particular job ,execute system call is used.
- **create process, terminate process:** To create a process we use a system call called create process, after the complete execution of the parent process the system call terminate process is used.
- **wait event, signal event:** Wait event makes the event wait for a while until a particular event done, and then the system call, signal event is used. Until the signal is not given, the operation is not done. Ex: Signal Clock

- **allocate and free memory:** Allocates the resources whatever the process needs, when it is created and after its execution making the memory free.

2. File Management:

- **Create, delete:** Create means creating a file. Delete means deleting a file. These are the system calls given by the user, and the system process the work.
- **Open, close:** Open means opening a file. Close means closing a file.
- **Read, write, reposition:** read means reading the contents of a file. Write means writing the contents to a file. Reposition means changing the position i.e., moving information from one drive to another.
- **get file attributes & set file attributes:** get file attributes means the details of the file i.e., when it is created, last modified, user name, type of file, date of creation etc., set file attributes means changing the current attributes.

3. Device Management:

- **Request device, release device:** The request generated to do particular action, if the devices already engage and set them to release.
- **Read, write, reposition:** The system calls for read, write and reposition of a device.
- **get device attributes & set device attributes:** It is used for getting and setting the attributes **.**Ex: Settings in phone

4. Information Maintenance:

- **get time/date , set time/date**
- **get system date , set system date**
- **get process ,file ,device attributes**
- **set process ,file ,device attributes**

5. Communication:

- **Create, delete:** For creating and deleting communication connection. We are having sharing on and off options between two systems.
- **Send, receive:** The send and receive system calls are used to send and receive messages based on the requirement we need.

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

1.5 OPERATING SYSTEM STRUCTURE

- A System as large and complex as a modern operating system must be engineered carefully if it is to be function properly and to be modified easily.
- The common approach is to partition the task into small components.
- Each of these modules should have inputs, outputs and functions.

There are four types of operating systems structures.

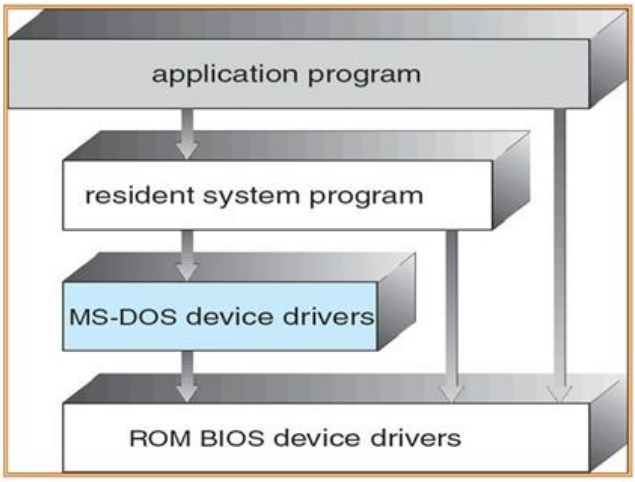
1. **Simple Structure**
2. **Layered approach**
3. **Micro kernels**
4. **Modules**

1. Simple Structure:

- Many commercial operating systems do not have well-defined structures.
- Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope.

Example 1: MS-DOS operating system.

- It was written to provide the most functionality in the least space, so it was not divided into modules carefully.
- In MS-DOS, the interfaces and levels of functionality are not well separated.
- There is no CPU Execution Mode (user and kernel), and so errors in applications can cause the whole system to crash.

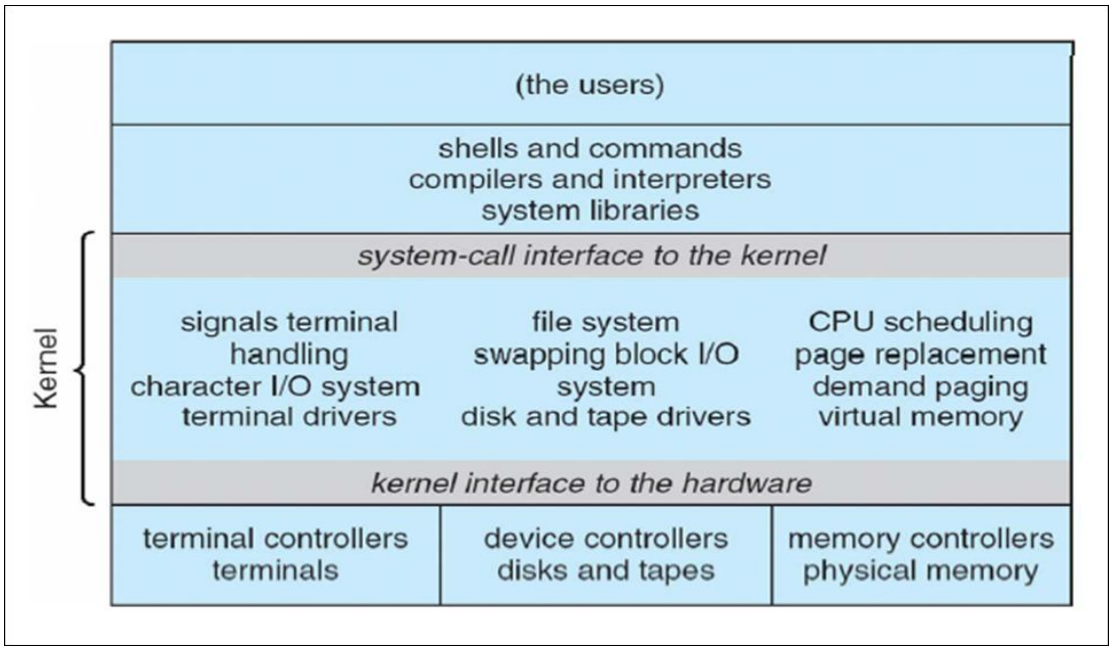


MS-DOS layer structure

Example 2: The original UNIX operating system.

- Like MS-DOS, UNIX initially was limited by hard ware functionality.

- It consists of **two separable parts**: The kernel and the system programs.
- The kernel:
  - It is further divided in to a series of interfaces and device drivers which have been added and expanded over the years as UNIX has evolved.
  - Everything below the system call interface and above the physical hardware is the kernel.
  - The Kernel provides the file system, CPU scheduling, Memory management and other operating system functions through system calls.
  - This monolithic structure was difficult to implement and maintain.



Traditional UNIX Operating System

2. Layered Approach:

- A system can be made modular in many ways.
- One method is layered approach.

- Here, the operating system is broken into a number of layers or levels.
- The bottom layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.
- A layer of an operating system say layer M consists of data structures and a set of routines that can be invoked by higher level layers.
- Layer M in turn, can invoke operations on lower level layers.

Advantages:

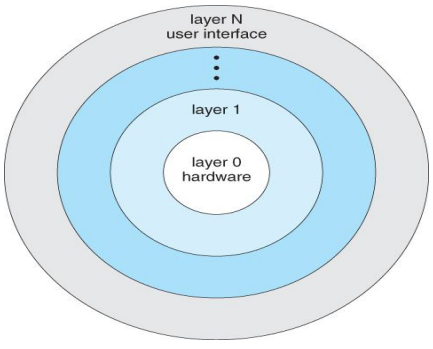
1. Simplicity of construction and debugging
   - The layers are selected so that each uses functions and services of only lower-level layers.
   - This approach simplifies debugging and system verification.
   - The first layer can be debugged without any concern for the rest of the system, because it uses only the basic hardware to implement its functions.
   - Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on.
   - If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.
   - Thus, the design and implementation of the system is simplified.
   - Each layer is implemented with only those operations provided by lower-level layers.
   - A layer does not need to know how these operations are implemented; it needs to know only what these operations do.
   - Each layer hides the existence of certain data structures, operations, and hardware from higher-lever layers.

Disadvantages:

- The major difficulty with this approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.
- **Example**: The device driver for the backing store must be at a lower level than the memory-management routines. Because, memory management requires the ability to use the backing store.
- Final problem with layered implementations is that they tend to be less efficient than other types.

Example:

- When a user program executes an I/O operation, It executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.
- At each layer, the parameters may be modified; data may need to be passed, and so on.
- Each layer adds overhead to the system call; the net result is a system call that takes longer than does one on a non layered system.



Layered Operating system

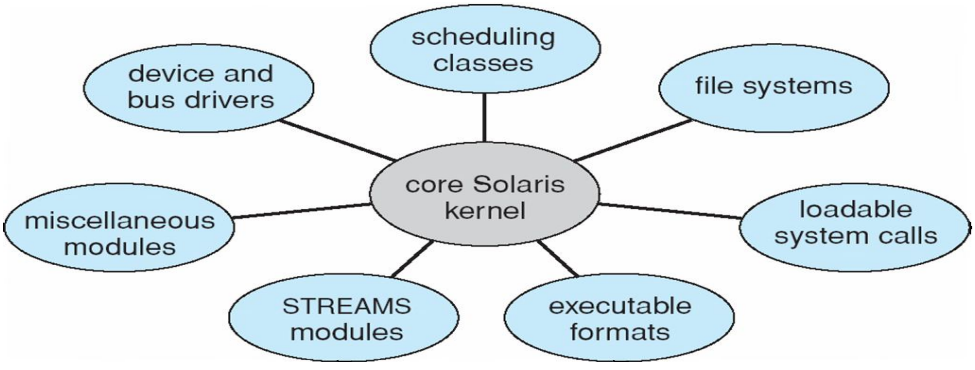
3. Micro kernels:

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the Microkernel approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- The result is a smaller kernel.
- There is a little consensus that which services should remain in the kernel and which should be implemented in user space.
- The main function of the microkernel is to provide a **communication facility**.
- This communication facility is provided between the client program and the various services that are also running in user space.
- Communication is provided by message passing.
- Example: If the client program wishes to access a file, it must interact with file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- **Advantages** :
    - One benefit of the microkernel approach is ease of extending the operating system.
    - All new services are added to user space and consequently do not require modification of the kernel.
    - The microkernel also provides more security and reliability.
- Some operating systems that have used micro kernel approach: Tru64UNIX, QNX.

- **Drawback:**
  - Micro kernels can suffer from performance decreases due to increased system function overhead.

4. Modules:

- The best current methodology for operating system design involves using object-oriented programming techniques to create a modular kernel.
- The kernel has a set of core components and dynamically links in additional services either during boot time or run time.
- Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX such as Solaris, Linux and Mac OS X.
- Example 1: Solaris operating system structure:
  It is organized around a core kernel with seven types of loadable kernel modules.
  1. Scheduling classes.
  2. File systems.
  3. Loadable system calls.
  4. Executable formats.
  5. STREAMS modules.
  6. Miscellaneous.
  7. Device and Bus drivers.



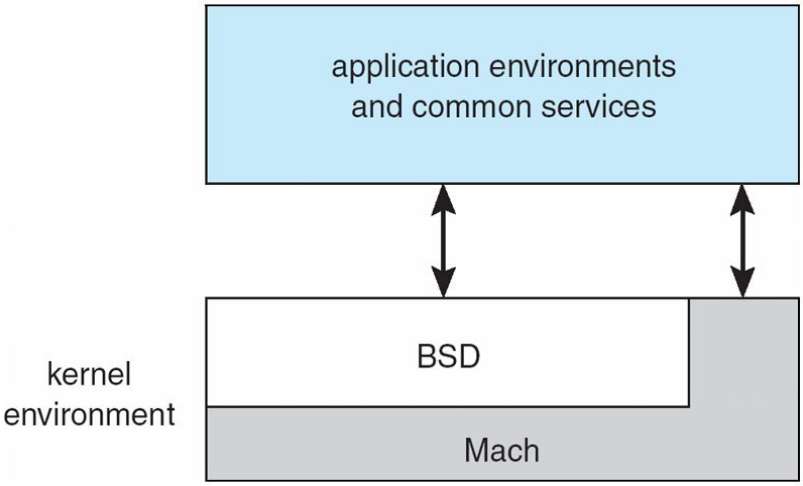
Solaris loadable module

Advantages:

1. It is more flexible than layered system in that any module can call any other module.
2. It is more efficient, because modules do not need to invoke message passing in order to communicate.

Example 2: Mac OS X Structure

- The Apple Macintosh Mac OS X operating system uses a hybrid structure.
- Mac OS X structures the operating system using a layered technique where one layer consists of the Mach microkernel.
- The top layer includes application environments and a set of services providing a graphical interface to applications.
- Below these layers is the kernel environment, which consists primarily of the Mach microkernel and the BSD kernel.
- Mach provides memory management:
    - Support for remote procedure calls.
    - Inter process communication.
    - Message passing.
    - Thread scheduling.
- BSD component provides
    - BSD command line interface
    - Support for networking and file systems.
    - Implementation of POSIX APIs, including Pthreads.
- In addition to Mach and BSD, the kernel environment provides an I/O kit for development of device drivers and dynamically loadable modules.


application environments
and common services
BSD
kernel
environment
Mach

UNIT-I
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. An _____ acts as an interface between the user and the computer system.
2. Which concept explains the working of an Operating System? [ ]
   a) It is event driven
   b) It is object oriented
   c) It is procedure based system software
   d) It is a collection of procedures that interact with each other
3. A kernel is an essential part of an operating system [True/False]
4. Which of these is/are the desirable features of an Operating system
   a) Extensible b) Portable c) Reliable d) All [ ]
5. Which one of the following is the mode bit associated for user mode and kernel mode respectively [ ]
   a) 1 and 0 b) 0 and 1 b)1 and 2 d) 2 and 1
6. CPU has two modes: privileged and non-privileged. In order to change the mode from privileged to non-privileged **(GATE-2001)**
   a) a hardware interrupt is needed. [ ]
   b) a software interrupt is needed.
   c) a privileged instruction (which does not generate an interrupt) is needed.
   d) a non-privileged instruction (which does not generate an interrupt) is needed.
7. _____ is a mechanism which involves in ensuring that all access to system resources is controlled.
8. Some of the important activities that an Operating System performs [ ]
   a) Job accounting b) Security

c) Error detecting aids d) All of these

9. Which of the following system calls are used to maintain system information.
   a) Get/set time or date. [ ]
   b) request device, release device.
   c) send, receive messages.
   d) get process attributes, set process attributes.

10. _____ provides an interface to the services made available by an operating system.

11. In which of the following users do not interact with the computer directly [ ]
   a) Batch operating system b) DOS operating system
   c) Time-sharing Operating Systems d) None of these

12. Which of the following functionality is provided by micro kernel approach
   a) Communication. [ ]
   b) ease of extending of an operating system.
   c) reliability and security.
   d) All of the above.

13. _____ and _____ are two fundamental models of implementing communication.

14. One function of an operating system is to handle interrupts. Interrupts are [ ]
   a) a delay in processing due to operating system overload
   b) signals from hardware or software requesting attention from the operating system
   c) messages received from other computers
   d) None of the above.

14.System calls are invoked by using (**NPTEL/GATE1999**)
   a) software interrupt b) polling [ ]
   c) indirect jump d) a privileged instruction

SECTION-B

SUBJECTIVE QUESTIONS

1. Define operating system. Explain the **operations** of an operating system?
2. With a neat sketch explain the **Dual-mode** operation?
3. Explain the need of attaching **timer** in operating system?
4. With a neat sketch explain the **structure** of operating system.
5. With a neat sketch explain the structure of traditional **UNIX** operating system?
6. Describe the **services** that an operating-system provides to users?
7. List and explain different types of **system calls**?
8. What are the advantages and disadvantages of **layered approach**?
9. Write pros and cons of **micro kernels**?
10. Explain the need of **module structure** in operating system?

Unit – 2

Process Management

Objectives:

- Students will be able to develop the concepts of process management techniques

Syllabus:

Process, process state, process controls block (PCB),

Process scheduling- scheduling queues, schedulers, context switch, scheduling criteria, scheduling algorithms, Operations on processes, Inter process communication.

Outcomes:

Students will be able to

- Learn the Process, Process state diagram and various fields in process control block.(PCB)
- Explain the terms scheduling queues, schedulers, context switch and scheduling criteria.
- Differentiate Preemptive and non preemptive scheduling algorithms.
- Explain various techniques used for inter process communication.
- Know the benefits of multi-threading and models.

Learning Material

Process:

- A program in execution is called a process.
- A process will need certain resources such as CPU time, memory, files and I/O devices to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.
- An operating system executes a variety of programs: Batch system jobs, Time shared systems, user programs or tasks
- Generally the terms ***job*** and ***process*** are almost the same.
- Process execution must progress in sequential fashion

A process includes:

- program counter
- stack
- data section

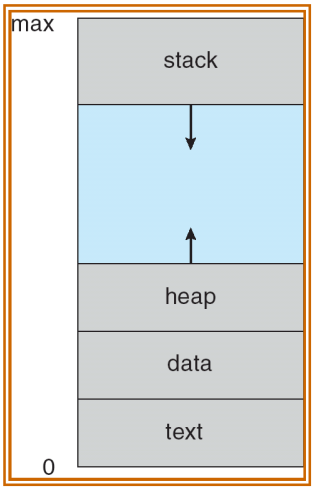


Fig: Process in memory

Process:

- A process is a program under execution.
- It generally includes process stack, containing temporary data (such as sub routine parameters, return addresses, temporary variables) and data section containing global variables.
- A **program** is a **passive entity** such as the contents of files stored on a disk.
- A P**rocess** is an **active entity** with a program counter, specifying the next instruction to execute and set of associated resources.

Process State:

- As a process executes, it changes state.
- The state of process is defined in part by the current activity of that process.
- Each process may be in one of the following states.

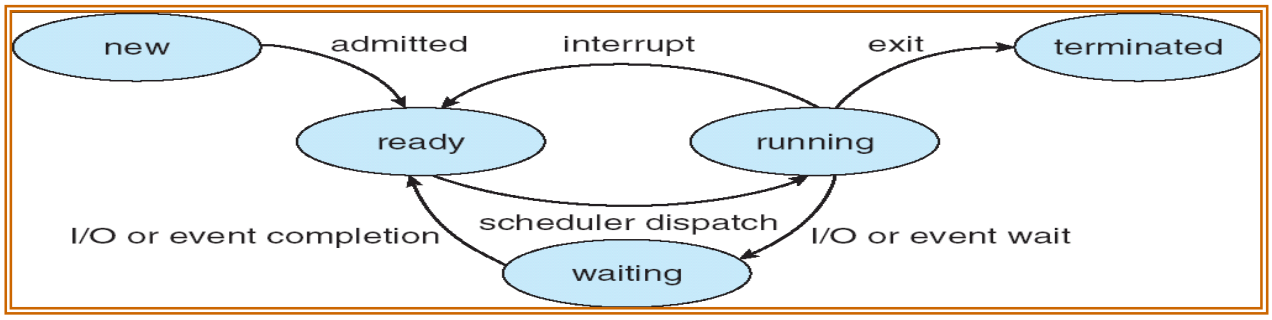


Fig: Process State Diagram

New: The process is being created.

Running: Instructions are being executed.

Waiting: The process is waiting for some event to occur (such as I/O completion or reception of a signal).

Ready: The process is waiting to be assigned to the processor.

Terminated: The process has finished execution.

- At any point of time only one process can be running. Many processes may be ready and waiting.

Process Control Block:

- Each process is represented in the operating system by a process control block also called task control block.

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

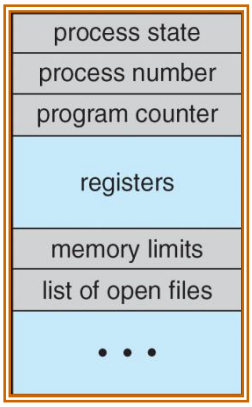


Fig: Process Control Block (PCB)

It contains many pieces of information associated with a specific process, including these:

1. **Process State:** The state may be new, ready, running, and waiting, halted and so on...

2. **Program Counter:** The counter indicates the address of next instruction to be executed for this process.
3. **CPU Registers:** The registers vary in number and type, depending on computer architecture. They include accumulators, index registers, stack pointers and general purpose register plus any condition code information.
4. **CPU Scheduling Information:** The information includes a process priority, pointers to scheduling queues and any other scheduling parameters
5. **Memory Management Information :** This include information such as the values of the base and limit registers, the page tables or segment tables depending on the memory system used by operating system
6. **Accounting Information:** This **includes** the amount of CPU and real time used, time limits, account numbers, job or process numbers and so on...
7. **I/O Status Information:** This includes the list of I/O devices such as tape drivers allocated to this process, list of open files and so on....

The simply serve as the repository for any information that may vary from process to process.

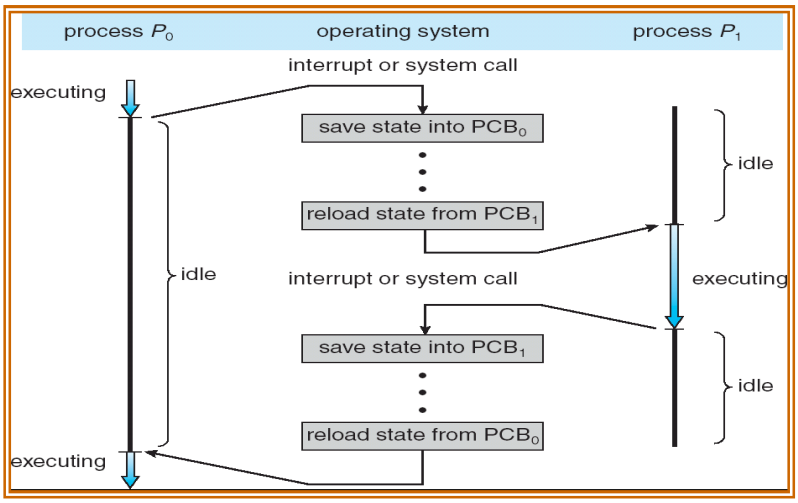


Fig: CPU switches from process to process

Process Scheduling:

- The objective of multiprogramming is to have some process running at all times.
- To maximize CPU utilization for any processor system, there will never be more than one running process.
- If there are more process the rest will have to wait until the CPU is free and can be rescheduled.

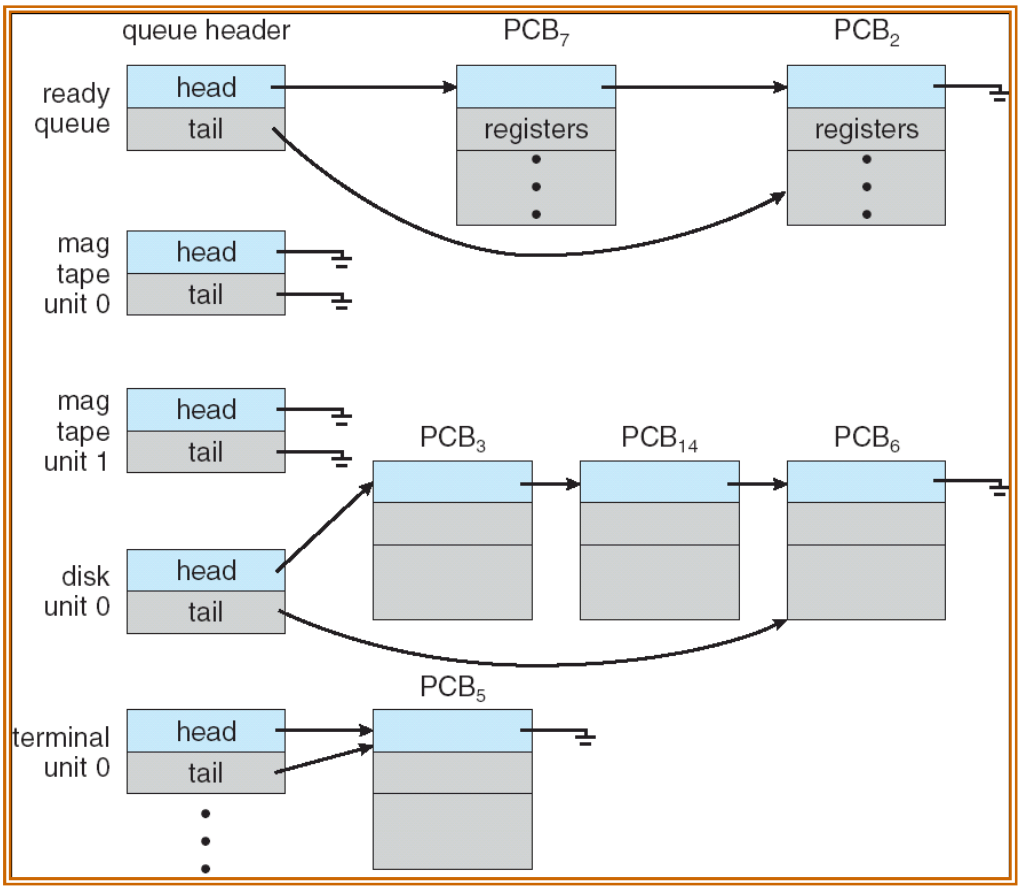


Fig: The ready queue and various I/O device queues

a) Scheduling Queues:

1. **Job Queue:** As processes enter the system they are put into a job queue this queue consists of all process in the system
2. **Ready queue:** The processes that are residing in main memory and are ready and waiting to execute are kept a list called the ready queue. This

queue is generally stored as a linked list. A ready queue header will contain pointers to the first and last PCB's in the list. Each PCB has a pointer field that points to the next process in the ready queue.

3. **Device Queue:** A list of process waiting for a particular I/O device is called a device queue.

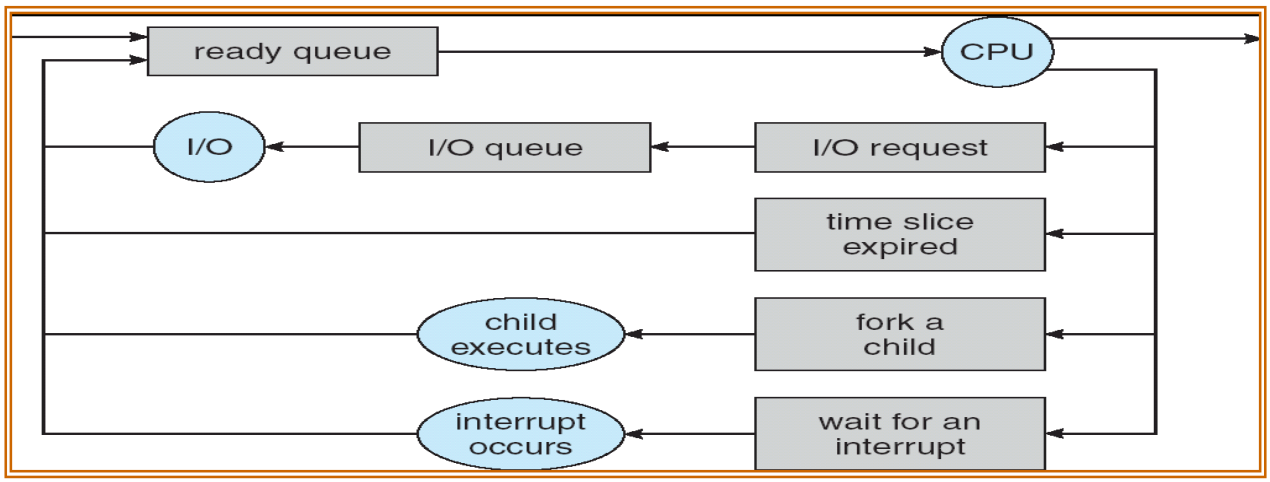


Fig: Queuing diagram representation of process scheduling.

- In queuing diagram each rectangular box represents a queue.
- The circles represent the resources that serve the queues
- The arrows indicate the flow of processes in the system.

b) Schedulers:

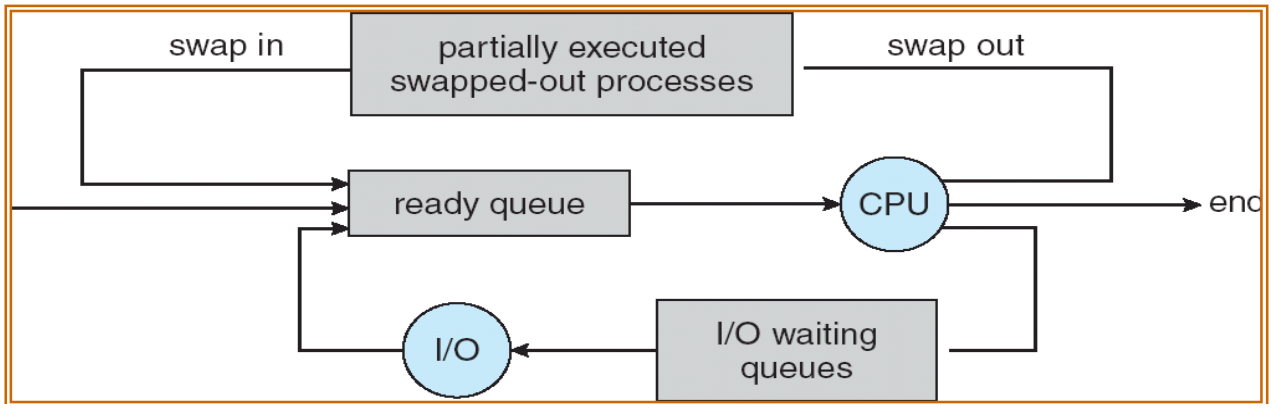


Fig: Addition of medium-term scheduling to the queuing diagram

1. Long-term Scheduler:

- It is also known as job scheduler.
- It selects processes from this pool and loads them into memory for execution.
- It executes much less frequently; minutes may separate the creation of one new process and the next.
- It controls the degree of multiprogramming.

2. Short-term Scheduler:

- It is also known as CPU scheduler.
- It selects from among the processes that are ready to execute and allocates the CPU to one of them.
- It must select a new process for the CPU frequently.
- It executes at least once every 100ms.Because of the short time between executions; the short-term scheduler must be fast. If it takes 10ms to decide to execute a process for 100ms, then 10/(100+10)=9% of the CPU is being used(Wasted) simply for the scheduling work.

3. Medium-term Scheduler:

- The medium-term scheduler temporarily removes processes from main memory and places them in secondary memory and vice versa.
- This is referred as "swapping out" or "swapping in" / "paging out" or "paging in".
- The medium-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is page faulting frequently, or a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is

available, or when the process has been unblocked and is no longer waiting for a resource.

Processes can be described as either:

- ***I/O-bound process*** – spends more time doing I/O than computations, many short CPU bursts
- ***CPU-bound process*** – spends more time doing computations; few very long CPU bursts

c) Context Switch:

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- This is called Context Switch.
- Context-switch time is overhead; the system does no useful work while switching Time dependent on hardware support.
- When a context switch occur kernel saves the context of old process in its PCB and loads the context of new process that is scheduled to run.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  - Switches from running to waiting state
  - Switches from running to ready state
  - Switches from waiting to ready
  - Terminates
- Scheduling under 1 and 4 is *no preemptive*
- All other scheduling is *preemptive*

Scheduling Criteria:

- Different CPU Scheduling algorithms have different properties.
- The criteria include the following.

- **CPU utilization**: To keep the CPU as busy as possible. it range from 0 to 100 percent. In a real system it should range from 40%(for lightly loaded system) to 90% (for heavily loaded system)
- **Throughput**: The number of processes that complete their execution per time unit
- **Turnaround time**: The interval from the time of submission of process to the time of completion is the turnaround time. This time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing in the CPU and doing I/O. The amount of time to execute a particular process.
- **Waiting time:** The amount of time that a process has been waiting in the ready queue. It is the sum of the periods spent waiting in the ready queue.
- **Response time** : amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time and response time.

Scheduling Algorithms :

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.
- There are many different CPU scheduling algorithms are there:

1. **FCFS(First –Come, First-Served)**
2. **SJF(Shortest-Job-First)**
3. **Priority Scheduling**
4. **Round-Robin(RR)**

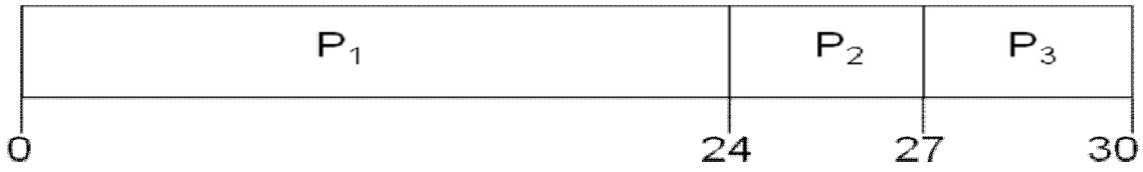
1. FCFS (First –Come, First-Served):

- It is the simplest CPU scheduling algorithm, the process that requests the CPU first is allocated the CPU first.
- The implementation of FCFS is easily managed with FIFO queue.
- When a process enters the ready queue its PCB is linked on to the tail of the queue.
- When CPU is free, it is allocated to the process at the head of the queue.
- FCFS is non preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.

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

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- The Gantt Chart for the schedule is:

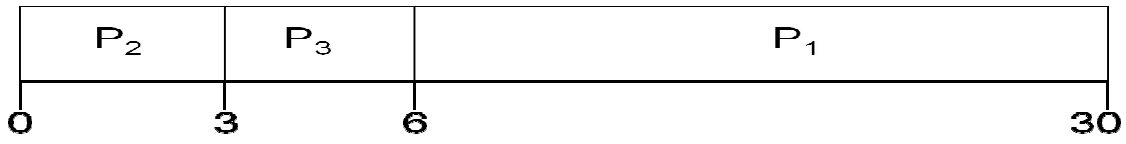


- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- Suppose that the processes arrive in the order
  - P_2 , P_3 , P_1

Convoy Effect:

- All other processes wait for the one big process to get off the CPU. This effect results in lower CPU utilization and device utilization.

The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: (6 + 0 + 3) / 3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

2. SJF (Shortest-Job-First):

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time

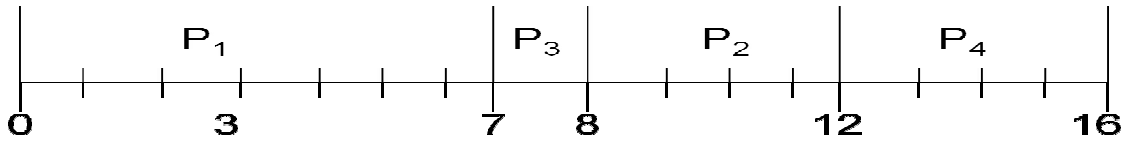
➢ Two schemes:

- Non preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
- Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

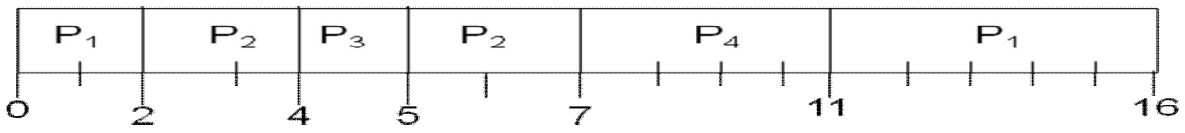


- Average waiting time = (0 + 6 + 3 + 7) / 4 = 4

Example of Preemptive SJF:

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (preemptive)



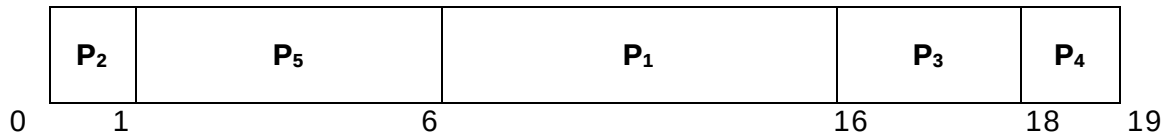
- Average waiting time = (9 + 1 + 0 +2)/4 = 3

3. Priority Scheduling:

- SJF algorithm is a special case of the general priority scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with highest priority.
- Equal-priority processes are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where the priority(p) is the inverse of the (predicted) next CPU burst .
- The large CPU burst, the lower the priority, and vice versa.
  - Priorities are generally indicated by some fixed range of numbers, such as 0 to 7, or 0 to 4095. Here we assume low numbers represent high priority.
  - As an example, consider the following set of processes, assumed to have arrived at time 0, in the order p1, p2,...., p5, with the length of the CPU burst given in milliseconds.

Process	Burst Time	Priority
P_1	**10**	**3**
P_2	**1**	**1**
P_3	**2**	**4**
P_4	**1**	**5**
P_5	**5**	**2**

Using priority scheduling, we would schedule these processes according to the following Gantt chart:



The average waiting time is 8.2 milliseconds.

- Priorities can be defined either internally or externally. Internally defined priorities are influenced by time limits, memory requirements, the number of open files, and the ratio of I/O burst to average CPU burst.
- External priorities are set by criteria outside operating system, such as importance of the process, type and amount of funds being paid for computer use.
- Priority scheduling can be either preemptive or non preemptive.
- When a process arrives at the ready queue, its priority compared with the priority of currently running process.
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of newly arrived process is higher, than the currently running process.
- A non preemptive scheduling algorithm will simply put the new process at the head of the ready queue.

Drawback:

Starvation

- Starvation or indefinite blocking is the major problem in priority scheduling algorithms.
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes indefinitely.

- High priority processes can prevent low- priority processes from ever getting the CPU.

Aging

- Aging is a solution to the problem of indefinite blocking of low – priority processes.
- Aging is a technique of gradually increasing the priority of a process that wait in the system for a long time.
- **For ex:** Priorities range from low to high that is 127 to 0, we could increase the priority of a waiting process by 1 every 15 minutes.

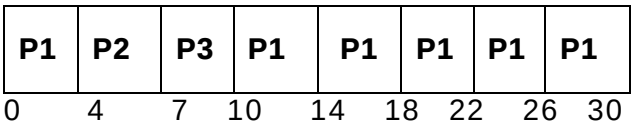
4. Round Robin Scheduling:

- This algorithm is designed especially for time sharing systems.
- It is similar to FCFS scheduling, but pre-emption is added to switch between processes.
- A small unit of time called time quantum or time slice is defined.
- Time quantum is generally from 10 to 100 ms. the ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of upto 1 time quantum.
- The ready queue acts as FIFO queue of processes, new processes are added to tail of ready queue.
- The CPU scheduler pick the first processes from the ready queue The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- In RR scheduling we are having **two possibilities** first one is the process may have CPU burst of less than 1 time quantum.
    - In this case the process itself releases the CPU voluntarily. Scheduler will then proceed to the next process in ready queue.

- Second possibility is CPU burst of currently running process is longer than 1 time quantum, the timer will go off and cause interrupt to the operating system.
- A context switch will be executed, and the process will be executed, and the process will be put any the tail of ready queue.
- The CPU scheduler will then select the next process in the ready queue.
- The average waiting time in RR policy is long.
  - consider the following set of processes that arrive at 0, with the length of the CPU burst given in milliseconds.

Process	Burst Time
P_1	24
P_2	3
P_3	3

  - If we use time quantum of 4 milliseconds, then process p1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is pre-empted after the first time quantum, and CPU is given to the next process in the queue, process p2.
  - Since p2 does not need 4 milliseconds, it quits before its time quantum expires. The CPU is given to process p3.
  - Once each process received 1 time quantum the CPU is returned to process p1 for an additional time quantum



The average waiting time is 17/3=5.66 ms

5. Multilevel Queue Scheduling:

- Processes are easily classified into different groups.
- For example common division is made between foreground (interactive) processes and background (batch) processes.
- These two types of processes have different response time requirements**.**

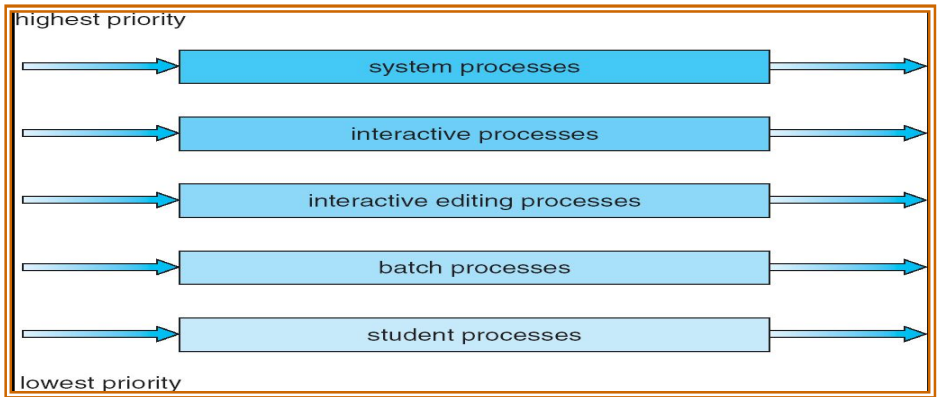


Fig: Multilevel queue scheduling.

- A multi level queue scheduling algorithm partitions ready queue into several separate queues.
- The processes are permanently assigned to one queue, generally based on some property of process, such as memory size, process priority, or process type.
- For example, separate queues might be used for foreground and background processes.
- The foreground queue might be scheduled by an RR- algorithm.
- Example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority.

1. System processes
2. Interactive process
3. Interactive editing processes
4. Batch processes
5. Student processes

- If interactive editing processes entered the ready queue while a batch processes was running, the batch process would be pre-empted.
- The foreground queue can be given can be given 80 percent of CPU time for RR scheduling among its processes, where as the background queue receives 20 percent of CPU time for its processes in FCFS basis.

6. Multilevel Feedback Queue Scheduling:

- This algorithm allows a process to move between queues. The idea is to separate processes according to the CPU bursts.
- If a process uses too much CPU time, it will be moved to lower-priority queue.
- A process that waits too long in a lower priority queue may be moved to a higher priority queue.
- For example, consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2. The scheduler first executes all the process in the queue 0.

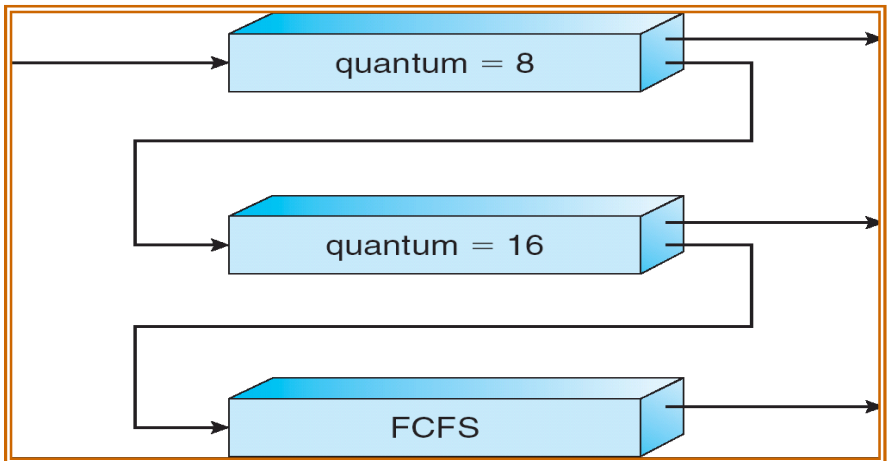


Fig: Multilevel feedback queues.

- Only when queue 0 is empty it executes processes in queue 1.
- Processes in queue 2 will only be executed if queue 0 and 1 are empty.
- A process that arrives for queue 1 will in turn be pre-empted by a process arriving for queue 0.

Operations on Processes

- Process in most systems can execute concurrently and they may be created and deleted dynamically.
  - Process Creation
  - Process Termination

1. Process Creation:

- A process may create several new processes by using Create-process system call, during execution.
- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- In most operating system s a process can be identified by using a unique process identifier (pid), which is typically an integer number.

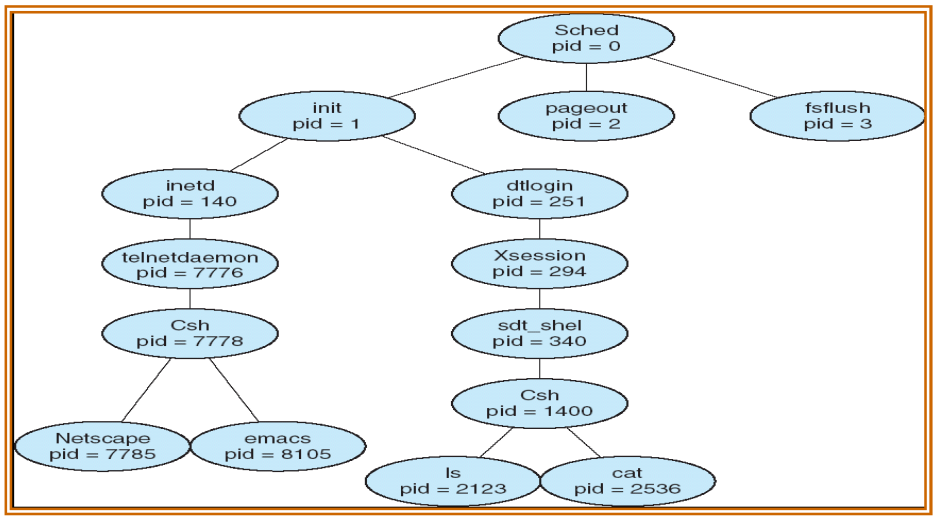


Fig: A tree of processes on a typical Solaris system

When a process creates a new process, two possibilities exist in terms of execution:

- Parent and children execute concurrently
- Parent waits until children terminate

Resource sharing

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

- A new process is created by the fork ( ) system call.
- The new process consists of copy of Address space of the original process. Here parent process easily communicates with its child process.

- Both parent and child process continues execution at the instruction after the fork ( ) with one difference:
    - The return code for fork ( ) is zero for child process whereas the non zero process identifier of the child is returned to parent.

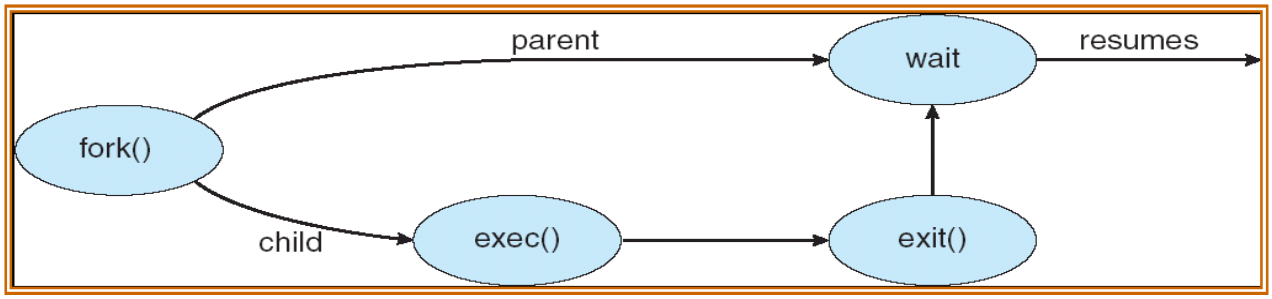


Fig: Process Creation

- The exec ( ) system call is invoked after fork ( ) and then parent will create more children or if it has nothing else to do it waits while the child runs by invoking wait ( ) system call.
- If child completes its execution and terminates then parent also terminates using exit ( ) system call.

2. Process Termination:

- A Process terminates when it finishes executing its last statement and asks the operating system to delete it by using exit ( ) system call.
- All the resources the resources of the process including physical and virtual memory, open files and I/O buffers and deallocated by the operating system.
- Termination can occur in other circumstances as the process can cause termination of another process. Parent process can terminate its child process.

Parent may terminate execution of children processes (abort)

- Child has exceeded allocated resources
- Task assigned to child is no longer required

- If parent is exiting

- Some operating system do not allow child to continue if its parent terminates.
- If a process terminates either normally or abnormally, then all of its children must also be terminated.
- This phenomenon is referred to as - cascading termination

Inter-process Communication:

- Processes executing concurrently in the operating system may be either independent process or cooperating process.
  - **Independent Process:**
    An independent process is a process which cannot affect or be affected by other processes in the system. Any process that does not share data with any other process is independent.
  - **Cooperating Process:**
    It can effect or be affected by other process executing in the system. Any process that shares information with other process is a cooperating process.

There are several reasons behind process cooperation:

1. **Information Sharing:** Several users may be interested in the same piece of information for example a shared file; we must allow concurrent access to such information.
2. **Computation Speed:** If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with others.
3. **Modularity:** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

4. **Convenience:** Even an individual user may work on many tasks at the same time. For instance, a user may be editing, printing and compiling in parallel.

- Cooperating processes require an inter process communication (IPC), which allow them to exchange data and information.

There are two fundamental models in Inter Process Communication:

1. Shared memory systems.

2. Message Passing Systems.

- In shared- memory model, a region of memory that is shared by cooperating processes is established.
- Processes can then exchange information by reading and writing the data to the shared region.
- In message- passing, the communication takes place by means of messages exchanged between cooperating processes.

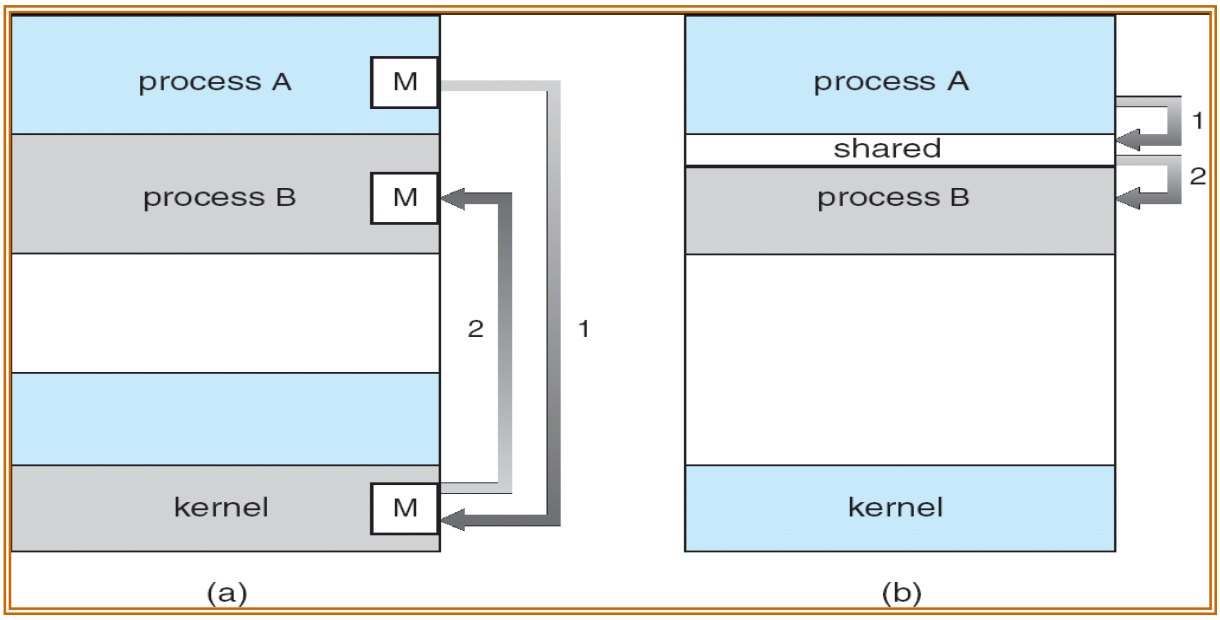


Fig: Communications models (a) Message passing (b) Shared Memory

- Message passing is useful for exchanging smaller amounts of data, it is also easier to implement than shared memory.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.
- Shared memory is faster than message passing systems.

1. Shared memory Systems:

- Inter-process communication using shared memory requires communicating processes to establish a region of shared memory.
- A shared memory region resides in the address space of the process creating the shared memory segment.
- The other processes that wish to communicate using this shared memory segment must attach it to their address space.
- Ex: The concept of cooperating process can be illustrated by Producer-Consumer problem.
- The producer produces the information that is consumed by a consumer process.
- If producer and consumer process are executing concurrently, then the buffer is filled by producer and emptied by consumer.

  Two types of buffers are used:

  Unbounded Buffer: This has no limit on the size of buffer. The consumer may have to wait for new items, but producer can always produce new items.

  Bounded Buffer: Here buffer size is fixed. The consumer must wait, if buffer is empty and producer must wait if buffer is full.

2. Message Passing System:

- This system allows processes to communicate without sharing some address space, here communicating processes may reside on different computers connected by a network.
- A message passing facility provides at least two operations: send (message) and receive (message).
- Messages sent by a process can be either fixed size or variable size.
- ***Fixed size:*** If fixed size messages only sent, the system level implementation is straight forward and programming is more difficult.
- ***Variable size:*** Variable sized messages required a more complex system level implementation, but programming task become simpler.
- If process p and q want to communicate they must send messages to and receive messages from each other, a communication link must exist between them.
- The link may be physical implementation such as shared memory hardware bus, or network or logical implementation.
- Logical implementation of a link include send ( )/ receive ( ) operations.
  1. Naming
  2. Synchronization
  3. Buffering

1. Naming:

- Processes that want to communicate must have a way to refer each other. They use either direct or indirect communication.

Direct Communication

- Each process that wants to communicate must explicitly name the recipient or sender of the communication.

➢ Send (p, message) ---- send a message to process p.

➢ Receive (q, message) --- receive a message from process q.

- Here communication link is established with exactly two processes.
- This scheme exhibits symmetry in addressing;
- Therefore both sender and receiver processes must name the other process. Asymmetric in addressing, here only sender names, the recipient, where recipient is not require to name the sender.

➢ Send (p, message) ---- send a message to process p.

➢ Receive (id, message) --- receive a message from any process.

- The variable id is set to the name of process with which communication has taken place.

Indirect Communication

- Here, the messages are sent to and received from mailboxes or ports.
- A process can communicate with other process can communicate with other process only if the processes has shared mailbox.

➢ Send (A, message) ---- send a message to mailbox A.

➢ Receive (A, message) --- receive a message from mailbox A.

- Here communication link may be associated with more than two processes.
- For example the processes p1, p2 and p3 all shared mailbox A. Process P1 sends message to A. while both P2 and P3 both executes receive() from A.
- The operating system will provide a mechanism that allows a process to do the following:

➢ Create a new mail box

➢ Send and receive messages through the mail box

➢ Delete mail box

2. Synchronization

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

3. Buffering

- Whether a communication is direct or indirect, messages exchanged by communicating processes reside in temporary queue.

These queues are 3 types.

- **Zero capacit**y: Maximum length of queue is zero, here link cannot have any messages waiting in it, therefore the sender must block until recipient receives message.
- **Bounded Capacity**: Queue has finite length n; thus almost n messages can reside in it. If queue is not full when a new message is sent, the message is placed in queue, and sender continues execution without waiting. If link is full the sender must block until space is available.

UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Long term scheduler is also known as_____ [ ]
   A. High scheduler C. CPU scheduler
   B. Job scheduler D. None of the above
2. Short term scheduler is also known as_____ [ ]
   A. High scheduler C. CPU scheduler
   B. Job scheduler D. None of the above
3. Process Control Block is also called as_____ [ ]
   A. Program Control Block C. Project Control Block
   B. Task Control Block D. Procedure Control Block
4. In which of the following scheduling policies does context switching never takes place. [ ]
   A. Round-Robin C. Shortest Job First
   B. Preemptive D. FCFS
5. Suppose that a process is in 'BLOCKED' state waiting for some I/O service. When the service is completed, it goes to the [ ]
   A. Running State C. Ready State
   B. Suspended State D. Terminated State
6. Which of the following scheduling algorithm give minimum average waiting time? [ ]
   A. FCFS B. SJF C. Round-Robin D. Priority
7. Which of the following scheduling policy is well suited for time-shared OS
   A. FCFS B. SJF C. Round-Robin D. Elevator [ ]
8. Shortest Remaining Time First is a preemptive version of ____ [ ]
   A. FCFS B. SJF C. Round-Robin D. Priority

9. Which combination of the following features will suffice to characterize an OS as a multi-programmed OS? **(GATE-2002)**

I. More than one program may be loaded into main memory at the same time for execution.

II. If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

III. If the execution of a program terminates, another program is immediately scheduled for execution. [ ]

A. i B. i and ii C. i and iii D. i, ii and iii

10. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____ [ ]

A. Job queue B. Ready queue C. Device queue D. FIFO queue

11. Which of the following statement(s) is false about SJF? [ ]

S1: It causes minimum average waiting time

S2: It can cause starvation

A. Only S1 B. Only S2 C. Both S1 and S2 D. Neither S1 nor S2

12. Pre-emptive scheduling is the strategy of temporarily suspending a running process [ ]

A. before the CPU time slice expires

B. to allow starving processes to run

C. when it requests I/O

D. to avoid collision

13. What is the range of a time quantum in Round-Robin Scheduling?

A. 10-100 ms C. 10-100 ns

B. 100-1000 ms D. 100-1000ns

14. As a rule of thumb what percentage of the CPU bursts should be shorter than the time quantum? [ ]

A. 80% B. 70% C. 60% D. 50%

15. Interval between the time since submission of the job to the time its results become available, is called [ ]
    A. Response Time C. Throughput
    B. Waiting time D. Turnaround Time
16. The scheduling in which CPU is allocated to the process with least CPU-burst time is called [ ]
    A. Priority Scheduling C. Round Robin Scheduling
    B. Multilevel Queue Scheduling D. Shortest job first Scheduling
17. Which scheduling policy is used for a batch processing operating system
    A. Shortest-job First. C. Round-Robin. [ ]
    B. Priority Based D. First-Come-First-Serve.
18. Which of these is a technique of improving the priority of process waiting in Queue for CPU allocation [ ]
    A. Starvation B. Relocation C. Promotion D. Aging
19. Consider a set of n tasks with known runtimes r1, r2, ... rn to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput? **(GATE-2001)**

    A. Round-Robin C. Shortest-Job-First [ ]
    B. Highest-Response-Ratio-Next D. First-Come-First-Served
20. Which of the following scheduling algorithms is non-preemptive? (**GATE CS 2002**) [ ]
    A. Round Robin
    B. First come first serve
    C. Multilevel Queue Scheduling
    D. Multilevel Queue Scheduling with Feedback

SECTION-B

SUBJECTIVE QUESTIONS

1. With a neat sketch explain **process state diagram**?
2. Explain about the contents of **process control block**?
3. Define long term scheduler and short term scheduler?

4. Compare and contrast short term, medium term and long term scheduling.?
5. Discuss **criteria** involved in scheduling a process?
6. Explain about inter process communication **(IPC)**?
7. Demonstrate two different **operations** performed on processes?
8. What is convey effect? Explain with an example?
9. Discuss the **problem** involved in priority scheduling algorithm with a suitable example and provide a **solution** to that problem?
10. Differentiate shared memory and message passing models of process communication?
11. Explain the role of schedulers with the help of process transition diagram?
12. With a suitable example explain about context switching?
13. Write about Priority and SJF(Shortest Job First) scheduling algorithms with an example.

Problems:

1. Suppose that the following processes arrive for execution at the times indicated

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

What is the average waiting and turnaround time for these processes using

a) FCFS scheduling algorithm
b) SJF Non Preemptive scheduling algorithm
c) SJF Preemptive scheduling algorithm

2. Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds.

Process	Arrival Time	Burst Time
P_1	0	10
P_2	3	6
P_3	7	1
P_4	8	3

Calculate average waiting and average turnaround time using

a) Non preemptive priority CPU scheduling algorithm

b) Preemptive priority CPU scheduling algorithm

c) Round robin scheduling algorithm

3. Consider the following set of processes, with the arrival times and the CPU-burst times given in milliseconds (**GATE-CS-2004**)

Process	**Arrival Time**	**Burst Time**
P1	0	5
P2	1	3
P3	2	3
P4	4	1

What is the average turnaround time for these processes with the preemptive shortest remaining processing time first (SRPT) algorithm ?

4. Consider the following set of Processes with CPU Burst times in milliseconds, arrival times in milliseconds and Priorities:

Process	**Burst time**	**Arrival Time**	**Priority**
P1	8	1	2
P2	5	0	1
P3	14	2	4
P4	3	4	3

Draw the Gantt Chart. Calculate Average Turnaround Time and Average Waiting Time for

i) Round Robin (if Time Quantum = 4msec)

ii) Priority Scheduling.

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

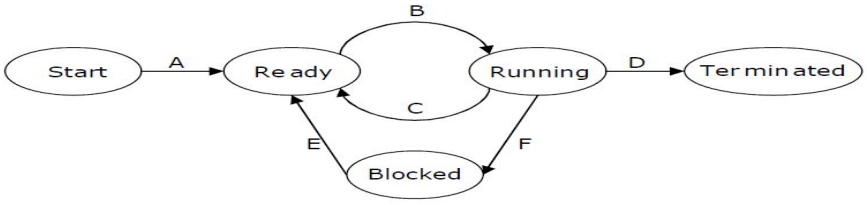
1. An operating system uses Shortest Remaining Time first (SRT) process scheduling algorithm. Consider the arrival times and execution times for the following processes: **[GATE 2007]**

Process	Execution time	Arrival time
P1	20	0
P2	25	15
P3	10	30
P4	15	45

What is the total waiting time for process P2? [ ]

A. 5
B. 15
C. 40
D. 55

2. In the following process state transition diagram for a uni processor system, assume that there are always some processes in the ready state: Now consider the following statements: **[GATE 2009]**



I. If a process makes a transition D, it would result in another process making transition A immediately.

II. A process P2 in blocked state can make transition E while another process P1 is in running state.

III. The OS uses pre emptive scheduling.

IV. The OS uses non-pre emptive scheduling.

Which of the above statements are TRUE? [ ]

A. I and II

B. I and III

C. II and III

D. II and IV

3. Which of the following statements are true? **[GATE 2010]**

a) Shortest remaining time first scheduling may cause starvation

b) Pre emptive scheduling may cause starvation

c) Round robin is better than FCFS in terms of response time [ ]

A. I only C. I and III only

B. II and III only D. I,II and III.

4. Consider the following table of arrival time and burst time for three processes P0, P1 and P2. **[GATE 2011]**

Process	Arrival time	Burst Time
P0	0 ms	9 ms
P1	1 ms	4 ms
P2	2 ms	9 ms

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?

A.5.0 ms B. 4.33 ms C. 6.33 D. 7.33.

5. Consider the 3 processes, P1, P2 and P3 shown in the table.**[GATE 2012]**

Process	Arrival time	Time Units Required
P1	0	5
P2	1	7
P3	3	4

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units) are [ ]

A. FCFS: P1, P2, P3
RR2: P1, P2, P3
B. FCFS: P1, P3, P2
RR2: P1, P3, P2
C. FCFS: P1, P2, P3
RR2: P1, P3, P2
D. FCFS: P1, P3, P2
RR2: P1, P2, P3

6. A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero?**[GATE2013]**
   A. This algorithm is equivalent to the first-come-first-serve algorithm.[ ]
   B. This algorithm is equivalent to the round-robin algorithm.
   C. This algorithm is equivalent to the shortest-job-first algorithm.
   D. This algorithm is equivalent to the shortest-remaining-time-first algorithm.
7. An operating system uses *shortest remaining time first* scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds):

Process	Arrival time	Burst Time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

The average waiting time (in milliseconds) of the processes is ____**(GATE-2014)**

8. Consider the following set of processes that need to be scheduled on a single CPU. All the times are given in milliseconds.

Process Name	Arrival Time	Execution Time
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Using the *shortest remaining time first* scheduling algorithm, the average process turnaround time (in msec) is **(GATE-2014)**

9. Consider a uniprocessor system executing three tasks T_1, T_2 and T_3, each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of 3, 7 and 20 milliseconds, respectively. The priority of each task is the inverse of its period, and the available tasks are scheduled in order of priority, with the highest priority task scheduled first. Each instance of T_1, $T_2$and T_3 requires an execution time of 1, 2 and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1st millisecond and task preemptions are allowed, the first instance of T_3 completes its execution at the end of _____ milliseconds.

(GATE-2015)

A. 5 **B.** 10 **C.** 12 **D.**15 [ ]

10. For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time?

(GATE-2015)

Process	Arrival Time	Burst Time
A	0	3
B	1	6
C	4	4
D	6	2

A. First Come First Serve [ ]

B. Non – preemptive Shortest Job First

C. Shortest Remaining Time

D. Round Robin with Quantum value two

11. SConsider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining-time first.

Process	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

The average turnaround time of these processes is_____. **(GATE-2016)**

12. Consider the following CPU processes with arrival times (in mili seconds) and length of CPU bursts (in mili seconds) as given below: **(GATE-2017)**

Process	Arrival time	Burst time
P1	0	7
P2	3	3
P3	5	5
P4	6	2

If the pre-emptive shortest remaining time first scheduling algorithm is used to schedule the processes, then the average waiting time across all processes is_____ milliseconds.

13. Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds) , and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is_____ (**GATE2017)**

UNIT – III

Memory Management Strategies

Objectives:

- Students will be able To develop the concepts of memory management techniques

Syllabus:

Memory Management Strategies

- Swapping, contiguous memory allocation (memory mapping and protection, memory allocation, fragmentation), paging (basic method, hardware support, shared pages), Segmentation (basic method, Hardware).

Virtual-Memory Management:

- Demand paging (Basic concepts, Performance of Demand Paging), page replacement (FIFO, Optimal, LRU), Allocation of frames (Minimum number of frames, Allocation Algorithms), Thrashing (Cause of Thrashing, Working-Set model, Page fault frequency).

Subject Outcomes:

Students will be able to

- Describe the benefits of a virtual memory system
- Explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- Discuss the principle of the working-set model
- Describe various ways of organizing memory hardware
- Discuss various memory-management techniques, including paging and segmentation
- Describe both pure segmentation and segmentation with paging

Learning Material

1. Swapping:

- A process must be in memory to be executed.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **For example:**
    - Consider a **round –robin** CPU scheduling algorithm, when time quantum expires , the memory manager will start to swap out the process that just finished and to swap another process into memory space that has been freed.
    - Swapping policy is used for **priority based** scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower priority processes and then load and execute the higher priority process.
- **Roll out–**Swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- **Roll in:** Swapping in the higher priority process is known as roll-in.
- Swapping requires a backing store.
- The backing store is commonly a fast disk.
- It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- When the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks whether the next process in the queue is in memory.
- If not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.

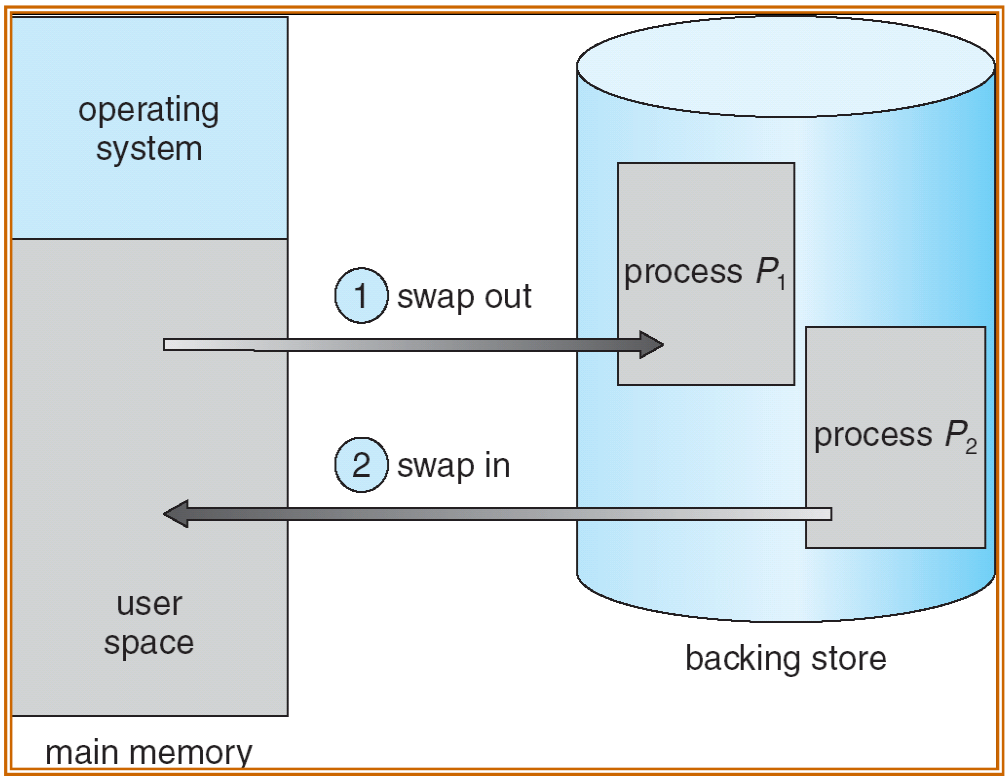


Fig: Swapping of two processes using a disk as a backing store

- The context-switch time in such a swapping system is fairly high.
- **For example**
  - If the user process is 10 MB in size and the backing store is a standard hard disk with a transfer rate of 40 MB per second.
  - The actual transfer of the 10-MB process to or from main memory takes

    10000 KB/40000 KB per second = 1/4 second

    = 250 milliseconds.

2. Contiguous Memory Allocation:

- The main memory must accommodate both the operating system and the various user processes.
- So the main memory must be allocated in an efficient way possible

- The memory is usually divided into two partitions:

  1) Operating system

  2) User processes.

- We can place the operating system in either low memory or high memory.
- The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, so programmers place the operating system in low memory.
- We have to allocate all available memory to the processes.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.

A) Memory mapping and Protection:

- We can provide memory mapping and protection by using a relocation register and a limit register.
- The relocation register contains the value of the smallest physical address;
- the limit register contains the range of logical addresses
- For example, relocation = 100040 and limit = 74600.
- With relocation and limit registers, each logical address must be less than the limit register;
- the MMU maps the logical address *dynamically* by adding the value in the relocation register.

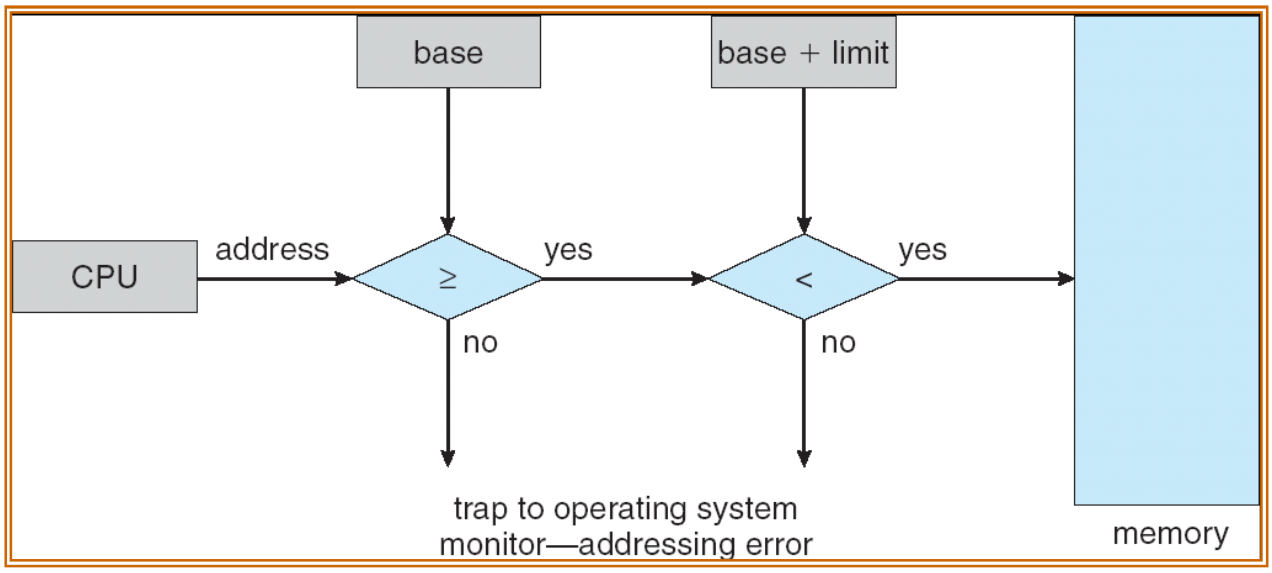


Fig: Hardware support for relocation and limit register

- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- Because every address generated by a CPU is checked against these registers.
- We can protect both the operating system and the other users' programs and data being modified by the running process.
- The relocation-register scheme provides an effective way to allow the operating system size to change dynamically.

B) Memory Allocation:

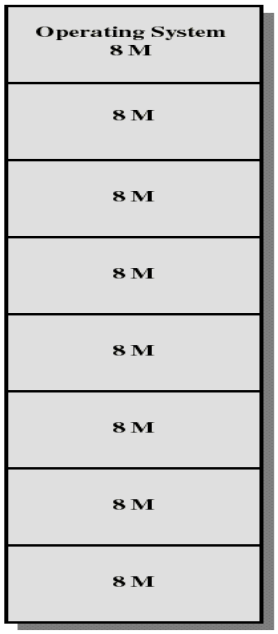
- **Multiprogramming with Fixed number of Tasks: (Fixed Partitioning)**
  - OS occupies some fixed portion of main memory
  - Rest of **main memory is divided into several fixed-number of partitions of equal size**
  - Each partition may contain **exactly one process**
  - Any **process whose size is <= to the partition size** can be loaded into any available partition
  - If all the partitions are full – swapping is done

- Difficulties with equal – sized fixed partitions:
  - A program may be too big to fit into a partition
  - Example:
    1. Partition size – 8 M

       Program size – 20 M

       Main memory utilization is extremely inefficient
    2. Program size – 2 M

       Partition size – 8 M = 6M wasted

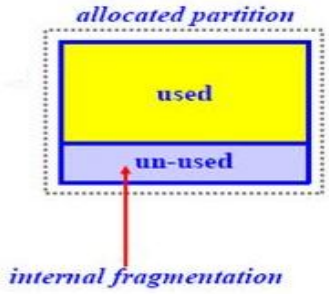


Equal-size partitions

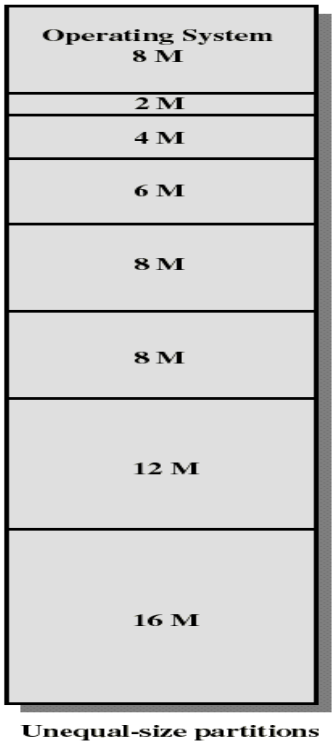
- **Drawback: Internal fragmentation:**
  - Amount of space wasted inside a partition allocated to a process is called as Internal Fragmentation.
  - allocated memory may be slightly larger than requested memory
  - Example:
  - Program size –> 2 M

    Partition size –> 8 M

    = 6M Internal fragmentation



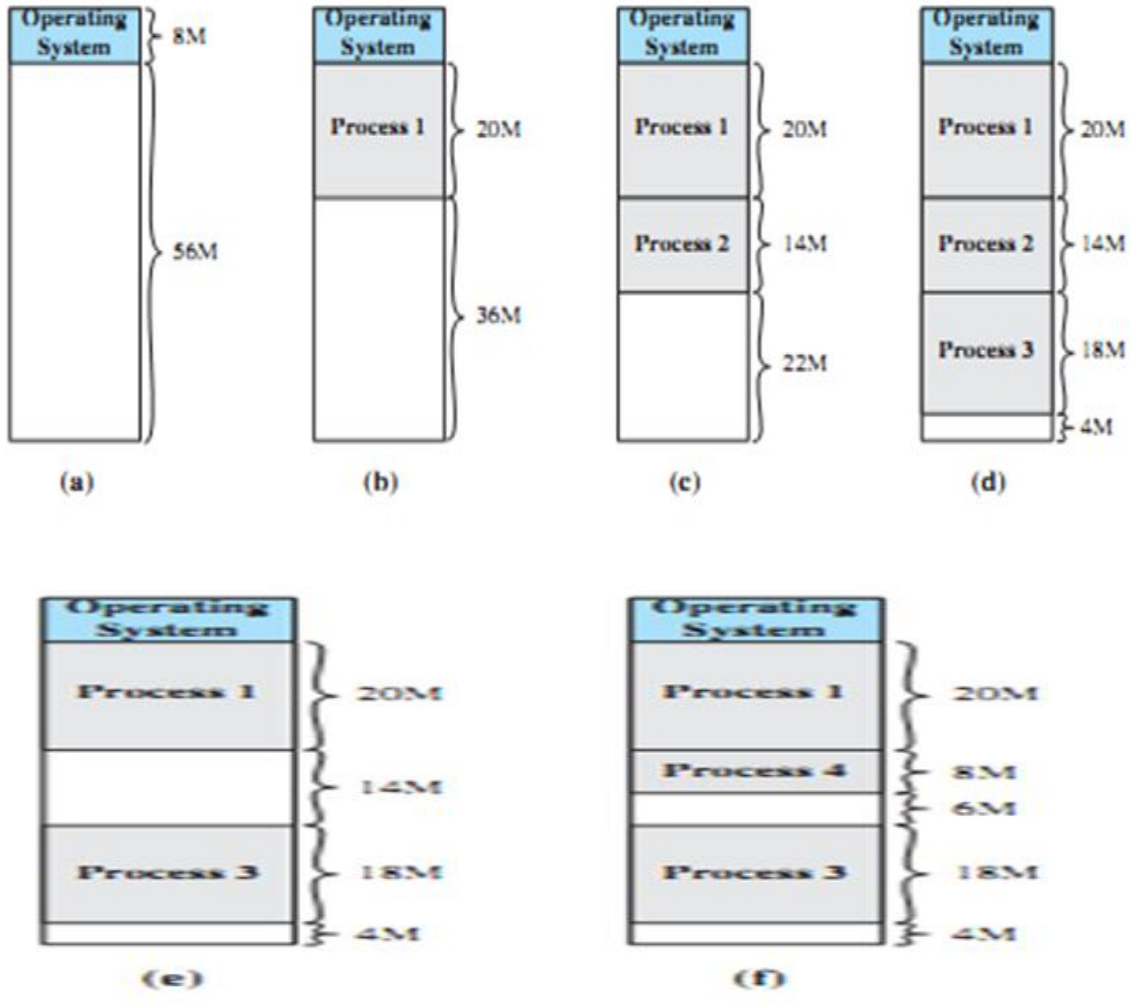
- **MFT – Unequal size partitions**
  - **Example:**



Unequal-size partitions

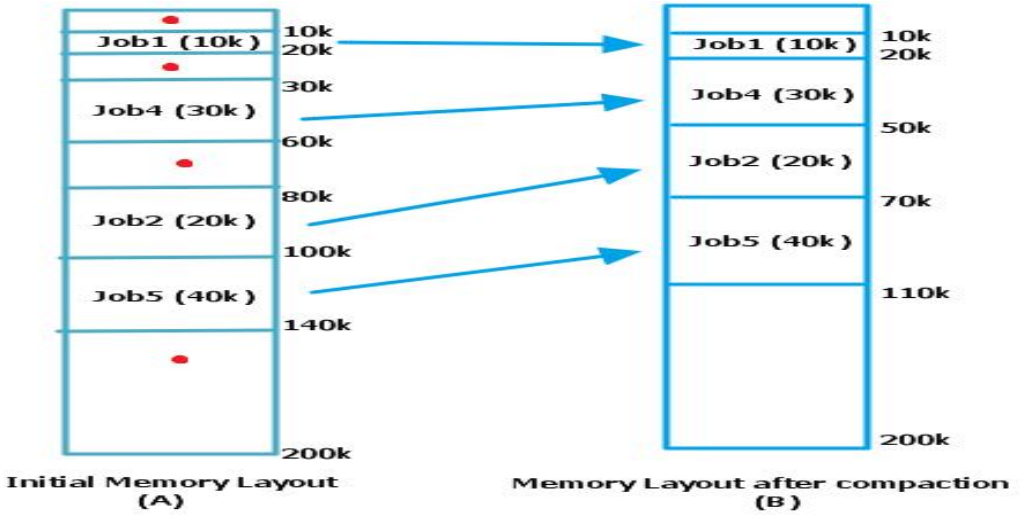
- **Advantages:**
  - Simple to implement
  - Little OS overhead
- **Disadvantages:**
  - Internal Fragmentation
  - Maximum number of active processes is fixed
  - Degree of multiprogramming is fixed.

- **Multiprogramming with Variable number of Tasks (Dynamic partitioning):**
  - Partitions are of variable length and number
  - Initially all memory available for user processes is considered as one large block of available memory (hole)
  - When a process arrives and needs memory, search for a hole large enough for this process.
  - If found – allocate only as much memory as is need keeping the rest available to satisfy future requests



- **Drawback: External Fragmentation**
  - Total memory space exists to satisfy a request, but it is not contiguous.

- Storage space is fragmented into a large number of small holes
- **Solution :**
  - Compaction: OS shifts the processes so that they are contiguous and so that all of the free memory is together in one block



  - **Advantages:**
    - No Internal Fragmentation
    - More efficient use of main memory
  - **Disadvantages:**
    - External Fragmentation
    - Inefficient use of processor due to the need for compaction to counter external fragmentation.
    - Compaction is very expensive scheme.

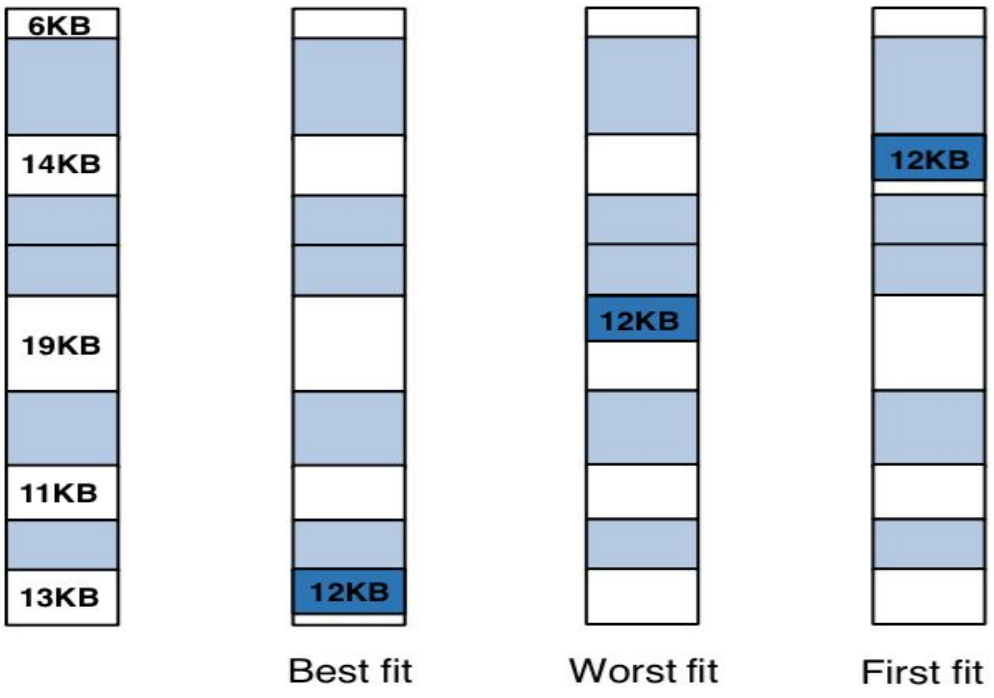
Dynamic Storage Allocation Problem:

- This problem deals with how to satisfy a request of size n from a list of free holes.
- This problem has many solutions and these strategies are used to select a free hole from a set of available holes.
  - First fit
  - Best fit
  - Worst fit

- **First fit:** Allocate the *first* hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit ended. We can stop searching when we find a free hole that is large enough.
- **Best fit**. Allocate the *smallest* hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- **Worst fit:** Allocate the *largest* hole. We must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole.

Both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization.

Example: New process size is 12 KB



- **50-percent rule**: If we are having *N* allocated blocks, another 0.5 *N* blocks will be lost to fragmentation. That is, one-third of memory may be unusable. This property is known as 50-percent rule.

3. Paging:

- Paging permits a program memory to be noncontiguous.
- Thus allowing a program to be allocated physical memory whenever it is available.
- Every address generated by the CPU is divided into two parts:
  1. Page number (p)
  2. Page offset (d).
- The page number is used as an index into a page table.
- The page table contains the base address of each page in physical memory.
- This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

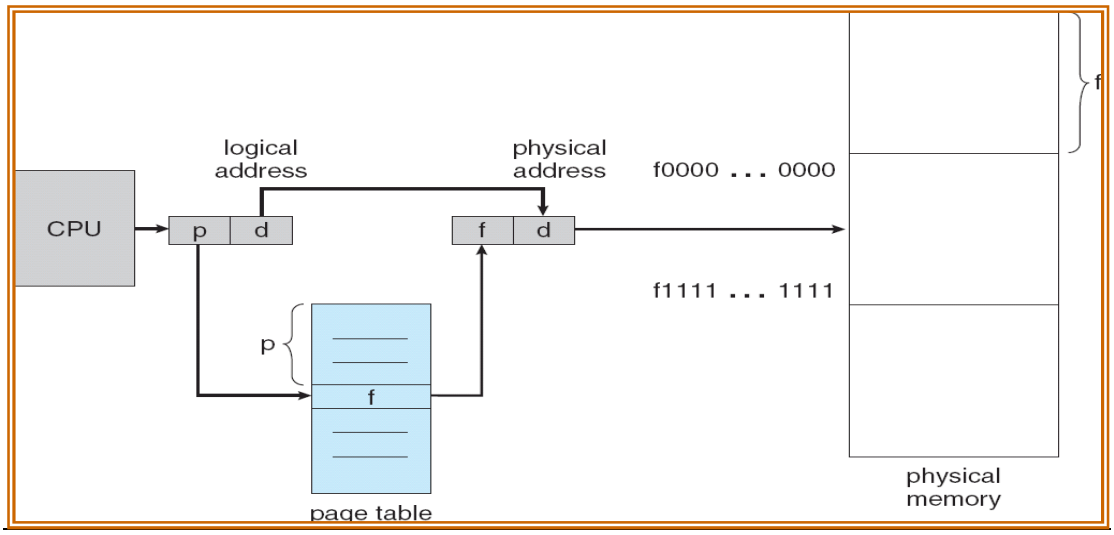


Fig: Paging Hardware

A) Basic method:

- Physical memory is broken into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called **pages**.
- When a program is to be executed, its pages are loaded into any available frames.
- The page table is defined to translate from user pages to memory frames (IBM 370 uses 2048 or 4096 bytes for page).

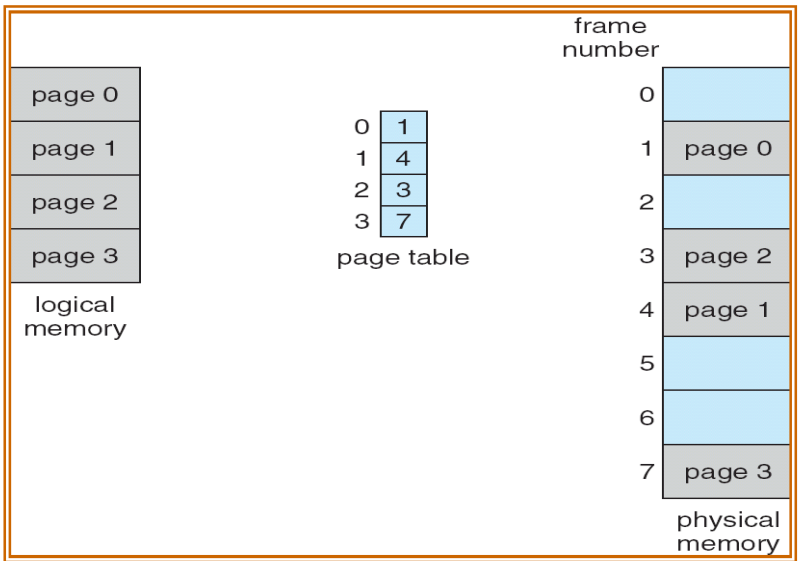


Fig: Paging model of logical and physical memory.

- If a page size is 2n addressing units (bytes or words) long, then the lower -order n bits of a logical addresses designates the page offset and the remaining higher order bits designates the paging number.

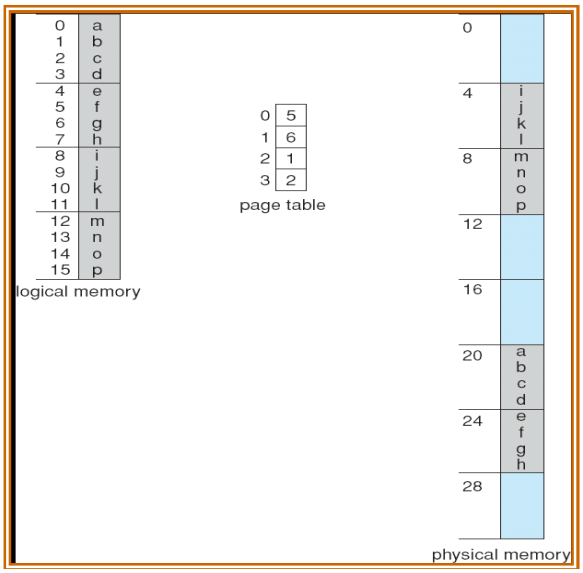


Fig: Paging example for a 32-byte memory with 4-byte pages.

- Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. This logical address 0 maps to physical address 20 [= (5 x 4) + 0].

- Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 x 4) + 3].
- Every Logical address is mapped by the passing hardware to same physical address.
- physical address of word = (frame number x page size offset)

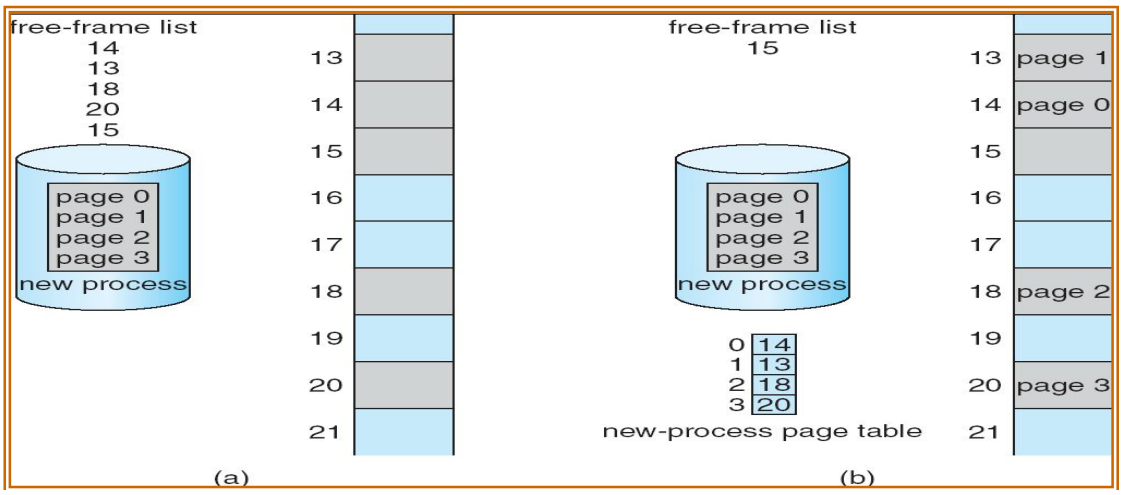


Fig: Free frames (a) before allocation and (b) after allocation.

B) Hardware support:
Implementation of page table:

a. Registers
b. PTBR
c. TLB

➢ **Registers:**

- Page table is implemented as a set of dedicated registers
- Very high speed- to make the translation efficient.
- Load and modification of these registers are controlled only by the operating system.
- Example: DEC PDP-11
- If page table is small-It is satisfactory.
- Most of the computers allow page table to be very large(1 Million entries).

- **PTBR**
  - Page table is placed in main memory
  - PTBR points to the page table.
  - Changes in page table-changing in PTBR register.
  - **Drawback**: If we want to access location *i,* we must first index into the page table, using the value in the PTBR offset by the page number for 'i'.
  - This task requires a memory access.
  - It provides us with the frame number, which is combined with the page offset to produce the actual address.
  - We can then access the desired place in memory.

  With this scheme, *two* memory accesses are needed to access a byte
    i. one for the page-table entry
    ii. One for the byte).
  - The memory access is slowed by a factor of 2. This delay would be intolerable under most circumstances.
- The solution for this problem is **TLB** (Translation Look –aside Buffer),
- It is a special, small, fast lookup hardware cache.
- The TLB is used with page tables; it contains only a few of the page-table entries.
- When a logical address is generated by the CPU, its page number is presented to the TLB.

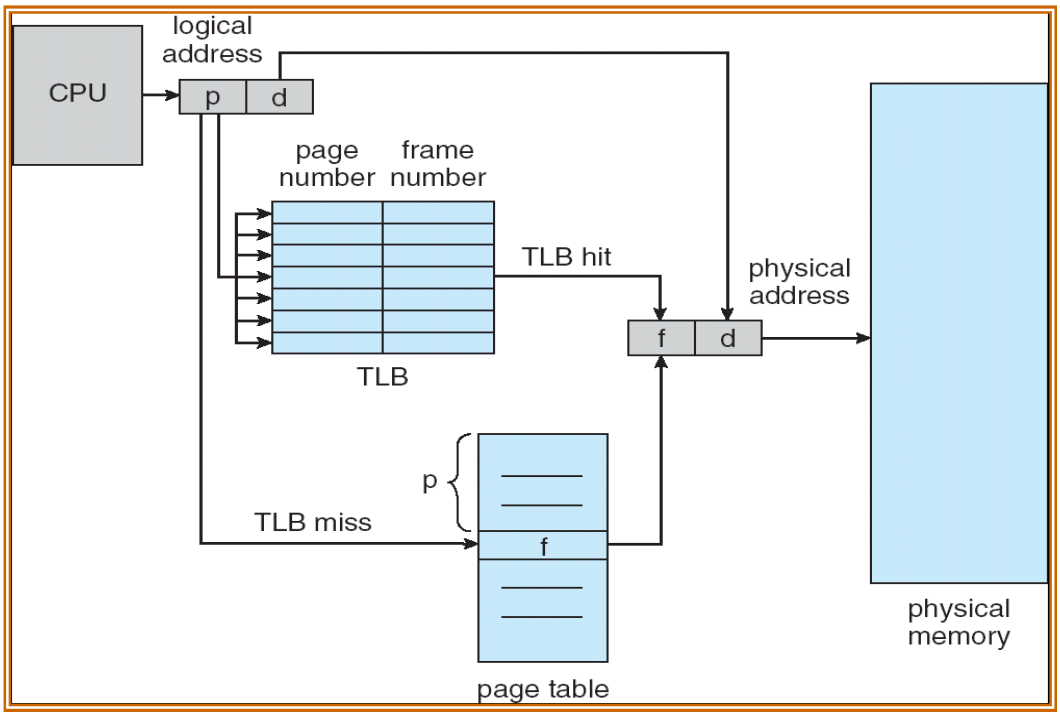


Fig: Paging hardware with TLB.

TLB Miss: If the page number is not in the TLB

TLB Hit: If the page number is in the TLB

Hit Ratio:

- The percentage of times that a particular page number is found in the TLB is called the hit ratio.
- **Example:** 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time.
    - If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then the mapped-memory access takes 120 nanoseconds when the page number is in the TLB.
    - If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of 220 nanoseconds.

- To find the effective we weight the case by its probability:
- Effective access time = 0.80 x 120 + 0.20 x 220
  = 140 nanoseconds.
- In this example, we suffer a 40-percent slowdown in memory-access time (from 100 to 140 nanoseconds).
- For a 98-percent hit ratio,
  - effective access time = 0.98 x 120 + 0.02 x 220
    =122 nano seconds

Protection:

- Memory protection in a paged environment is accomplished by protection bits associated with each page.
- These bits are kept in the page table. One bit can define a page to be read/write or read-only.
- Every reference to memory goes through the page table to find the correct frame number.
- At the same time physical address is computed.
- The protection bits are checked to verify that no writes are being made on read-only page.
- An attempt is treated as a trap to the operating system.
- One mode bit is added to the page table. Valid/ Invalid bit.
- The OS sets this bit for each page to allow or disallow access to that page. Ex: A 14-bit address space (0 to16383).
- Addresses in page 0, 1, 2, 3, -----5 are mapped normally through the page table.
- Any attempt to generate an address in page 6 or 7 is a trap to operating system.

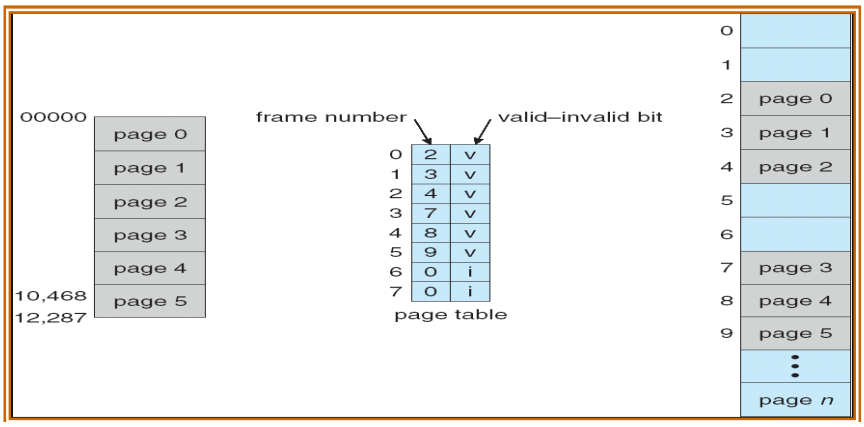


Fig: Valid (v) or invalid (i) bit in a page table.

C) Shared Pages:

- Another advantage of paging is the possibility of sharing common code (Time sharing system).

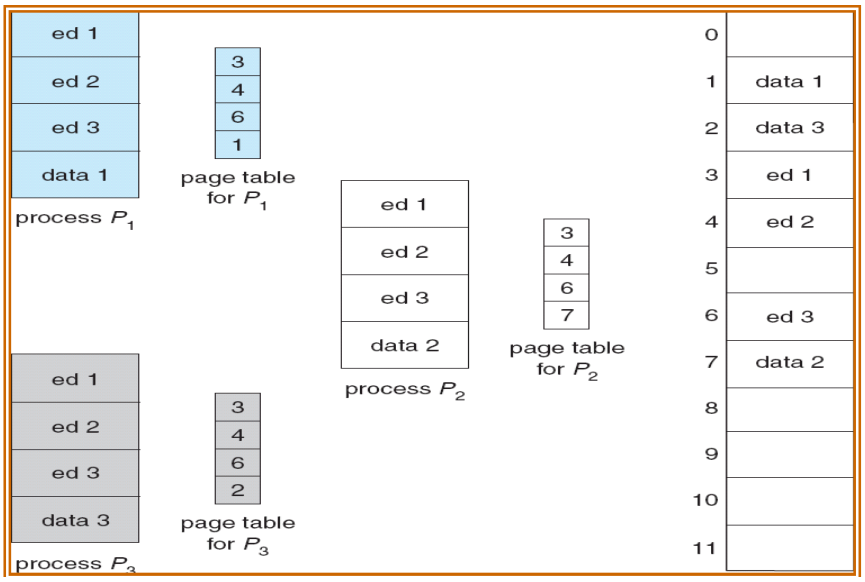


Fig: Sharing of code in a paging environment.

- Consider a system that supports 40 users, each of which executes a text editor 30K and 5K for data space. We would need 1400K (35*40).
- If the code is reentrant code it could be shared, three-page editor being shared among three processes. Each process has its own data page.

- If the code is “reentrant” (pure code) then it never changes during execution.
- Two or more processes can execute the same code at the same time.
- Each process has its own copy of registers and data storage to hold the data for its execution.
- So we need a copy of the editor (30 K), plus 40 copies of the 5 K of data space for user, total space required is now 230 K.
- Compilers, assemblers, database systems can also be shared.

4.Segmentation:

- The user's view of memory is not the same as the actual physical memory.
- The user's view is mapped onto physical memory segmentation is a memory management scheme which supports the user view of memory.

A) Basic method

- A logical address space is a collection of segments. Each segment has a name and offset within the segment.

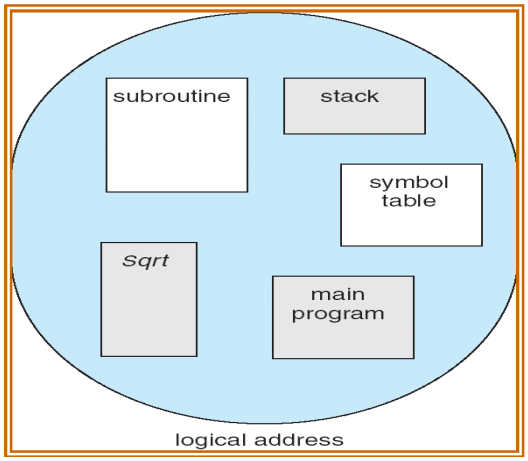


Fig: User's view of a program.

- We must define an implementation to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by a segment table.

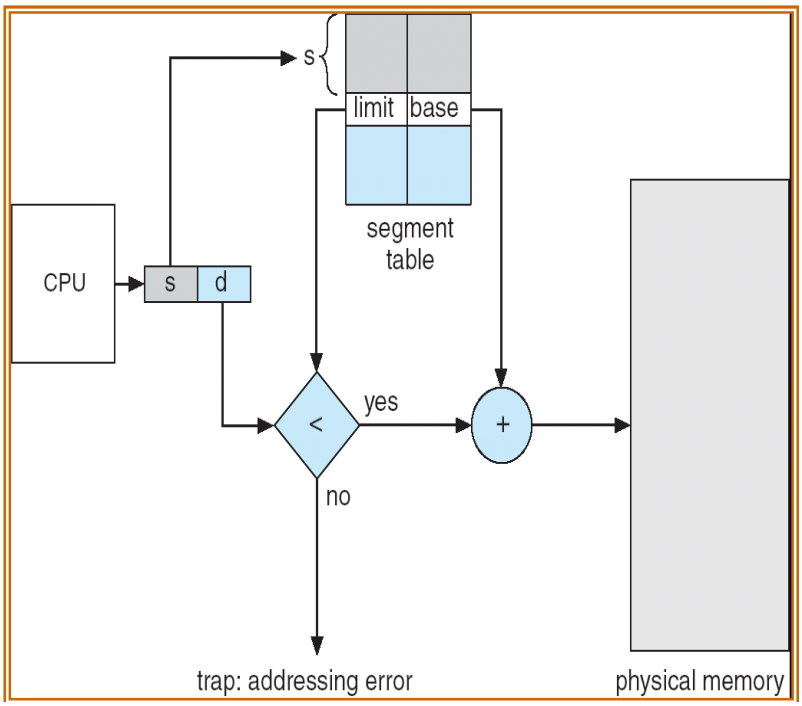


Fig: Segmentation hardware.

- Logical address consists of two parts: a segment number, s, and an offset into that segment, *d.*
- The segment numbers are used as an index to the segment table.
- Each entry of the segment table has a *segment base* and a *segment limit.*
- The offset *d* of the logical addresses must be between 0 and the segment limit. If it is not, we trap to the operating system.

B) Hardware:

- If it is legal, it is added to the segment base address to produce the addresses in physical memory of the desired word.
- The segment table is array of base/limit register pairs.

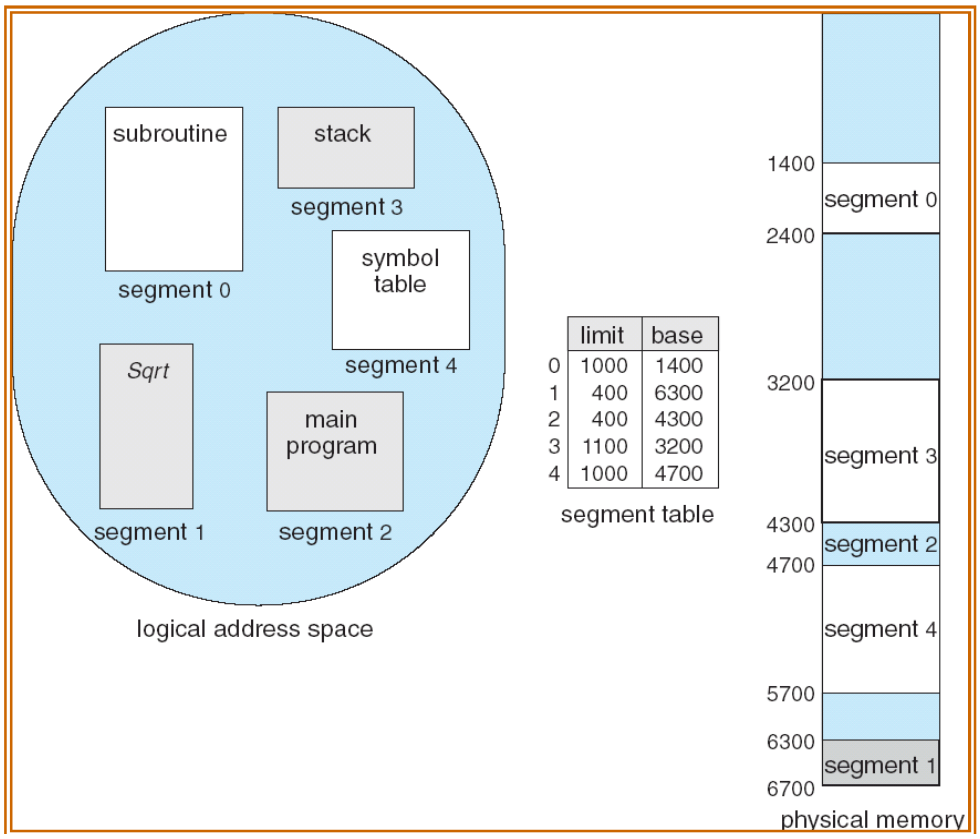


Fig: Example of segmentation.

- We have five segments the segment table has separate entry for each segment.
- The segment table contains the beginning address and the limiting address.
- **For example,** segment 2 is 400 words long a beginning at 4300. So a reference word 53 of segment 2 is mapped on the physical address as 4300+53=4353.

Virtual-Memory Management:

- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- One major advantage of this scheme is that programs can be larger than physical memory.

- Virtual memory abstracts main memory into an extremely large , uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory storage limitations.
- Virtual memory also allows processes to share files easily and to implement shared memory.
- The instructions that are currently executing must be in physical memory.
- In order to meet this requirement the entire logical address space should placed in physical memory.
- Dynamic loading can help to ease this requirement.
- In many cases the entire program is not needed in main memory. For instance consider the following.
  - Certain options and features of a program may be used rarely.
  - Arrays, lists, and tables are often allocated more memory than they actually need. Example an array may be declared 100 by 100 elements, even though it uses 10 by 10 elements.

Benefits of virtual memory:

- A program would no longer be constrained by the amount of physical memory that is available.
- CPU utilization and throughput will be increased and response time or turnaround time will be decreased.
- Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.
- Thus, running a program that is not entirely in memory would benefit both the system and the user:

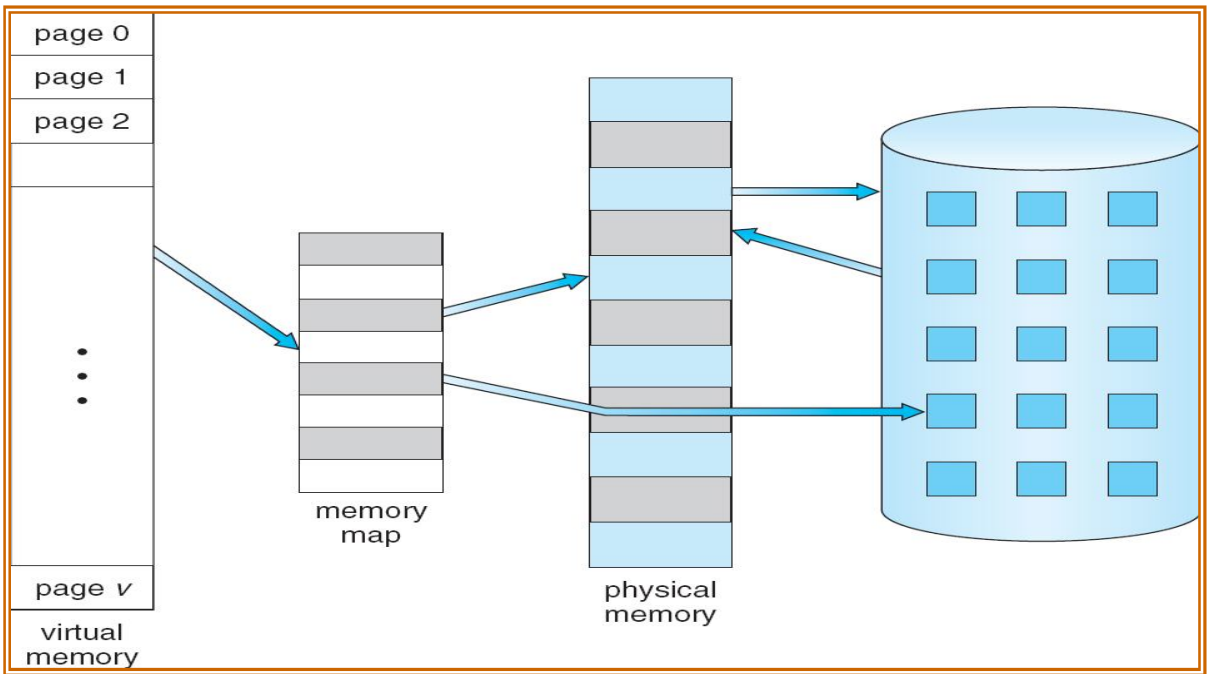


Fig: Diagram showing virtual memory that is larger than physical memory.

- Virtual memory involves the separation of logical memory from physical memory.
- This separation, allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.
- Virtual memory makes the task of programming much easier.

Virtual address space:

- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.
- Here a process begins at a certain logical address say, address 0—and exists in contiguous memory.
- Physical memory may be organized in page frames, that the physical page frames assigned to a process may not be contiguous.

- Memory management unit (MMU) to map logical pages to physical page frames in memory.

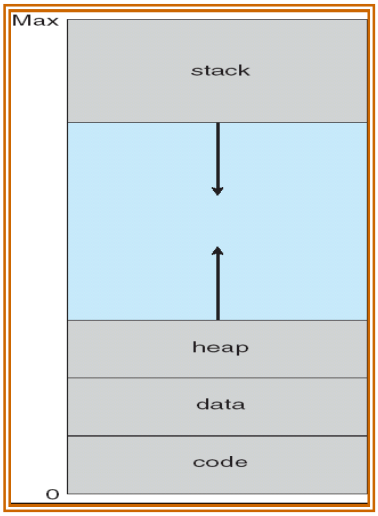


Fig: Virtual address space

- Here the heap to grow upward direction memory as it is used for dynamic memory allocation.
- Similarly, we allow for the stack to grow downward in memory through successive function calls.
- The large blank space (or hole) between the heap and the stack is part of the virtual address space but will require actual physical pages only if the heap or stack grows.
- Virtual address spaces that include holes are known as sparse address spaces.
- Virtual memory also allows files and memory to be shared by two or more processes through page sharing. This leads to the following benefits:
  - System libraries can be shared by several processes through mapping of the shared object into a virtual address space.

- Virtual memory enables processes to share memory. Virtual memory allows one process to create a region of memory that it can share with another process.
- Virtual memory can allow pages to be shared during process creation with the fork ( ) system call, thus speeding up process creation.

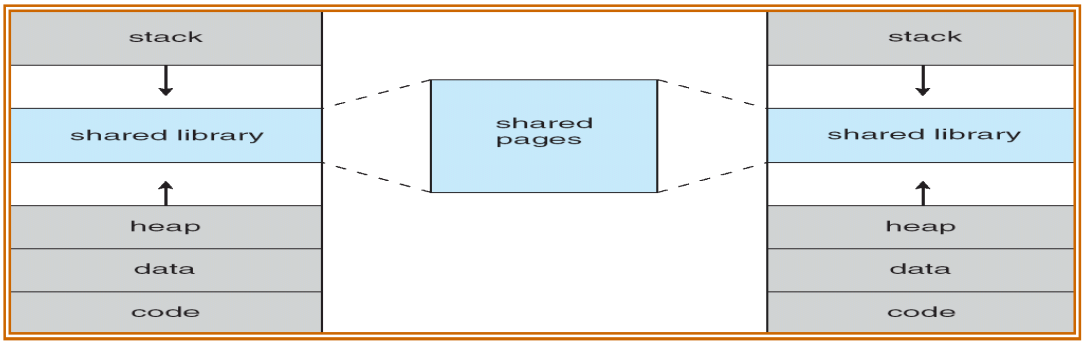


Fig: Shared library using virtual memory.

1. Demand paging:

- Initially load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems.
- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

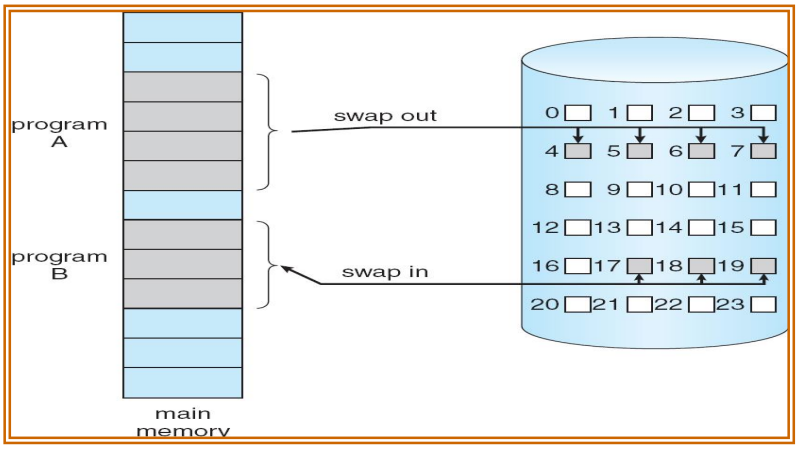


Fig: Transfer of a paged memory to contagious disk space.

- Demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper.
- **Lazy swapper:** It never swaps a page into memory unless that page will be needed.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. Here the word ***pager*** is used, rather than *swapper,* in connection with demand paging.

Basic Concepts:

- What happens if the process tries to access a page that was not brought into memory?
- Access to a page marked invalid causes a paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.
- This trap is the result of the operating system's failure to bring the desired page into memory.

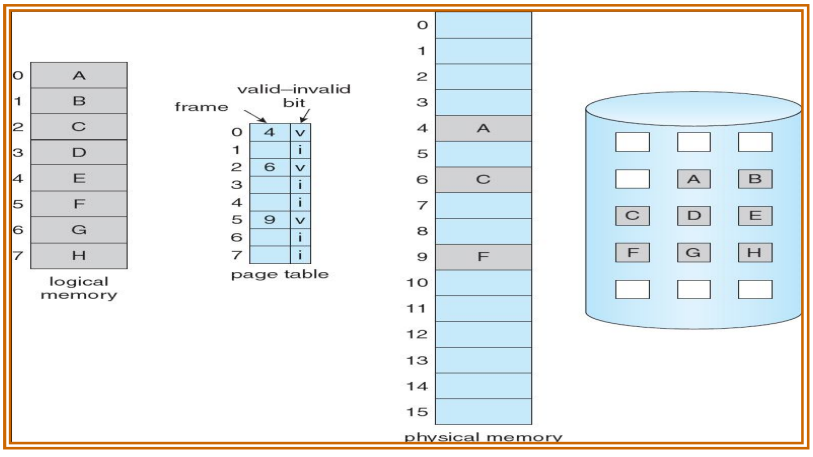


Fig: Page table when some pages are not in memory.

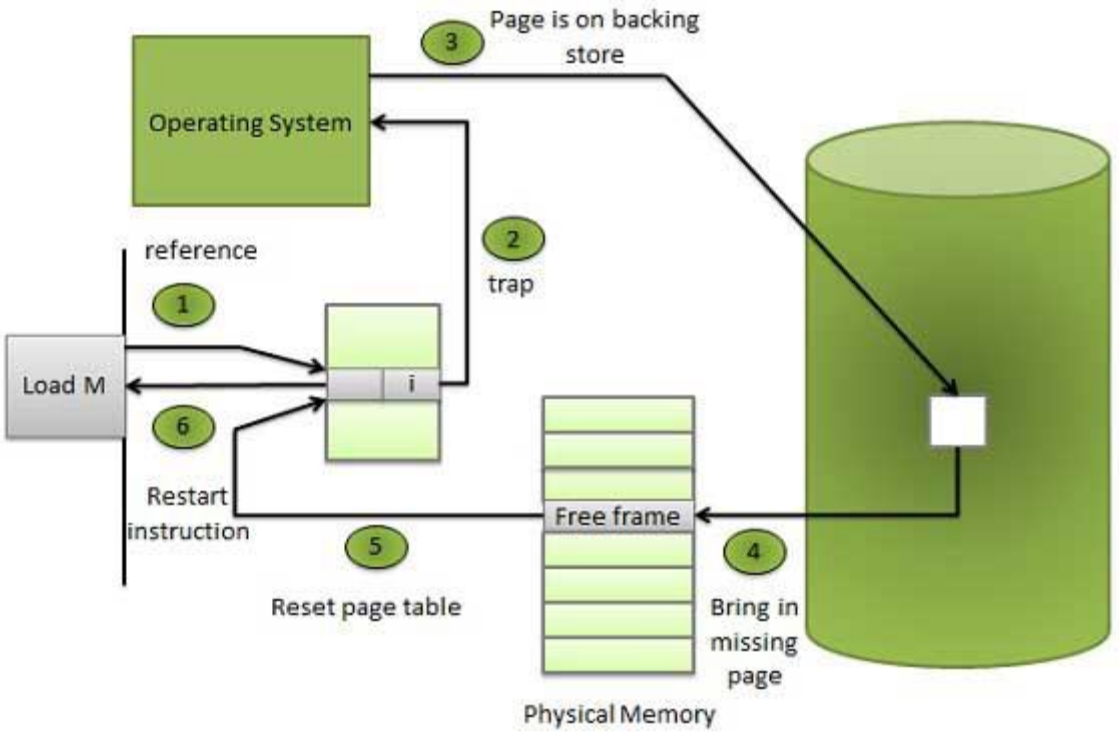


Fig: Steps in handling a page fault.

The procedure for handling this page fault is straightforward:

1) We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an Invalid memory access.

2) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3) We find a free frame (by taking one from the free-frame list, for example). We schedule a disk operation to read the desired page into the newly allocated frame.

4) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

5) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

6) Restart the instruction that was interrupted. By the illegal address –trap. The process can now access the page as if it had always been in the memory.

The hardware is same for paging and swapping

- **A page table** with the ability to mark an entry invalid through a valid/invalid bit.
- **Secondary memory:** This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known swap space.

Performance of Demand Paging:

A page fault causes the following sequence to occur:
1. Trap to the operating system.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    a. Wait in a queue for this device until the read request is serviced.
    b. Wait for the device seek and/ or latency time.
    c. Begin the transfer of the page to a free frame.
6. While waiting, allocate the CPU to some other user (CPU scheduling, optional).
7. Receive an interrupt from the disk I/0 subsystem (I/0 completed).
8. Save the registers and process state for the other user (if step 6 is executed).
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

Page Replacement:

- While executing a user process, a page fault occurs.
- The hardware traps to the operating system; which checks its internal tables to see that this page fault and not an illegal memory access.
- This operating system determines where the desired page is residing on the backing store, but then finds that there are *no* free frames on the free-frame list; all memory is in use.

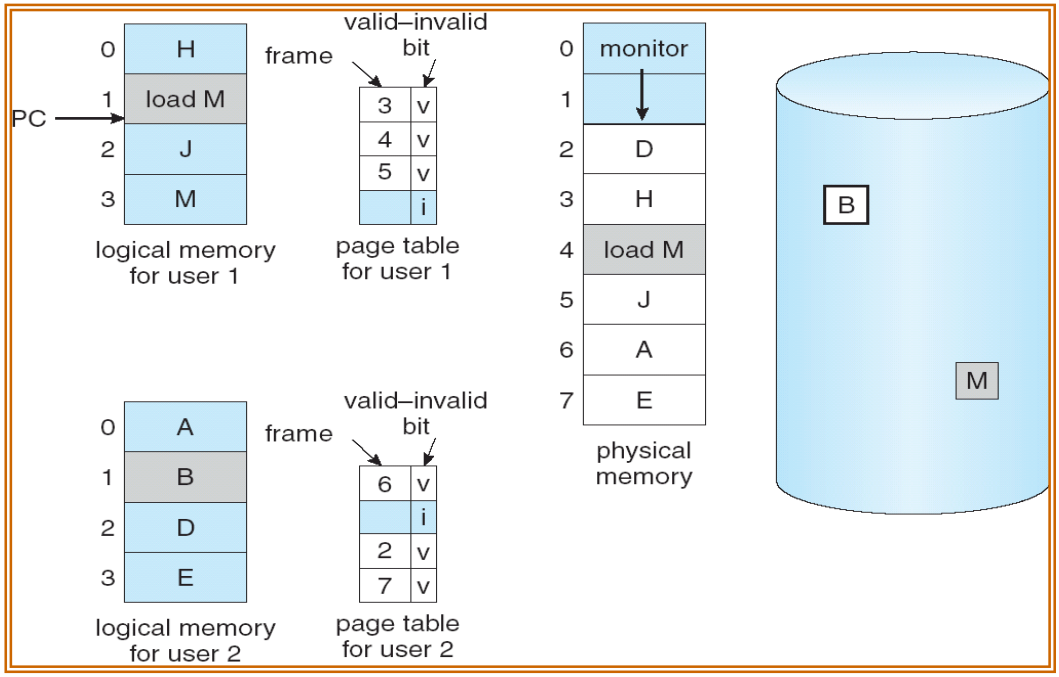


Fig: Need for Page Replacement

A) Basic Page Replacement:

- If no frame is free, find one which is not currently being used and free it.
- We can free a frame by writing its contents to the backing store, and changing the page table (and all other tables) to indicate that the page is no longer in memory.
- The freed frame can now be used to hold the page for which the process faulted.

1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

- This overhead by the use of a dirty **bit**.
- Each page or frame may have a dirty bit associated with it in the hardware.
- The modify bit for a page is set by the hardware.
- When we select a page for replacement, we examine its dirty bit.
- If the bit is set, we know that the page has been modified since it was read in from the baking store.
- In this case, we must write that page to the backing store.

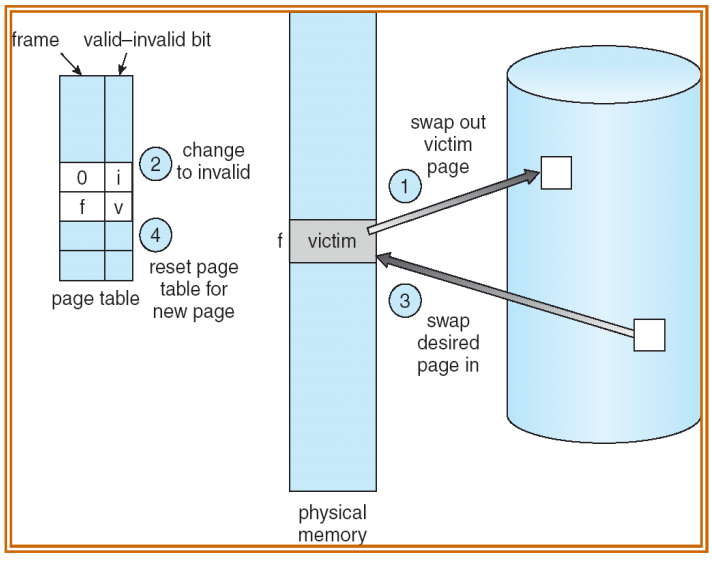


Fig: Page Replacement

- With demand paging, the size of the logical address space is no longer constrained by physical memory.
- If we have a user process of twenty pages, we can execute it in ten frames simply by using **demand paging**
- To implement demand paging two problems to be solved ***frame-allocation algorithm*** and a ***page-replacement algorithm*****.**
- If page replacement is required, we must select the frames that are to be replaced.

B) Page Replacement Algorithms:

- We have to select the algorithm which the lowest page-fault rate.
- An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults.
- The string of memory references is called a **reference string**.
- We can generate reference strings artificially (by using a random-number generator

1. FIFO Page Replacement:T

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- We can create a FIFO queue to hold all pages in memory.
- We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

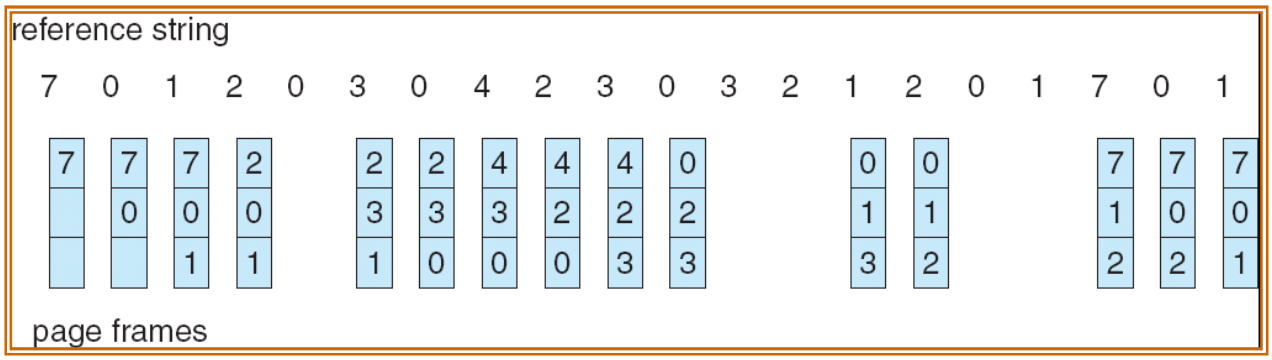


Fig: FIFO Page-Replacement algorithm

Belady's Anomaly:

- The page-fault rate may *increase* as the number of allocated frames increases. This phenomenon is called Belady's Anomaly.
- To illustrate the problems that are possible with a FIFO page-replacement algorithm, we consider the following reference string:

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  - 3 frames (3 pages can be in memory at a time per process)-9 page faults
  - 4 frames-10 page faults

2. Optimal Page Replacement:

- Replace the page that will not be used for the longest period of time.
- Optimal Page Replacement has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.

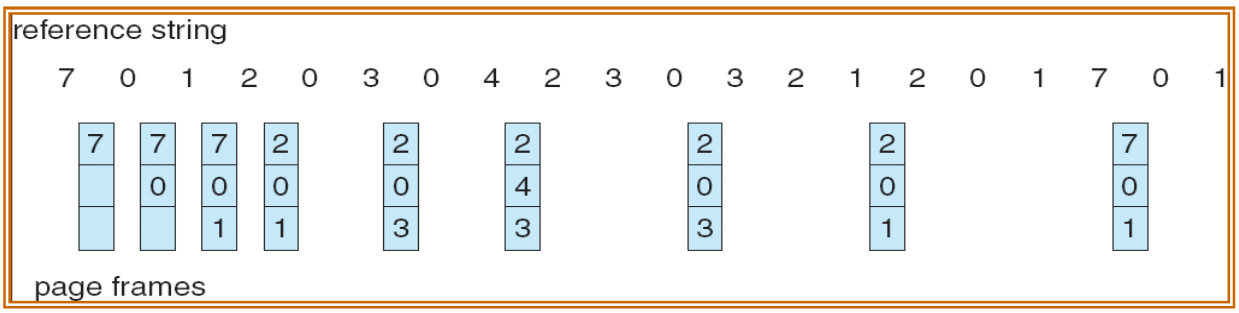


Fig: Optimal Page-Replacement algorithm

- For example, on our sample reference string, the optimal page-replacement algorithm would yield nine page faults.
- The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14.
- The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again.

3. LRU Page Replacement:

- If we use the recent past as an approximation of the near future, then we can replace then that *has not been used* for the longest period of time.
- This approach is the approach is known as LRU Page Replacement.

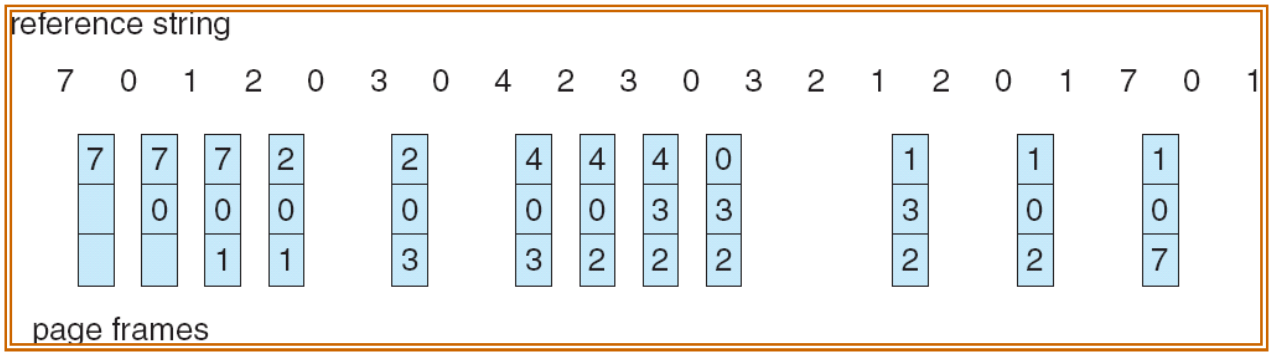


Fig: LRU Page-Replacement algorithm

- The LRU algorithm produces twelve faults.
- The first five faults are the same as those for optimal replacement. When the reference to page 4 occurs.LRU replacements sees that, of the three frames in memory, page 2 was used least recently.
- Thus, the LRU algorithm replaces page 2, not knowing that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3, since it is now the least recently used of the three pages in memory.
- Despite these problems, LRU replacement with twelve faults is much better than FIFO replacement with fifteen.

ALLOCATION OF FRAMES:

- Normally, there are fixed amounts of free memory with various processes at different time in a system.
- The question is how this fixed amount of free memory is allocated among the different processes.
- The simplest case is the single process system.
- All available memory for user programs can initially be put on the free frame list (pure demand paging).
- When the user program starts its execution, it will generate a sequence of page faults.
- The user program would get all free frames from the free frame list.
- As soon as this list was exhausted, and the more free frames are required, the page replacement algorithm can be used to select one of the in-used pages to be replaced with the next required page and so on.
- After the program was terminated, all used pages are put on the free frame list again.
- The frame allocation procedure is more complicated when there are two of more programs in memory at the same time.

A) MINIMUM NUMBER OF FRAMES:

- We cannot allocate more than the total number of available frames in the system.
- On the other hand, there is a minimum number of frames which must be allocated.
- This minimum number is determined by the instruction architecture.
- It is obvious that we must provide enough frames to hold all the different pages that any single instruction can reference.
- For example, all memory reference instructions of-a machine have only one memory address.

- So we need at least one frame for the instruction code and one frame for the memory reference.
- If one level indirect addressing is allowed, a load instruction on page m can refer to an address on page. It is an indirect reference to page k.
- We need three pages.

B) ALLOCATION ALGORITHM:

- The simplest way is to divide m available frames among n processes to give everyone an equal share, m/n frames.
- This is called **equal allocation**.
- Various processes will need different amounts of memory. If the equal allocation is applied, there can be some frames wasted.
- Therefore, other allocation scheme can be used to give available memory to each process according to its size. This is called, proportional allocation.
- Let the size of the virtual memory for process p_i be s_i, the number of frames allocated to the process p_i be a_i, and define

$$S = \sum s_i$$

- If the total number of available frames is m, then a_i can be calculated:

$$a_i = (s_i / S) * m.$$

- Of course ai must be adjust to be a integer, greater than the minimum number of frames required by the instruction set with a sum not exceeding m.
- In both of these cases, the number of frames allocated to each process may vary according to the multiprogramming level.

Global versus Local Allocation:

- When it's necessary to find free page frames, what set of pages should become candidates for replacement?
- **Local replacement** policies replace pages that belong to the process that needs the new frame.

- **Global policies** consider all unlocked frames. Most systems use global replacement because it is easy to implement, has minimal overhead, and performs reasonably well.

	Local Replacement	Global Replacement
Fixed Allocation	Rarely used -A process is given a fixed number of frames. Page faults are satisfied from this set.	This combination isn't possible
Variable Allocation	The process is given a fixed allocation and pages to be replaced are chosen from this set. Periodically, the resident, set size is re-evaluated. Pages can be added or subtracted.	Replacement pages are chosen from any page in memory. Resident set size varies, although by a blind process. This is the most common approach

THRASHING:

- Consider a process which does not have enough frames. It is possible to reduce the no of allocated frames to the minimum.
- There are some no: of pages that are in active use. if the process does not have no of frames ,it will very quickly page fault since all of the pages are in active use.
- It must replace a page which will be needed again a right way,
- Consequently if very quickly faults arrive again and again.
- This high paging activity is called “Trashing”. (A process is trashing if it is spending more time in paging than executing).

A) Causes of thrashing

- The OS monitors CPU utilization, if CPU utilization is too low;
- the degree of multiprogramming is increased by introducing a new process to the system.
- A global page replacement algorithm is used.

- It will create page faults.
- If the graph is drawn between CPU utilization and multiprogramming as the degree of multiprogramming increases CPU utilization also increases until maximum reached.
- If degree of multi programming increased further trashing sets and CPU utilization decreases slowly.

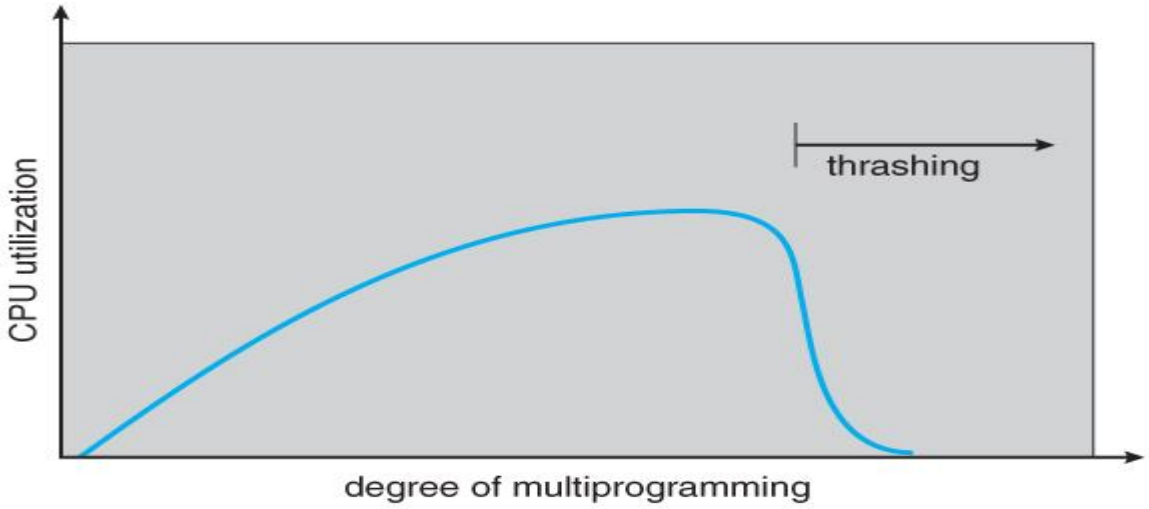


Fig: showing trashing between CPU utilization and degree of multiprogramming.

- The effect of thrashing can be limited by using a locator priority replacement algorithm.
- With local replacement if one process starts trashing it cannot steal frames from another process and cause it to trash also.
- If process are trashing they will be in the queue for paging device most of the time average service time for page fault increases there by effective access time increases.

B) Working-Set Model

- Δ ≡ working-set window ≡ a fixed number of page references Example: 10,000 instruction.

- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  1. if Δ too small will not encompass entire locality
  2. if Δ too large will encompass several localities
  3. if $\Delta = \infty \Rightarrow$ will encompass entire program
  4. $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes

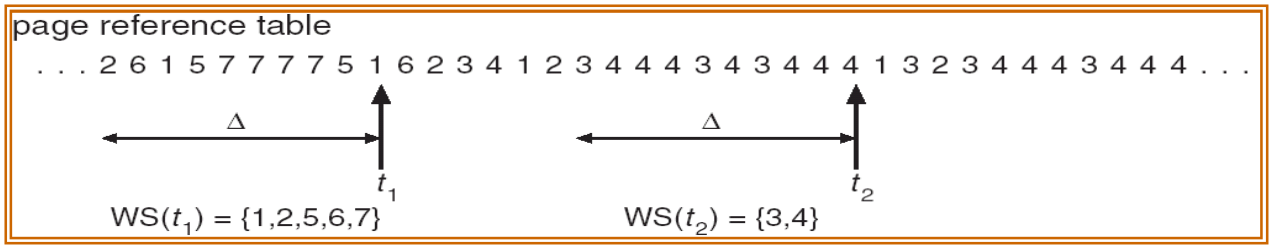


Fig: Working-set model.

- If a page is in active use, it will be in the working set.
- If it is no longer being used, it will drop from the working set 6 time units after its last reference.
- Thus, the working set is an approximation of the program's locality.
- For example, given the sequence of memory references. If 6 = 10 memory references, then the working set at time t1 is {1, 2, 5, 6, and 7}.
- By time *t2,* the working set has changed to {3, 4}.
- The accuracy of the working set depends on the selection of 6. If 6 are too small, it will not encompass the entire locality; if 6 are too large, it may overlap several localities

c) Page-Fault Frequency:

- The specific problem is how to prevent thrashing.

- Thrashing has a high page-fault rate. Thus, we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
- We can establish upper and lower bounds on the desired page-fault rate
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame;
- If the page-fault rate falls below the lower limit, we remove a frame from the process.
- Thus, we can directly measure and control the page-fault rate to prevent thrashing.

UNIT-III
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. An address generated by the CPU is commonly referred as _____
2. _____holds smallest legal physical memory address [ ]
   A) Limit register B) Relocation register
   C) Segment offset D)Segment register
3. _____specifies size of the range of addresses [ ]
   A) Limit register B) Relocation register
   C)Segment offset D) Segment register
4. User program never sees the real physical addresses (T/F)
5. _____ is a solution to external fragmentation. [ ]
   A) Compaction B) Blocks C) Thrashing D) frames
6. Breaking physical memory into fixed-sized blocks called [ ]
   A) Pages B) Frames C) Partitions D) None
7. Paging Suffers from _____ [ ]
   A) Internal Fragmentation B) External Fragmentation
   C) Both Internal and External Fragmentation D) None
8. Breaking logical memory into blocks of the same size called_____
9. Segmentation Suffers from _____ [ ]
   A) Internal Fragmentation B) External Fragmentation
   C) Both Internal and External Fragmentation D) None
10. A routine is not loaded until it is called is referred as [ ]
    A) Dynamic loading B) Dynamic linking
    C) Shared libraries D) Dynamic binding
11. Effective Access Time= _____
12. In paging logical address is divided into two parts_____,_____
13. In segmentation logical address is divided into_____,_____

14. Which of the following page replacement algorithms suffers from Belady's anomaly? [ ]

A) FIFO B) LRU C) OPTIMAL D) LFU

15. Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will [ ]

A) always decrease the number of page faults

B) always increase the number of page faults

C) sometimes increase the number of page faults

D) never affect the number of page faults **(GATE-2001)**

16. The optimal page replacement algorithm will select the page that

A) Has not been used for the longest time in the past. [ ]

B) **Will not be used for the longest time in the future.**

C) Has been used least number of times.

D) Has been used most number of times. (**GATE-2002)**

17. A virtual memory system uses First In First Out (FIFO) page replacement policy and allocates a fixed number of frames to a process. Consider the following statements. [ ]

P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.

Q: Some programs do not exhibit locality of reference.

Which one of the following is TRUE?

A) Both P and Q are true, and Q is the reason for P

B) **Both P and Q are true, but Q is not the reason for P.**

C) P is false, but Q is true

D) Both P and Q are false (**GATE-2007)**

18. The essential content(s) in each entry of a page table is / are

A) Virtual page number [ ]

B) **Page frame number**

C) Both virtual page and page frame number

D) Access right information **(GATE-2009)**

19. Dirty bit for a page in a page table [ ]

A) helps avoid unnecessary writes on a paging device

B) helps maintain LRU information

C) allows only read on a page

D) None of the above **(ISRO 2015)**

20. Consider a 32-bit machine where four-level paging scheme is used. If the hit ratio to TLB is 98%, and it takes 20 nanosecond to search the TLB and 100 nanoseconds to access the main memory what is effective memory access time in nanoseconds? [ ]

A) 126 B) 128 C) 122 D) 120

(ISRO 2011)

21. A page fault [ ]

A) Occurs when a program accesses an available page on memory

B) is an error in a specific page

C) is a reference to a page belonging to another program

D) occurs when a program accesses a page not currently in memory

(ISRO2009)

22. The page replacement algorithm which gives the lowest page fault rate is

A) LRU B) FIFO C) Optimal page replacement D) Second chance algorithm

(ISRO 2008)

23. Which of the following statements are true? [ ]

a) External Fragmentation exists when there is enough total memory space to satisfy a request but the available space is contiguous.

b) Memory Fragmentation can be internal as well as external.

c) One solution to external Fragmentation is compaction. **(NET 2018)**

A) (a) and (b) only B) (a) and (c) only

C) (b) and (c) only D) (a), (b) and (c)

24. Consider the following segment table in segmentation scheme:

Segment	Base	Limit
0	200	200
1	500	12510
2	1527	498
3	2500	50

What happens if the logical address requested is -Segment Id 2 and offset 1000? [ ]

A) Fetches the entry at the physical address 2527 for segment Id2

B) A trap is generated

C) Deadlock

D) Fetches the entry at offset 27 in Segment Id 3 **(ISRO2015)**

SECTION-B

SUBJECTIVE QUESTIONS

1. Compare need of swap-in and swap-out operations?
2. Explain about MVT and MFT in detail?
3. Briefly explain the concept of contiguous memory allocation.
4. Classify two Counting-Based page replacement algorithms.
5. Explain paging scheme for memory management, discuss the paging hardware and paging model.
6. Differentiate Internal and External fragmentation.
7. With a neat diagram explain how segmentation works?
8. What is the necessity of Demand Paging?
9. Illustrate the concepts of demand paging? Why it is called as lazy swappers?
10. Demonstrate in detail Copy-on-Write technique?
11. Summarize various page replacement algorithms?
    a) FIFO b) LRU c) LFU d) OPTIMAL
12. Define thrashing. Explain working set window model to handle thrashing problem.
13. Compare and Contrast First Fit, Best Fit and Worst Fit.
14. Illustrate the concept of Segmentation with neat Sketch.

Problems:

1. Find the number of page faults in FIFO and LRU page replacement algorithms for the following reference string;
   7 0 2 1 3 4 2 1 0 2 1 4 3 2 1 0 0 1 2 1 (no. of frames=3)
2. Make use of the reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1.** Identify number of page faults using (Assume that there are 3 page frames which are initially empty) LRU, Optimal page replacement algorithms.

3. Make use of the reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1.** Identify number of page faults using FIFO page replacement algorithm. Assume that there are 3 page frames which are initially empty.

4. Explain Optimal page replacement algorithm. Apply the same to find out page faults for the reference string **1,2,3,4,5,3,2,1,6,7,8,7,6,9,1,2,4,3,5** by assuming frame size as 4.

5. Consider the following reference **1,2,3,4,5,3,2,1,6,7,8,7,6,9,1,2,4,3,5** String, How many Page Faults would occur for LRU and FIFO Page Replacement Algorithms for frame size of 3.

6. Consider a logical address space of 8 pages of 1024 words mapped into memory of 32 frames. How many bits are there in the logical address?

7. Consider the following page reference string : 1**, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6**. Which of the following options, gives the correct number of page faults related to LRU, FIFO, and optimal page replacement algorithms respectively, assuming 05 page frames and all frames are initially empty ?

8. A computer has 16 pages of virtual address space but the size of main memory is only four frames. Initially the memory is empty. A program references the virtual pages in the order **0, 2, 4, 5, 2, 4, 3, 11, 2, 10.** How many page faults occur if LRU page replacement algorithm is used?

9. Consider a virtual page reference string **1, 2, 3, 2, 4, 2, 5, 2, 3, 4.** Suppose LRU page replacement algorithm is implemented with 3 page frames in main memory. Then the number of page faults are___.

10. A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below? **4, 7, 6, 1, 7, 6, 1, 2, 7, 2**

SECTION-C
QUESTIONS AT THE LEVEL OF GATE

1. Suppose that the virtual Address space has eight pages and physical memory with four page frames. If LRU page replacement algorithm is used, _____ number of page faults occur with the reference string.0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1 [ ]
A) 13 B) 12 C) 11 D) 10 **(NET 2016)**

2. Consider the data given in above question. Least Recently Used (LRU) page replacement policy is a practical approximation to optimal page replacement. For the reference string 1, 2, 1, 3, 7, 4, 5, 6, 3, 1, how many more page faults occur with LRU than with the optimal page replacement policy? [ ]
A) 0 B) 1 C) 2 D) 3 **(GATE 2017)**

3. Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process? [ ]
A) 200 KB and 300 KB B) 200 KB and 250 KB
C) 250 KB and 300 KB D) 300 KB and 400 KB **(GATE 2015)**

4. Assume that there are 3 page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____. [ ]
A) 5 B) 6 C) 7 D) 8 **(GATE 2014)**

5. Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then [ ]
A) OPTIMAL < LRU < FIFO B) OPTIMAL < FIFO < LRU

C) OPTIMAL=LRU D) OPTIMAL=FIFO **(GATE 2012)**

6. Assume that a main memory with only 4 pages, each of 16 bytes, is initially empty. The CPU generates the following sequence of virtual addresses and uses the Least Recently Used (LRU) page replacement policy. 0, 4, 8, 20, 24, 36, 44, 12, 68, 72, 80, 84, 28, 32, 88, 92. How many page faults does this sequence cause? What are the page numbers of the pages present in the main memory at the end of the sequence? [ ]

A) 6 and 1, 2, 3, 4 B) 7 and 1, 2, 4, 5

C) 8 and 1, 2, 4, 5 D) 9 and 1, 2, 3, 5 **GATE2008)**

7. A process has been allocated 3 page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, 1. If optimal page replacement policy is used, how many page faults occur for the above reference string? [ ]

A) 7 B) 8 C) 9 D) 10 **(GATE-2007)**

8. Consider a fully associative cache with 8 cache blocks (numbered 0-7) and the following sequence of memory block requests: 4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7. If LRU replacement policy is used, which cache block will have memory block 7? [ ]

A) 4 B) 5 C) 6 D) 7 **(GATE 2004)**

Unit – IV

Deadlocks and Mass-storage structure

Objectives:

- Students will be able to know the problems of deadlock and study the various avoidance mechanisms

Syllabus: Deadlocks and Mass-storage structure

Deadlocks-

- System model
- Deadlock characterization: Necessary conditions, Resource-Allocation Graph
- Methods for handling deadlocks:
  - deadlock- prevention
  - Avoidance: Safe state, Resource-Allocation-Graph, Banker's Algorithm
  - Detection: single instance of each resource type, several instances of a resource type
  - Recovery
    - process termination
    - resource pre-emption

Mass-storage structure- Overview (Magnetic disks, Magnetic tapes), Disk Scheduling (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK Scheduling), Disk Management (Disk Formatting, Boot blocks, Bad blocks).

Outcomes:

Students will be able to

- Develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- Present a number of different methods for preventing or avoiding deadlocks in a computer system.

Learning Material

4.1 System Model:

Definition of Deadlock:

- In a multiprogramming environment several processors may compete for a finite number of resources.
- A process requests resources and are not available at that time.
- So the process enters into waiting state sometimes the waiting process can never change its state.
- This situation is called a deadlock.

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

- **Request:**
  - The process requests the resource.
  - If the request cannot be granted immediately. Then the requesting process must wait until it can acquire the resource.
- **Use:**
  - The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release:** The process releases the resource.
- The resources may be either physical resources (for example, printers, tape drives, memory space, and CPU cycles) or logical resources (for example, files, semaphores, and monitors).

Deadlock Examples:

Example 1:

- Consider a system with three CD RW drives.
- Suppose each of three processes holds one of these CD RW drives.

- If each process now requests another drive, the three processes will be in a deadlocked state.

Example 2:

- Consider a system with one printer and one DVD drive.
- Suppose that process P_i is holding the DVD and process P_j is holding the printer.
- If P_i requests the printer and P_j requests the DVD drive, a deadlock occurs.

4.2 Deadlock Characterization

Necessary Conditions:

There are four conditions that are necessary for the occurrence of a deadlock:

1. **Mutual Exclusion:**
   - At least one resource must be held in a non-sharable mode;
   - If any other process requests this resource, then that process must wait for the resource to be released.
2. **Hold and Wait:**
   - A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.
3. **No preemption:**
   - Once a process is holding a resource (i.e. once its request has been granted), then that resource cannot be taken away from that process until the process voluntarily releases it.
4. **Circular Wait:**
   - A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[i] is waiting for P[(i + 1) % (N + 1)].

Resource-Allocation Graph:

- In some cases deadlocks can be understood more clearly through the use of Resource Allocation Graphs.
- RAG contains the following properties:

- A set of resource categories, { R1, R2, R3, . . ., RN }, which appear as square nodes on the graph.
- Dots inside the resource nodes indicate specific instances of the resource. (E.g. two dots might represent two laser printers.)
- A set of processes, { P1, P2, P3, . . ., PN }
- **Request Edge:**
  - A set of directed arcs from Pi to Rj, indicating that process Pi has requested Rj, and is currently waiting for that resource to become available.
- **Assignment Edge:**
  - A set of directed arcs from Rj to Pi indicating that resource Rj has been allocated to process Pi, and that Pi is currently holding resource Rj.
  - Note that a request edge can be converted into an assignment edge by reversing the direction of the arc when the request is granted.

Note:

- If a resource allocation graph contains no cycles, then the system is not deadlocked.
- If a resource allocation graph does contain **cycle**s AND each resource category contains only a **single instance**, then a **deadlock exists**.
- If a resource category contains **more than one instance**, then the presence of a **cycle** in the resource allocation graph indicates the possibility of a **deadlock**, **but does not guarantee** one.

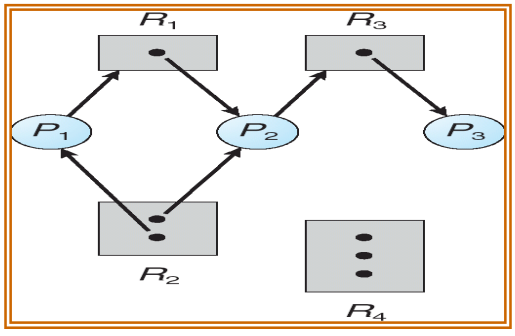


Fig: Resource allocation graph

- The content of above resource-allocation graph is represented as follows:
  - o The sets *P,* R and *E:*
    - ➢ *P* == {P1, P2, P3}
    - ➢ R== {R1, R2, R3}
    - ➢ *E* == {P_I→ R_I, p_2→ R_3, R_I→ p_2, R_2→p_2, R_2→P_I, R_3→ P_3}
  - o Resource instances:
    - ➢ One instance of resource type R_1
    - ➢ Two instances of resource type R_2
    - ➢ One instance of resource type R_3
    - ➢ Three instances of resource typeR_4
  - o Process states:
    - ➢ Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
    - ➢ Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3.
    - ➢ Process P_3 is holding an instance of R_3 .

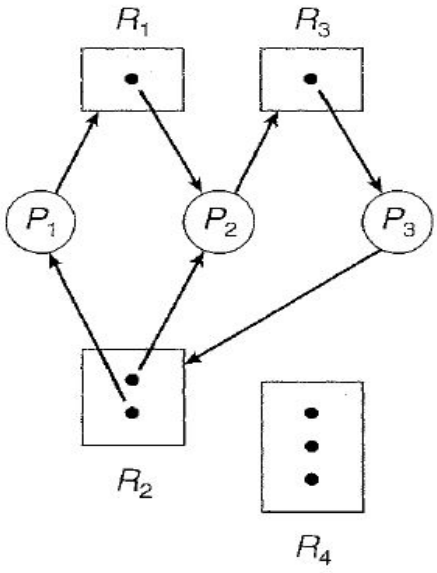


Fig: Resource allocation graph with a deadlock

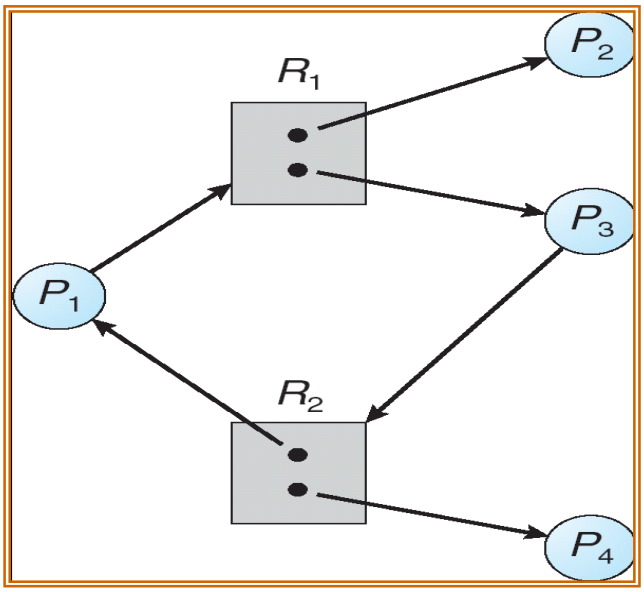


Fig: Resource allocation graph with a cycle but no deadlock

- In this example, we also have a cycle However, there is no deadlock.
- Observe that process *P4* may release its instance of resource type R2.
- That resource can then be allocated to P3, breaking the cycle.
    - *If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is a cycle, then the system may or may not be in a deadlocked state*

4.3 Methods for Handling Deadlocks

Generally, the deadlock problem can be handled in one of three ways:

- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlocked state.
- We can allow the system to enter a deadlocked state, detect it, and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock- Prevention:

- By ensuring that at least one of these conditions cannot hold, we can *prevent* the occurrence of a deadlock.

1. Mutual Exclusion:

- The mutual-exclusion condition must hold for non-sharable resources. For ex: a printer cannot be simultaneously shared by several processes.
- Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock.
- Read-only files are a good example of a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.
- We cannot prevent deadlocks by denying the mutual exclusion condition, because some resources are strictly sharable.

2. Hold and Wait

- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
  - **One protocol** that can be used requires each process to request and be allocated all its resources before it begins execution.
  - **An alternative protocol** allows a process to request resources only when it has none.
    - A process may request some resources and use them.
    - Before it can request any additional resources it must release all the resources that it is currently allocated.

Example:

- To illustrate the difference between these two protocols:
- We consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.

<u>According to Protocol 1:</u>

- If all resources must be requested at the beginning of the process, then the process must initially request the DVD drive, disk file, and printer.

- It will hold the printer for its entire execution, even though it needs the printer only at the end.

According to Protocol 2:

- The process to request initially only the DVD drive and disk file.
- It copies from the DVD drive to the disk and then releases both the DVD drive and the disk file.
- The process must then again request the disk file and the printer.
- After copying the disk file to the printer, it releases these two resources and terminates.

Both these protocols have two main disadvantages:

- First, resource utilization may be low, since resources may be allocated but unused for a long period.
- Second, starvation is possible.
  - A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

3. No Pre-emption

- Pre-emption of process resource allocations can prevent this condition of deadlocks, when it is possible.
- Approach 1:
  - If a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, forcing this process to reacquire the old resources along with the new resources in a single request, similar to the previous discussion.
- Approach 2:
  - When a resource is requested and it is not available, then the system looks to see what other processes currently have those resources *and* are blocked itself and waiting for some other resource.

- If such a process is found, then some of their resources may get pre empted 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.

4. 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.

Deadlock Avoidance:

- The most useful model requires that each process declare the maximum number of each type that it need.
- We can construct an algorithm that ensure the system will never enter into deadlock by giving the priori information about the maximum number of resources of each type that may requested for each process.
- A deadlock avoidance algorithm dynamically check the resource allocation state to ensure that the system never enter into a deadlock.
- The resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

1. Safe State

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.

- A sequence of processes <p_1,p_2,.....p_n> is a safe sequence for all the current allocation state if , for each p_i, the resource that pi can still request can be satisfied by the current available resource plus the resources held by all p_j, with j<i.
- In this situation the resources that process pi needs are not immediately available, and then p_i can wait until all p_j have finished, p_i can obtain all of its needed resources, completed its designated task, and return it's all allocated resources, and terminates.
- When p_i terminates, p_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then system state is said to be unsafe.

Fig: safe, unsafe, and deadlock state space

2. Resource-Allocation Graph Algorithm:

- In addition to request and assignment edges, the other edge is claim edge.

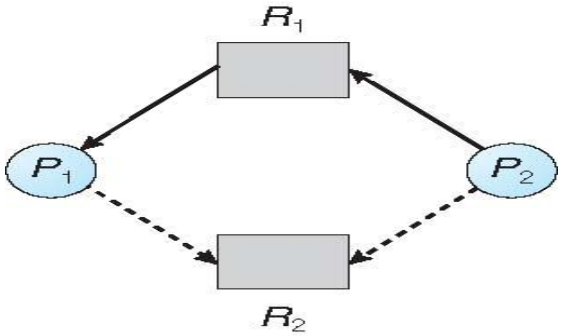


Fig: Resource-allocation graph for deadlock avoidance

- A claim edge P_i->R_j is represents that process pi may request R_j at some time in the future. This claim edge is converted to request edge.
- Similarly, when R_j is released by P_i, the assignment edge R_j->P_i is reconverted to a claim edge P_i->R_j.
- Here note that the resources must be claimed a prior in the system. That is, before process pi starts executing, all its claim edges must appear in resource allocation graph.
- We can relax this condition by allowing a claim edge P_i->R_j to be added to the graph only if all the edges associated with process pi are claim edges.
- Suppose P_i request resource R_j. The request can be granted only if converting the edge P_i->R_j to an assignment edge R_j->P_i does not result in the formation of a cycle in the resource allocation graph.
- If no cycle exists, then the resource allocation will leave the system safely. If cycle is found the system leads to unsafe state.
- From the below graph, P_2 requests R_2. Although R_2 is currently free, we cannot allocate it to P_2. Since this action will create a cycle in the graph.
- A cycle that indicates the system is in an unsafe state. If P_1 request R_2, and P_2 request R_1, then a deadlock will occur.

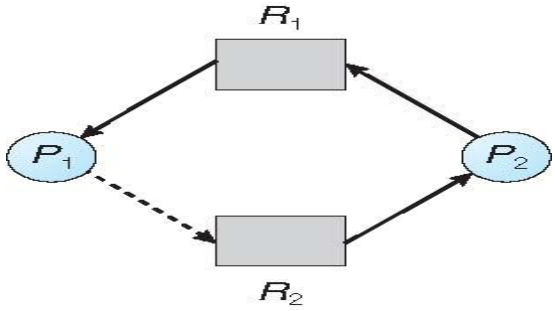


Fig: An unsafe state in a resource-allocation graph

Note: The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type.

3. Banker's Algorithm:

- The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.
- When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need.
- This number may not exceed the total number of resources in the system.
- System has M resources and N processes

Data structures:

- **Available:**
  - A vector of length *m* indicates the number of available resources of each type.
  - If *Available[j]* equals *k,* then *k* instances of resource type *Ri* are available.
- **Max:**
  - An *n* x *m* matrix defines the maximum demand of each process.
  - If *Max[i] [j]* equals *k,* then process P_i may request at most *k* instances of resource type R_j.
- **Allocation:**
  - An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

- If *Allocation[i][j]* equals k, then process P_i is currently allocated k instances of resource type R_j.

- **Need:**
  - An *n* x *m* matrix indicates the remaining resource need of each process.
  - If *Need[i][j]* equals *k,* then process P_i may need *k* more instances of resource type R_j to complete its task. Note that *Need[i][j]* equals *Max[i][j]- Allocation [i][j].*

3.1 Safety Algorithm

➢ This algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

1. Let *Work* and *Finish* be vectors of length *m* and *n,* respectively. Initialize *Work= Available* and *Finish[i] =false* for *i* = 0, 1, ... , *n* - 1.
2. Find an index *i* such that both

   a. *Finish[i] ==false*

   b. $Need_i$ <= *Work*

   If no such *i* exists, go to step 4.

3. Work = Work + Allocationi

   Finish[i] = true

   Go to step 2.

4. If *Finish[i] ==true* for all *i,* then the system is in a safe state.

This algorithm may require an order of *m* x n^2 operations to determine whether a state is safe.

3.2 Resource-Request Algorithm:

- This algorithm for determining whether requests can be safely granted.
- Let $Request_i$ be the request vector for process P_i.
- If $Request_i[j]==k$, then process P_i wants k instances of resource type R_j.
- When a request for resources is made by process P_i, the following actions are taken:

1. If $Request_i <= Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $Request_i <= Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources.
- However, if the new state is unsafe, then P_i must wait for $Request_i$, and the old resource-allocation state is restored.

3.3 An Illustrative Example:

- To illustrate the use of the banker's algorithm, consider a system with
  - Five processes *Po* through *P4*
  - Three resource types *A, B,* and C.
  - Resource type *A* has 10 instances, resource type *B* has 5 instances, and resource type C has 7 instances.
  - Suppose that, at time *T0* , the following snapshot of the system has been taken:

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

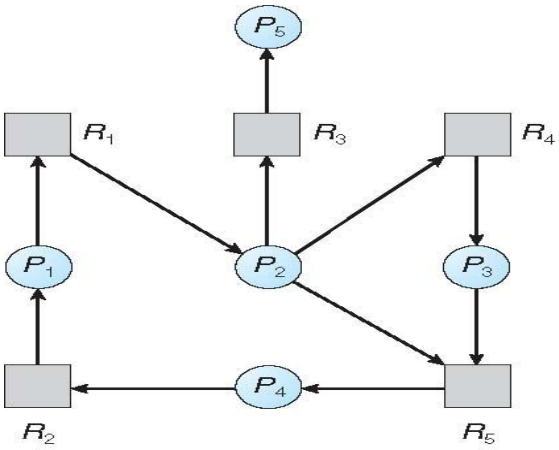
- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

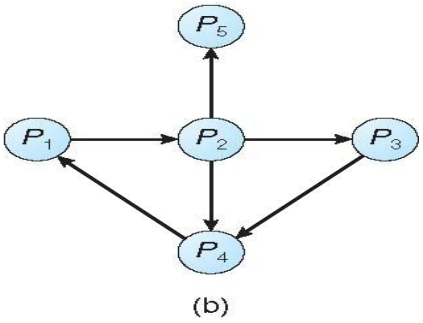
Deadlock Detection:

- If the system does not employ either deadlock prevention or deadlock avoidance algorithms then Deadlock situation may occur.
- In this situation, the system must provide:
  - An algorithm checks the state of the system to determine whether a deadlock has occurred.
  - An algorithm to recover from deadlock.

Single Instance of Each Resource Type

- If all resources are only one single instance, then we define a deadlock detection algorithm that uses wait-for graph.
- We obtain this graph by removing nodes of type resource and collapsing the appropriate edges.
- An edge from pi to P_j in a wait for graph implies that process P_i is waiting for process P_j to release a resource that P_i needs.
- An edge P_i->P_j exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and R_q->P_j for some resource R_q.





Resource- Allocation Graph **Corresponding Wait-for Graph**

- A deadlock exists in the system if and only if the wait-for graph contains a cycle.
- To detect deadlocks, the system needs to maintain the wait-for graph and periodically to invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Several Instances of a Resource Type

- The wait-for graph is not applicable for multiple instances of each resource type.
- The deadlock detection algorithm is applicable for multiple instances of a resource type.
- This algorithm uses several data structures that are similar to banker's algorithm

- **Available:** it indicates the number of available resources of each type.
- **Allocation:** it defines the number of resources of each type currently allocated to each process.
- **Request:** it tells the current request of each process.

Detection Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively Initialize:
   (a) Work = Available
   (b) For i = 1,2, ..., n, if $Allocation_i \neq 0$, then
   Finish[i] = false;otherwise, Finish[i] = true.
2. Find an index i such that both:
   (a) Finish[i] == false
   (b) $Request_i \leq Work$
   If no such i exists, go to step 4.
3. $Work = Work + Allocation_i$
   Finish[i] = true
   go to step 2.
4. If Finish[i] == false, for some i, $1 \leq i < n$, then the system is in deadlock state. Moreover, if Finish[i] == false, then P_i is deadlocked.

Example:

- To illustrate this algorithm, we consider a system with five processes *Po* through *P4* and three resource types *A, B,* and C.
- Resource type *A* has seven instances, resource type *B* has two instances, and resource type C has six instances.
- Suppose that, at time *T0,* we have the following resource-allocation state:

	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	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in Finish[i] = true for all i
- If P_2 requests an additional instance of type C

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

State of system:

- Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.
- Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Detection-Algorithm Usage:

We invoke detection algorithms based on two factors:

1) How often is a deadlock likely to occur?
2) How many processes will be affected by deadlock when it happens?

- If the deadlock occurs frequently, then the detection algorithm should be invoked frequently.
- Resource allocated to deadlocked processes will be idle until the deadlock can be broken.
- Deadlocks occur only when some process makes a request that can't be granted every time.
- We could invoke detection algorithm every time a request for allocation cannot be granted immediately.
- In this case we can identify the process not only the set of processes that is deadlocked but also the specific process that caused the deadlock.
- Invoking deadlock detection algorithm for every request may incur a overhead.

- The alternative is invoking the algorithm at less frequent intervals. For example, once per hour, or whenever CPU utilization drops below 40 percentage.

Recovery from Deadlock:

There are three basic approaches to recovery from deadlock:

1. Inform the system operator, and allow him/her to take manual intervention.
2. Terminate one or more processes involved in the deadlock
3. Preempt resources

1. Process Termination

- Two basic approaches, both of which recover resources allocated to terminate processes:

Abort all deadlocked processes: Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.

Abort one process at a time until the deadlock cycle is eliminated: Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.

- In the second case there are many factors that can go into deciding which processes to terminate next:

1. Process priorities.
2. How long the process has been running, and how close it is to finishing.
3. How many and what type of resources is the process holding.
4. How many more resources does the process need to complete.
5. How many processes will need to be terminated?
6. Whether the process is interactive or batch.
7. Whether or not the process has made non restorable changes to any resource.

2. Resource Pre emption:

- When preempting resources to relieve deadlock,

There are three important issues to be addressed:

Selecting a victim deciding

- Which resources to pre-empt from which processes involves many of the same decision criteria outlined above.

Rollback

- After preemption of resources from a process, what should be done with that process?
- It is missing some needed resource. So, it cannot continue its execution normally.
- We must roll back the process to some safe state and restart it from that state.
- Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only solution is total roll back.
- Abort the process and then restart it, although it is effective to rollback as necessary to break the deadlock.

Starvation

- How do you guarantee that a process won't starve because its resources are constantly being pre empted?
- In a system if selecting a victim is based on cost factor, it may happen that the same process is always picked as a victim.
- We must ensure that a process can be picked as a victim only a finite number of times.
- Solution: include number of rollbacks in cost factor

Mass-Storage Structure

Overview of Mass-Storage Structure

1. Magnetic Disks

Traditional magnetic disks have the following basic structure:

- It provides the bulk of secondary storage.
- Disks are relatively simple
- Each disk platter has a flat circular shape like CD
- Platter diameter ranges from 1.8 to 5.25

- The two surfaces of a platter are covered with magnetic material.
- We store information by recording magnetically on the platters.
- A read-write head flies just above each surface of every platter.
- Heads are attached to a disk arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks
- Tracks are subdivided into sectors.
- The set of arcs that are at one arm position makes up a cylinder.
- The storage capacity of common disk drives is measured in giga bytes.
- When disk is in use a drive motor spins at high speed
- Drives rotate at 60 to 250 times per second
- **Transfer rate** is 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.
- A head crash cannot be repaired; the entire disk must be replaced.
- Disks can be removable
- Drive attached to computer via **I/O bus**
- Busses vary, including
  - **EIDE(Enhanced integrated drive electronics)**,
  - **ATA(Advanced technology attachment)**,
  - **SATA(Serial ATA)**,
  - **USB**,
  - **Fiber Channel**,
  - **SCSI.**
- **Data transfers on a bus are carried out by special electronic processors called controllers.**
- **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array.
- To perform disk I/O operation,

- Computer → sends command to → host controller → sends that command via messages to → disk controller →Operates the disk drive hardware to complete the command

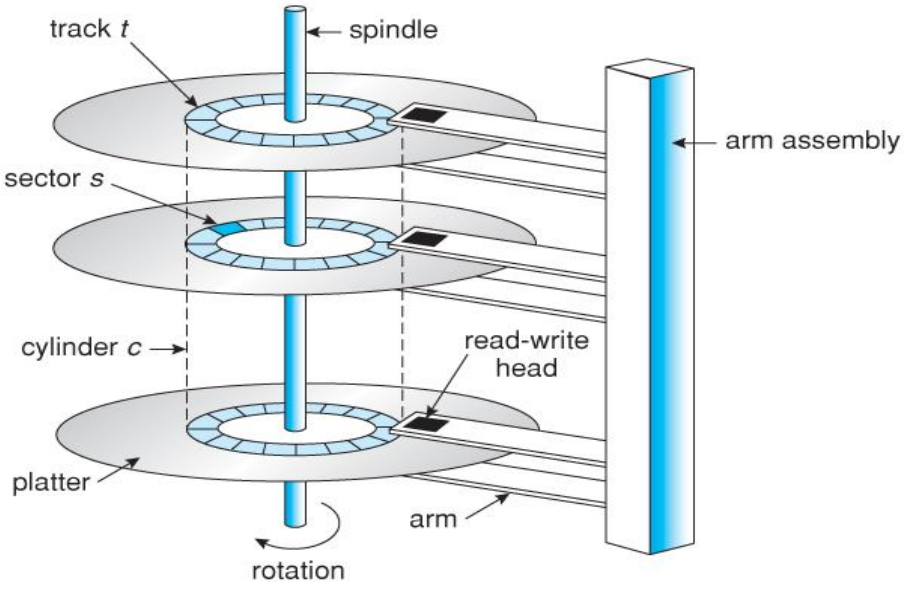


Figure 1.1 - Moving-head disk mechanism

2. Magnetic Tapes

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives.
- These are relatively permanent and holds large quantities of data
- Today these are used primarily for backup, storage of infrequently-used data and acts as a medium for transferring information from one system to another.
- Access time is slow compared to main memory and magnetic disk
- Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives.
- Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

3. Disk Scheduling

- One of the responsibilities of operating system is to use the hardware efficiently.
- For the disk drive meeting this responsibility is having fast access time and large disk bandwidth.
- Access time= Seek time +Rotational Latency
- Disk Bandwidth=Total number of bits transferred, divided by the total time between the first request for service and the completion of the last transfer.
- We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good manner.
- Several algorithms exist to schedule the servicing of disk I/O requests
- We illustrate scheduling algorithms with a request queue (0-19

  Set of requests: 98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53

1) FCFS Scheduling

- ***First-Come First-Serve*** is simple and intrinsically fair, but not very efficient.
- Consider in the following sequence the wild swing from cylinder 122 to 14 and then back to 124:

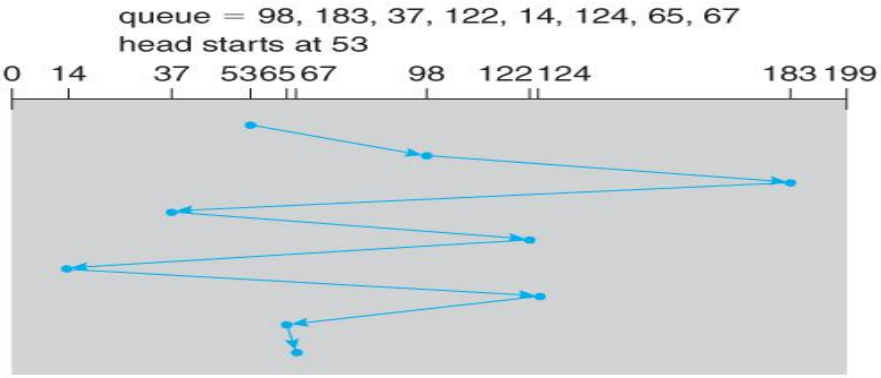


Figure: FCFS disk scheduling.

2) SSTF Scheduling

- ***Shortest Seek Time First*** scheduling is more efficient, but may lead to starvation if a constant stream of requests arrives for the same general area of the disk.
- SSTF reduces the total head movement to 236 cylinders, down from 640 required for the same set of requests under FCFS.

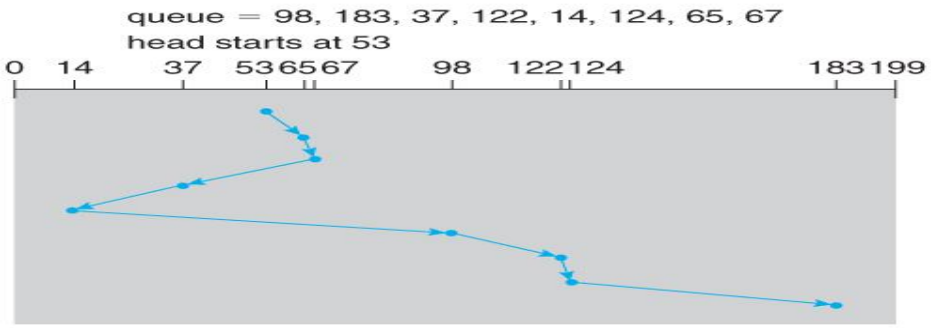


Figure: SSTF disk scheduling.

3) SCAN Scheduling

- The ***SCAN*** algorithm also known as the ***elevator*** algorithm
- 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.

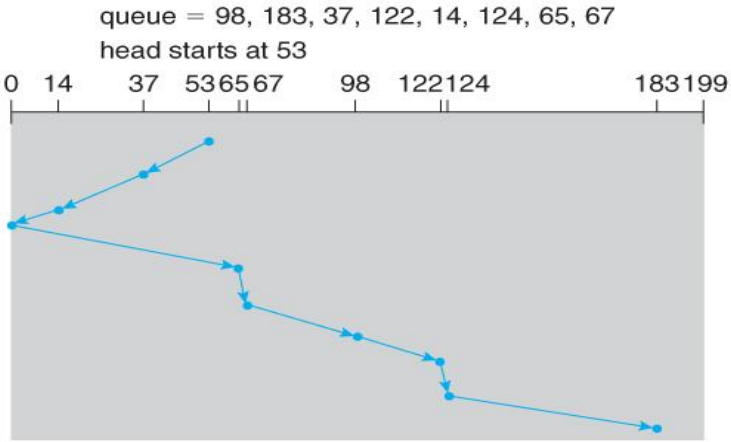


Figure: SCAN disk scheduling.

4) C-SCAN Scheduling

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

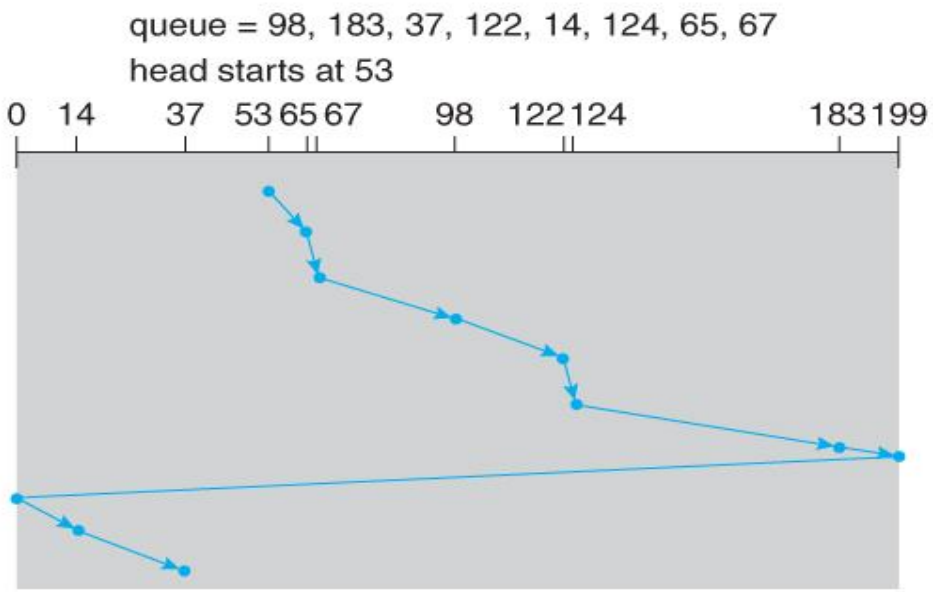


Figure: C-SCAN disk scheduling.

5) LOOK Scheduling

- ***LOOK*** scheduling improves upon SCAN by looking ahead at the queue of pending requests, and not moving the heads any farther towards the end of the disk than is necessary. The following diagram illustrates the circular form of LOOK:

6) C-LOOK Scheduling:

- LOOK a version of SCAN, C-LOOK a 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

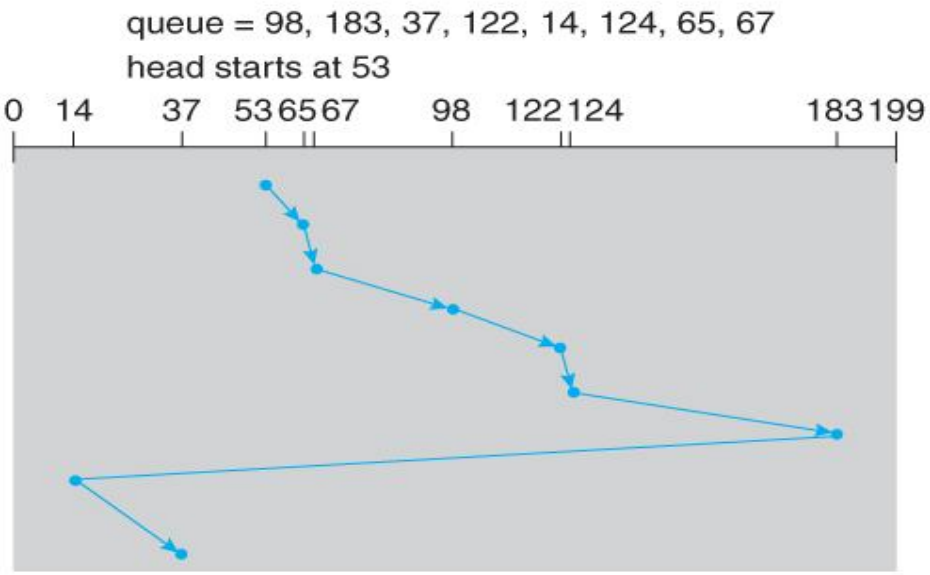


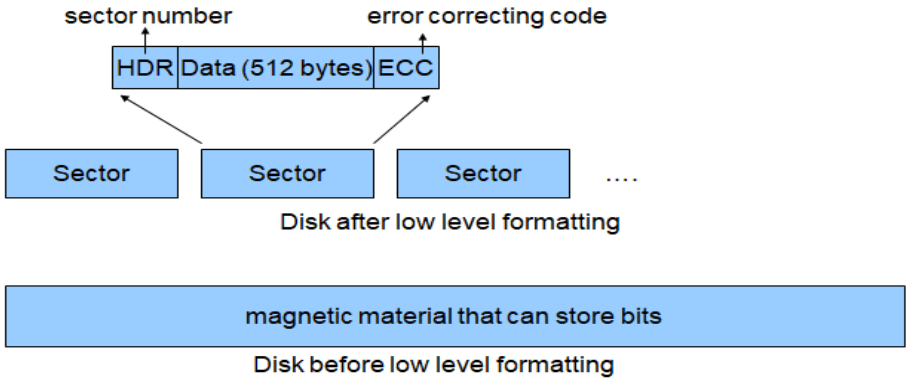
Figure: C-LOOK disk scheduling.

4. Disk Management:

Disk Formatting:

- Before a disk can be used, it has to be *low-level formatted*, which means laying down all of the headers and trailers marking the beginning and ends of each sector.
- **Low-level formatting**, or **physical formatting** — Dividing a new disk into sectors that the disk controller can read and write
  - This formatting fills the disk with a special data structure for each sector
  - Each sector can hold header information + data + Trailer
  - Header and trailer contain information used by the disk controller
    - Sector number
    - ECC (Error correcting code)

Low Level Formatting



- Controller –Write a sector -Update ECC with a value calculated based on data.
- Read a sector- recalculate ECC-compare with the stored value
- If stored value mismatch with recalculated value –data is corrupted in sector-Bad sector
- ECC- contains enough information if a few number of bits are corrupted.
- Controller can identify which bits are changed and calculate their correct values.
- The controller automatically does ECC processing whenever a sector is read or written.

Boot Blocks:

- Computer ROM contains a *bootstrap* program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it.
- The first sector on the hard drive is known as the *Master Boot Record, MBR,* and contains a very small amount of code in addition to the *partition table.*

- The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the *active* or *boot* partition.
- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.
- In a *dual-boot* (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS.
- The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sched, init, etc.), and finally providing one or more login prompts.
- Boot options at this stage may include *single-user* a.k.a. *maintenance* or *safe* modes, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

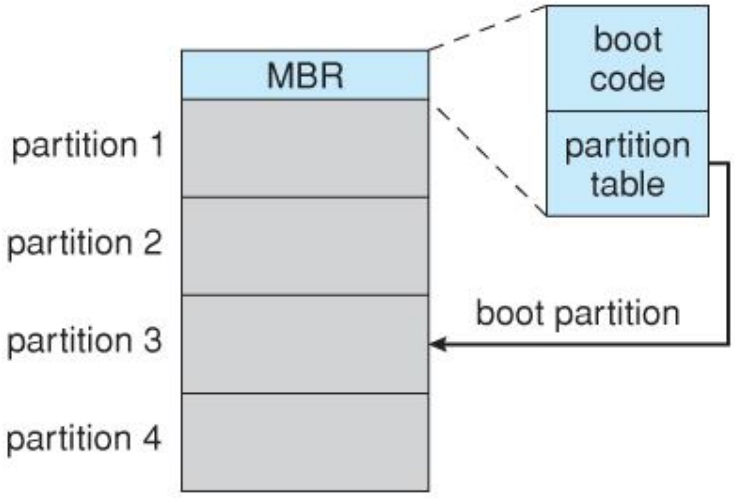


Figure: Booting from disk in Windows 2000.

Bad Blocks:

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time.
- For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time.
- If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.
- In the old days, bad blocks had to be checked for manually.
- Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries.
- Then the bad blocks would be mapped out and taken out of future service.
- Sometimes the data could be recovered, and sometimes it was lost forever.
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered.
- Most disks normally keep a few spare sectors on each cylinder, as well as at least one spare cylinder.
- ***Sector sparing***: Whenever possible a bad sector will be mapped to another sector on the same cylinder, or at least a cylinder as close as possible.
- ***Sector slipping*** may also be performed, in which all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained.
- If the data on a bad block cannot be recovered, then a ***hard error*** has occurred, which requires replacing the file(s) from backups, or rebuilding them from scratch.

UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. A direct edge P_i---->R_j is called a_____ [ ]
   A) Assignment edge C) Request edge
   B) Claim edge D) Release edge
2. A direct edge R_j----> P_i is called a_____ [ ]
   A) Assignment edge C) Request edge
   B) Claim edge D) Release edge
3. Deadlocks can be described in terms of a directed graph called a_____
   A) Directed Acyclic Graph [ ]
   B) Resource allocation graph
   C) Resource request graph
   D) Resource release graph
4. If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred.
   [T/F]
5. If each resource type has exactly several instances, then a cycle does not imply that a deadlock has occurred.
   [T/F]
6. The surface of a platter is logically divided into circular_____[ ]
   A) Sectors B) Tracks C) platters D) surfaces
7. C-SCAN refers to_____ [ ]
   A) Coding SCAN C) Ceil SCAN
   B) Circular SCAN D) City SCAN
8. SCAN algorithm is also called as_____ [ ]
   A) Circular SCAN B) elevator C) LOOK D)
   B) C-LOOK
9. The time to move from the disk arm to the desired cylinder is called_____

A) Rotational latency [ ]

B) Seek time C) Transfer rate D) Random-access time

10. The time for the desired sector to rotate to the disk head is called_____.

A) Rotational latency [ ]

B) Seek time C) Transfer rate D) Random-access time

11. Which one of the following statement about WAIT-FOR graph is true?

A) An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and R_q->P_j for some resource R_q. [ ]

B) An edge Pi->Rj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and R_q->P_j for some resource R_q.

C) An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i-> P_j and R_q->P_j for someresource R_q.

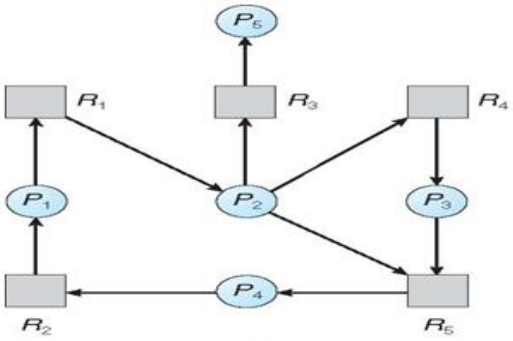
D)An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and P_i ->P_j for some resource R_q.

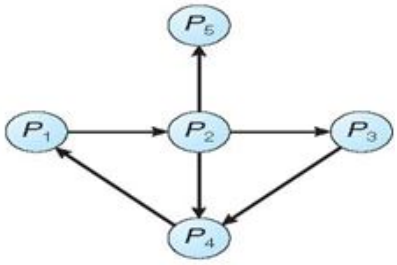
12. Which of the following approaches are used to recover from dead lock

A) Process termination C)Resource preemption

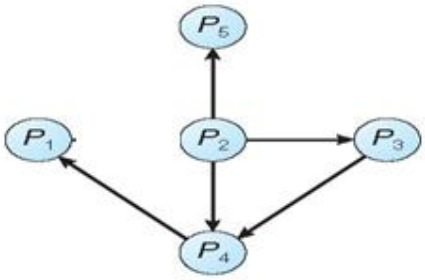
B) Both of the above methods D) None of the above [ ]

13. Which one of the following wait-for graph is equivalent to the given Resource Allocation graph? [ ]

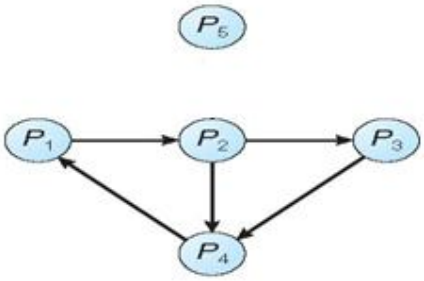




A)



B)



C)

D) No wait-for graph for the given RAG

14. **Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B, C, which have peak time demands of 3, 4, 6 respectively. The minimum value of 'm' that ensures that deadlock will never occur is [ ]**

A) 11 B) 12 C) 13 D) 14

15. Which algorithm of disk scheduling selects the request with the least seek time from the current head positions? [ ]
    A) SSTF scheduling C)FCFS scheduling
    B) SCAN scheduling D) LOOK scheduling
16. The circular wait condition can be prevented by [ ]
    A) Defining a linear ordering of resource types C) Using thread
    B) Using pipes D) All of the mentioned
17. For non sharable resources like a printer, mutual exclusion[ ]
    A) Must exist C) Must not exist
    B) May exist D) None of these
18. The disadvantage of a process being allocated all its resources before beginning its execution is : [ ]
    A) Low CPU utilization C) Low resource utilization
    B) Very high resource utilization D) None of these
19. To ensure no preemption, if a process is holding some resources and requests another resource that cannot be immediately allocated to it : [ ]
    A) Then the process waits for the resources be allocated to it
    B) The process keeps sending requests until the resource is allocated to it
    C) The process resumes execution without the resource being allocated to it
    D) Then all resources currently being held are preempted
20. A system has 12 magnetic tape drives and 3 processes : P0, P1, and P2. Process P0 requires 10 tape drives, P1 requires 4 and P2 requires 9 tape drives. [ ]

Process	Maximum needs	Currently allocated
P0	10	5
P1	4	2
P2	9	2

Which of the following sequence is a safe sequence?

A) P0, P1, P2 C) P1, P2, P0

B) P2, P0, P1 D) P1, P0, P2

21. The content of the matrix Need is : [ ]

A) Allocation – Available C) Max – Available

B) Max – Allocation D) Allocation – Max

22. An edge from process Pi to Pj in a wait for graph indicates that :

A) Pi is waiting for Pj to release a resource that Pi needs. [ ]

B) Pj is waiting for Pi to release a resource that Pj needs.

C) Pi is waiting for Pj to leave the system.

D) Pj is waiting for Pi to leave the system.

23. A computer system has 6 tape drives, with 'n' processes competing for them. Each process may need 3 tape drives. The maximum value of 'n' for which the system is guaranteed to be deadlock free is : [ ]

A) 2 B) 3 C) 4 D) 1

24. A system has 3 processes sharing 4 resources. If each process needs a maximum of 2 units then, deadlock : [ ]

A) Can never occur. C) any occur.

B) Has to occur. D) None of these.

SECTION-B

Descriptive Questions

1. Define deadlock and classify the necessary conditions for deadlock?
2. List and explain different methods used for handling deadlocks?
3. Describe in detail about BANKER'S algorithm?
4. With a neat sketch explain the overview of mass storage structure.
5. Differentiate SCAN, C-SCAN and LOOK, C-LOOK disk scheduling algorithms with an example?
6. What is sector sparing? Explain how it is useful in identifying bad blocks in mass storage?
7. Demonstrate in detail about swap-space management?

Problems:

1. Consider the snapshot of a system processes p1, p2, p3, p4, p5, Resources A, B, C, D
   Allocation[0 0 1 2, 1 0 0 0, 1 3 5 4, 0 6 3 2, 0 0 1 4]
   Max[0 0 1 2, 1 7 5 0, 2 3 5 6, 0 6 5 2, 0 6 5 6]
   Available[1 5 2 0] .
   i. What will be the content of the Need matrix?
   ii. Is the system in safe state? If Yes, then what is the safe sequence?

2. Consider the following and find out the possible resource allocation sequence with the help of deadlock detection algorithm processes p0, p1, p2, p3, p4, Resources A, B, C
   Allocation [0 1 0, 2 0 0 , 3 0 3, 2 1 1, 0 0 2]
   Max[0 0 0, 2 0 2, 0 0 0, 1 0 0, 0 0 2]
   Available[0 0 0].
   i. What will be the content of the Need matrix?
   ii. Is the system in safe state? If Yes, then what is the safe sequence?

3. A computer system uses the Banker's Algorithm to deal with deadlocks. Its current state is shown in the table below, where P0, P1, P2 are processes, and R0, R1, R2 are resources types.

	Maximum Need				**Current Allocation**			**Available**		
	R0	**R1**	**R2**		**R0**	**R1**	**R2**	**R0**	**R1**	**R2**
P0	4	1	2	**P0**	1	0	2	2	2	0
P1	1	5	1	**P1**	0	3	1			
P2	1	2	3	**P2**	1	0	2			

   i. Show that the system can be in safe state?

ii. What will the system do on a request by process P0 for one unit of resource type R1?

4. Four resources ABCD. A has 6 instances, B has 3 instances, C has instances and D has 2 instances.

Process	Allocation ABCD	Max ABCD
P1	3011	4111
P2	0100	0212
P3	1110	4210
P4	1101	1101
P5	0000	2110

i. Is the current state safe?

ii. If P5 requests for (1,0,1,0), can this be granted?

5. Why disk scheduling is needed? Schedule the given requests **98, 183, 37, 122, 14, 124, 65, 67, 10, 150** with the following disk scheduling algorithms and calculate seek time?
   a. FCFS disk scheduling
   b. SSTF disk scheduling
   c. SCAN disk scheduling
   d. C-SCAN disk scheduling
   e. LOOK disk scheduling
   f. C-LOOK disk scheduling

SECTION-C

Previous GATE/NET questions

1. A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is _____

GATE-CS-2014 [ ]

A) 6 B) 7 C) 8 D) 9

2. A system has 6 identical resources and N processes competing for them. Each process can request atmost 2 resources. Which one of the following values of N could lead to a deadlock?**GATE-CS-2015**[ ]

A) 1 B) 2 C) 3 D) 4

3. Considering a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken: **GATE-CS-2014**

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

i. What will be the content of the Need matrix?

ii. Is the system in safe state? If Yes, then what is the safe sequence?

4. An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P0, P1, and P2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the current state:
REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z
REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z
Which one of the following is TRUE? **GATE-CS-2014** [ ]

A) Only REQ1 can be permitted.
B) Only REQ2 can be permitted.
C) Both REQ1 and REQ2 can be permitted.
D) Neither REQ1 nor REQ2 can be permitted

5. Which of the following is NOT a valid deadlock prevention scheme?
GATE CS 2000 [ ]

A) Release all resources before requesting a new resource
B) Number the resources uniquely and never request a lower numbered resource than the last one requested.
C) Never request a resource after releasing any resource
D) Request and all required resources be allocated before execution

Synchronization

Objectives:

- Students will be able to introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data.
- Students will be able to present both software and hardware solutions of the critical-section problem.
- To discuss various inter process communication and synchronization problems.

Syllabus:

The critical section problem, Peterson's solution, synchronization hardware, semaphores, classic problems of synchronization (Bounded-Buffer problem, Readers-Writers problem, Dining-philosophers problem), monitors.

Outcomes:

Students will be able to

- Understand the concepts of critical section problems and its solutions.
- Outline the solutions of critical section problems.
- Develop algorithms for various Inter Process Communication and Synchronization problems

INTRODUCTION:

- Race condition: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
- To prevent race conditions, concurrent processes must be synchronized**.**

1. Critical Section Problem:

Definition: Consider a system consisting of *n* processes {P_0, P_1, …, P_{n-1}}. Each process has a segment of code, called critical section, in which the process may be changing common variables, updating a table, writing a file, and so on.

- The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.
- Each process must request permission to enter its critical section.
- The section of code implementing this request is the **entry section** followed by **exit section;** the remaining code is the **remainder section.**

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Figure: General structure of a typical process p_i.

➢ A solution to the critical-section problem must satisfy the following three requirements:

- **Mutual exclusion:** If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.
- **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
- **Bounded waiting.** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

2. Peterson's solution:

- A classic software-based solution to the critical-section problem known as Peterson's solution.
- Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections.
- The processes are numbered P_0 and P_1.
- For convenience, when presenting P_i, we use P_j to denote the other process; that is, j equals 1 - i.
- Peterson's solution requires the two processes to share two data items:

```
int turn;
boolean flag[2];
```

- The variable turn indicates whose turn it is to enter its critical section.
  - If turn ==i, then process *Pi* is allowed to execute in its critical section.
- The flag array is used to indicate if a process *is ready* to enter its critical section.
  - if flag [i] is true, this value indicates that *Pi* is ready to enter its critical section.
- To enter the critical section, process *Pi* first sets flag [i] to be true and then sets turn to the value j.
- If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time.

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

Figure: The structure of process A in Peterson's solution

- The eventual value of turn determines which of the two processes is allowed to enter its critical section first.
- We now prove that this solution is correct. We need to show that:

1. Mutual exclusion is preserved.

2. The progress requirement is satisfied.

3. The bounded-waiting requirement is met.

- To prove property 1, we note that each P_i enters its critical section only if either flag [j] == false or turn == i. If both processes can be executing in their critical sections at the same time, then flag [0] == flag [1] ==true.
- These two observations imply that *Po* and P1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1 but cannot be both.
- One of the processes say, P_i -must have successfully executed the while statement, whereas P_i had *to* execute at least one additional statement ("turn== j").
- To prove properties 2 and 3, we note that a process *P;* can be prevented from entering the critical section only if it is stuck in the while loop with the condition flag [j] ==true and turn=== j; this loop is the only one possible.
- If P_i is not ready to enter the critical section, then flag [j] ==false, and P_i can enter its critical section. If P_j has set flag [j] to true and is also executing in its while statement, then either turn === i or turn === j. If turn == i, then P_i will enter the critical section.
- If turn== j, then *Pi* will enter the critical section. However, once *Pi* exits its critical section, it will reset flag [j] to false, allowing P_i to enter its critical section.
- If P_i resets flag [j] to true, it must also set turn to i. Thus, since P_i does not change the value of the variable turn while executing the while

statement, P_i will enter the critical section (progress) after at most one entry by P_j (bounded waiting).

3. Synchronization hardware

- By using locks critical section problem is solved.
- Race conditions are prevented by requiring that critical regions be protected by locks.
- That is, a process must acquire a lock before entering a critical section; it releases the lock when it exits the critical section.

```
do {
        acquire lock
                critical section
        release lock
                remainder section
} while (TRUE);
```

Figure: Solution to the critical-section problem using locks.

- If lock= false, then no process is executing in critical section.

a) Test And Set() :

- Whenever a process is ready to enter in the critical section and it calls Test And Set() which sets lock=true.
- Here (critical section process is executing) at this condition if any process want to enter into critical section it should wait until the process executes in critical section.

```
boolean TestAndSet(boolean *target) {
   boolean rv = *target;
   *target = TRUE;
   return rv;
}
```

Figure: The definition of the TestAndSet () instruction.

- If the machine supports the TestAndSet () instruction, then we can implement mutual exclusion by declaring a Boolean variable lock, initialized to false.

- The structure of process P_i is shown below

```
do {
   while (TestAndSet(&lock))
      ; // do nothing

      // critical section

   lock = FALSE;

      // remainder section
} while (TRUE);
```

Figure: Mutual-exclusion implementation with TestAndSet ().

b) Swap () instruction:

- This instruction, in contrast to the TestAndSet () instruction, operates on the contents of two words; the common data structures are
- A global Boolean variable lock is declared and is initialized to false.
- In addition, each process has a local Boolean variable key.

```
void Swap(boolean *a, boolean *b) {
   boolean temp = *a;
   *a = *b;
   *b = temp;
}
```

Figure: The definition of the Swap () instruction

- The structure of process P_i *for* is shown below:

```
do {
   key = TRUE;
   while (key == TRUE)
      Swap(&lock, &key);

      // critical section

   lock = FALSE;

      // remainder section
} while (TRUE);
```

Figure: Mutual-exclusion implementation with the Swap() instruction.

- These algorithms satisfy the mutual-exclusion requirement; they do not satisfy the bounded-waiting requirement.

c) Modified Test and Set(): proving bounded waiting requirement

```
boolean waiting[n];
boolean lock;
```

- These data structures are initialized to false.
- To prove that the mutual exclusion requirement is met, we note that process *P;* can enter its critical section only if either waiting [i] == false or key == false.
- The value of key can become false only if the TestAndSet () is executed.
- The first process to execute the TestAndSet () will find key== false; all others must wait.
- The variable waiting [i] can become false only if another process leaves its critical section; only one waiting [i] is set to false, maintaining the mutual-exclusion requirement.

```
do {
   waiting[i] = TRUE;
   key = TRUE;
   while (waiting[i] && key)
      key = TestAndSet(&lock);
   waiting[i] = FALSE;

      // critical section

   j = (i + 1) % n;
   while ((j != i) && !waiting[j])
      j = (j + 1) % n;

   if (j == i)
      lock = FALSE;
   else
      waiting[j] = FALSE;

      // remainder section
} while (TRUE);
```

Figure: Bounded-waiting mutual exclusion with TestAndSet ().

4. Semaphores:

- Semaphore is nothing but a synchronization tool.
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations:
  - wait () and signal ().
  - The wait () operation was originally termed P (from the Dutch *proberen*, "to test");
  - signal () was originally called V (from *verhogen*, "to increment").
- The definition of wait () is as follows:

```
wait(S) {
    while S <= 0
       ; // no-op
    S--;
}
```

- The definition of signal () is as follows:

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

- When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Usage:

Semaphore is of two types:

1. The value of a **counting semaphore** can range over an unrestricted domain
2. The value of a **binary semaphore** can range only between 0 and 1.
    a. In some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.

- Semaphores are used to solve various synchronization problems.
- For example, consider two concurrently running processes: P_1 with a statement S_1 and P_2 with a statement S_2.
- Suppose we require that S_2 be executed only after S_1 has completed.
- We can implement this scheme readily by letting P_1 and P_2 share a common semaphore synch, initialized to 0, and by inserting the statements in process P_1 and the statements in process P_2.

```
S1;
signal(synch);

wait(synch);
S2;
```

Implementation:

- By using semaphores we have one disadvantage is busy waiting.
- If one process is executing in critical section the other processes waiting outside is known as **busy waiting.**
- Spin **Lock:** wastage of CPU cycles is known as Spin lock
- Solution to busy waiting:
    - Define a semaphore as a record

```
typedef struct
{
        int value;
        struct process *L;
} semaphore;
```

- Assume two simple operations:
  - ✦ block suspends the process that invokes it.
  - ✦ wakeup(P) resumes the execution of a blocked process P.

- Semaphore operations now defined as
  - wait(S)

```
{
        S.value--;
        if (S.value < 0)
        {
                add this process to S.L;
                block();
        }
}
```

  - signal(S)

```
{
        S.value++;
        if (S.value <= 0)
        {
                remove a process P from S.L;
                wakeup(P);
        }
}
```

Deadlocks and Starvation:

- The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
- The event in question is the execution of a signal () When such a state is reached, these processes are said to be deadlocked.
- We consider a system consisting of two processes, P_0 and P_1, each accessing two semaphores, S and Q, set to the value 1:

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
.	.
signal(S);	signal(Q);
signal(Q);	signal(S);

- Suppose that P_0 executes wait (S) and then P_1, executes wait (Q).
- When P_0 executes wait (Q), it must wait until P_1, executes signal (Q).
- Similarly, when P_1, executes wait (S), it must wait until P_0executes signal(S).
- Since these signal () operations cam1ot be executed, P_0 and P_1, are deadlocked.
- We say that a set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set.
- Problem related to deadlocks is **indefinite blocking** or **starvation** a situation in which processes wait indefinitely within the semaphore.

5. Classic problems of synchronization

The Bounded-Buffer Problem:

- We assume that the pool consists of *n* buffers, each capable of holding one item.
- The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
- The empty and full semaphores count the number of empty and full buffers.
- The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.
- The code for the producer process is shown below

```
do {
      . . .
   // produce an item in nextp
      . . .
   wait(empty);
   wait(mutex);
      . . .
   // add nextp to buffer
      . . .
   signal(mutex);
   signal(full);
} while (TRUE);
```

Figure: The structure of the producer process.

- The code for the consumer process is shown below:

```
do {
   wait(full);
   wait(mutex);
      . . .
   // remove an item from buffer to nextc
      . . .
   signal(mutex);
   signal(empty);
      . . .
   // consume the item in nextc
      . . .
} while (TRUE);
```

Figure: The structure of the consumer process

- The producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

Readers-Writers Problem:

- Suppose that a database is to be shared among several concurrent processes.
- Some of these processes may want only to read the database, whereas others may want to update the database.
- These two types of processes are distinguished as readers and writers
- If two readers access the shared data simultaneously, no adverse effects will result.

- If a writer and some other process (either a reader or a writer) access the database then problem arises.
- The information in the shared data is read by a process that processor is known as reader process. This performs in shared lock.
- The writers have exclusive access to the shared database while writing to the database. This synchronization problem is referred to as the *readers-writers problem.*
- In the solution to the first readers-writers problem, the reader processes share the following data structures:

```
semaphore mutex, wrt;
int readcount;
```

- The semaphores mutex and wrt are initialized to 1;
- readcount is initialized to 0.
- The semaphore wrt is common to both reader and writer processes.
- The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated.
- The readcount variable keeps track of how many processes are currently reading the object.
- The semaphore wrt functions as a mutual-exclusion semaphore for the writers.

```
do {
   wait(mutex);
   readcount++;
   if (readcount == 1)
      wait(wrt);
   signal(mutex);
      . . .
   // reading is performed
      . . .
   wait(mutex);
   readcount--;
   if (readcount == 0)
      signal(wrt);
   signal(mutex);
} while (TRUE);
```

Figure: The structure of a reader process.

```
do {
   wait(wrt);
      . . .
   // writing is performed
      . . .
   signal(wrt);
} while (TRUE);
```

Figure: The structure of a writer process

Reader-writer locks are most useful in the following situations:

- In applications where it is easy to identify which processes only read shared data and which processes only write shared data.
- In applications that have more readers than writers. This is because reader writer locks generally require more overhead to establish than semaphores or mutual-exclusion locks. The increased concurrency of allowing multiple readers compensates for the overhead involved in setting up the reader writer lock.

Dining-philosophers problem:

- Consider five philosophers who spend their lives thinking and eating.
- The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher.
- In the center of the table is a bowl of rice, and the table is laid with five single chopsticks

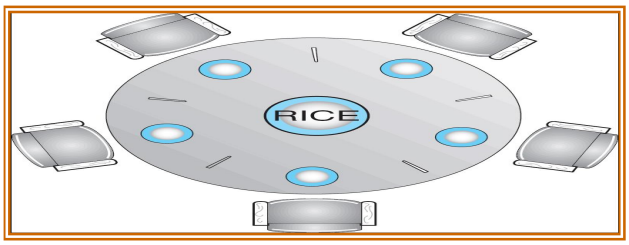


Figure: The situation of the dining philosophers

- When a philosopher thinks, she does not interact with her colleagues.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her.
- *A* philosopher may pick up only one chopstick at a time.
- Obviously, she cannot pick up a chopstick that is already in the hand of a neighbor.
- When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks.
- When she is finished eating, she puts down both of her chopsticks and starts thinking again.
- One simple solution is to represent each chopstick with a semaphore.
- *A* philosopher tries to grab a chopstick by executing a wait () operation on that semaphore;
- she releases her chopsticks by executing the signal () operation on the appropriate semaphores.

semaphore chopstick[5];

- The structure of philosopher i is shown below

```
do {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
      . . .
   // eat
      . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
      . . .
   // think
      . . .
} while (TRUE);
```

Figure: The structure of philosopher *i.*

- Several possible remedies to the deadlock problem are listed next.

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in a critical section).
- Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick

6. Monitors:

- Semaphores provide a convenient and effective mechanism for process synchronization.
- By using them incorrectly can result in timing errors that are difficult to detect.
- These errors happen only if some particular execution sequences take place and these sequences do not always occur.
  - Suppose that a process interchanges the order in which the wait() and signal() operations on the semaphore mutex are executed, resulting in the following execution:

```
signal(mutex);
    ...
  critical section
    ...
wait(mutex);
```

- This sequence violating the mutual-exclusion requirement.
  - Suppose that a process replaces signal (mutex) with wait (mutex). That is, it executes

```
wait(mutex);
    ...
  critical section
    ...
wait(mutex);
```

- In this case, a deadlock will occur.

- Suppose that a process omits the wait (mutex), or the signal (mutex), or both.
- In this case, either mutual exclusion is violated or a deadlock will occur.

Usage:

- A *monitor type* is an ADT which presents a set of programmer-defined operations that are provided mutual exclusion within the monitor.
- The monitor type also contains the declaration of variables whose values define the state of an instance of that type, along with the bodies of procedures or functions that operate on those variables.

```
monitor monitor name
{
    // shared variable declarations
    procedure P1 ( . . . ) {
        . . .
    }
    procedure P2 ( . . . ) {
        . . .
    }
            .
            .
            .
    procedure Pn ( . . . ) {
        . . .
    }
    initialization code ( . . . ) {
        . . .
    }
}
```

Figure: Syntax of a monitor.

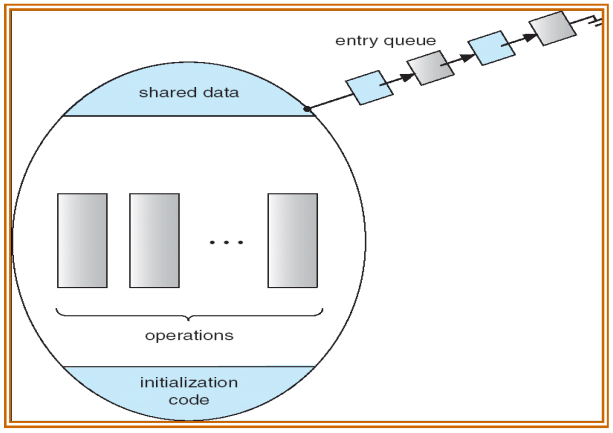


Figure: Schematic view of a monitor.

- A programmer who needs to write a tailor-made synchronization scheme can define one or more variables of type *condition:*

condition x, y;

- The only operations that can be invoked on a condition variable are wait () and signal ().

x. wait();

- The operation means that the process invoking this operation is suspended until another process invokes

x. signal();

- The x. signal () operation resumes exactly one suspended process.

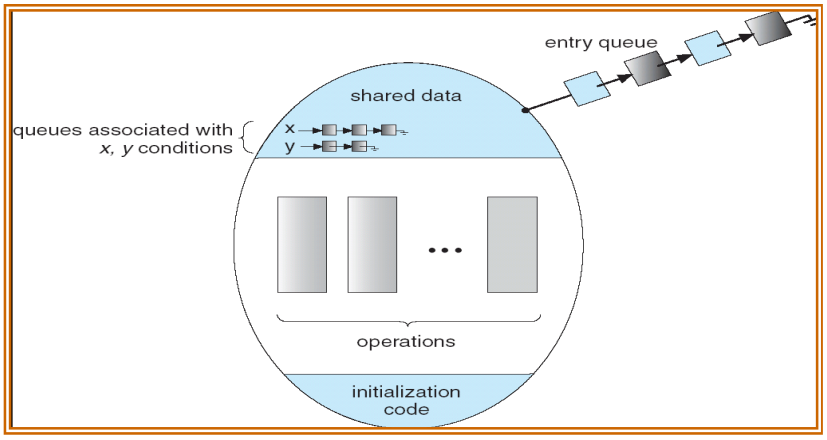


Figure Monitor with condition variables

- Suppose that, when the x. signal () operation is invoked by a process P, there exists a suspended process Q associated with condition x.
- If the suspended process Q is allowed to resume its execution, the signaling process P must wait. Otherwise, both P and Q would be active simultaneously within the monitor.

- Both processes can conceptually continue with their execution. Two possibilities exist:

1. **Signal and wait.** *P* either waits until *Q* leaves the monitor or waits for another condition.
2. **Signal and continue.** *Q* either waits until *P* leaves the monitor or waits for another condition.

Dining-Philosophers Solution Using Monitors:

- Presenting a deadlock-free solution to the dining-philosophers problem.
- This solution imposes the restriction that a philosopher may pick up her chopsticks only if both of them are available.
- Data structure to distinguish among three states in which we may find a philosopher

```
enum {THINKING, HUNGRY, EATING} state[5];
```

Philosopher *i* can set the variable `state[i] = EATING` only if her two neighbors are not eating: `(state[(i+4) % 5] != EATING)` and `(state[(i+1) % 5] != EATING)`.

We also need to declare

```
condition self[5];
```

- Each philosopher, before starting to eat, must invoke the operation pickup().
- After the successful completion of the operation, the philosopher may eat.
- The philosopher invokes the put down() operation.
- Thus, philosopher *i* must invoke the operations pickup() and put down() in the following sequence:

```
DiningPhilosophers.pickup(i);
                ...
                eat
                ...
DiningPhilosophers.putdown(i);
```

```
monitor dp
{
   enum {THINKING, HUNGRY, EATING} state[5];
   condition self[5];

   void pickup(int i) {
      state[i] = HUNGRY;
      test(i);
      if (state[i] != EATING)
         self[i].wait();
   }

   void putdown(int i) {
      state[i] = THINKING;
      test((i + 4) % 5);
      test((i + 1) % 5);
   }

   void test(int i) {
      if ((state[(i + 4) % 5] != EATING) &&
       (state[i] == HUNGRY) &&
       (state[(i + 1) % 5] != EATING)) {
          state[i] = EATING;
          self[i].signal();
      }
   }

   initialization_code() {
      for (int i = 0; i < 5; i++)
         state[i] = THINKING;
    }
}
```

Figure: A monitor solution to the dining-philosopher problem

UNIT-V
Synchronization
Assignment-Cum-Tutorial Questions

A. *Objective Questions*

1. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called [ ]
   a) dynamic condition. b) race condition
   c) essential condition d) critical condition
2. If a process is executing in its critical section, then no other processes can be executing in their critical section. This condition is called [ ]
   a) mutual exclusion b) critical exclusion
   c) synchronous exclusion d) asynchronous exclusion
3. Which one of the following is a synchronization tool? [ ]
   a) thread b) pipe
   c) semaphore d) socket
4. Mutual exclusion can be provided by the [ ]
   a) mutex locks
   b) binary semaphores
   c) both mutex locks and binary semaphores
   d) none of the mentioned
5. To enable a process to wait within the monitor, [ ]
   a) a condition variable must be declared as condition
   b) condition variables must be used as boolean objects
   c) semaphore must be used
   d) all of the mentioned
6. The segment of code in which the process may change common variables, update tables, write into files is known as : [ ]
   a) program b) critical section
   c) non – critical section d)synchronizing

7. The following three conditions must be satisfied to solve the critical section problem : [ ]
   a) Mutual Exclusion b) Progress
   c) Bounded Waiting d)All of the mentioned
8. An un-interruptible unit is known as : [ ]
   a) single b) atomic
   c) static d)none of the mentioned
9. If the semaphore value is negative : [ ]
   a) its magnitude is the number of processes waiting on that semaphore
   b) it is invalid
   c) no operation can be further performed on it until the signal operation is performed on it
   d) none of the mentioned
10. The two kinds of semaphores are : [ ]
   a) mutex & counting b)binary & counting
   c) counting & decimal d)decimal & binary
11. The bounded buffer problem is also known as : [ ]
   a) Readers – Writers problem b) Dining – Philosophers problem
   c) Producer – Consumer problem d)None of the mentioned
12. In the bounded buffer problem, there are the empty and full semaphores that : [ ]
   a) count the number of empty and full buffers
   b) count the number of empty and full memory spaces
   c) count the number of empty and full queues
   d) none of the mentioned
13. To ensure difficulties do not arise in the readers – writers problem, _____ are given exclusive access to the shared object. [ ]
   a) readers b)writers
   c) readers and writers d)none of the mentioned

14. The dining – philosophers problem will occur in case of :
    a) 5 philosophers and 5 chopsticks [ ]
    b) 4 philosophers and 5 chopsticks
    c) 3 philosophers and 5 chopsticks
    d) 6 philosophers and 5 chopsticks
15. All processes share a semaphore variable mutex, initialized to 1. Each process must execute wait(mutex) before entering the critical section and signal(mutex) afterward.

    Suppose a process executes in the following manner :

    signal(mutex);

    .....

    critical section

    .....

    wait(mutex);

    In this situation :

    a) a deadlock will occur [ ]
    b) processes will starve to enter critical section
    c) several processes maybe executing in their critical section
    d) all of the mentioned
16. A monitor is characterized by :
    a) a set of programmer defined operators [ ]
    b) an identifier
    c) the number of variables in it
    d) all of the mentioned
17. The monitor construct ensures that : [ ]
    a) only one process can be active at a time within the monitor
    b) n number of processes can be active at a time within the monitor (n being greater than 1)
    c) the queue has only one process in it at a time
    d) all of the mentioned
18. The operations that can be invoked on a condition variable are :
    a) wait & signal b) hold & wait [ ]

c) signal & hold d) continue & signal

19. A monitor is a module that encapsulates

a) shared data structures [ ]

b) procedures that operate on shared data structure

c) synchronization between concurrent procedure invocation

d) all of the mentioned

20. To enable a process to wait within the monitor, [ ]

a) a condition variable must be declared as condition

b) condition variables must be used as boolean objects

c) semaphore must be used

d) all of the mentioned

21. Mutual Exclusion can be provided by the [ ]

a) Mutex Locks b) Binary Semaphores

c) Both (a) and (b) d) None of the Mentioned

22. Process Synchronization can be done on [ ]

a) Hardware Level b)Software Level

c) Both (a) and (b) d) None of the mentioned

SECTION-B

Descriptive Questions

1. Prove that the Peterson's Solution for critical section problem is correct with the help of flag and turn variables.
2. Discuss hardware instructions used for process synchronization.
3. Define the instructions, test and set ( ) and swap ( )
4. Explain about Synchronization Hardware.
5. What is a semaphore? What are its operations?
6. What is a Critical Section Problem? Write any two classic problems of Synchronization.
7. What is Readers-Writers problem? How it can be considered as synchronization problem? Explain its solution with Mutex locks.
8. Explain in detail how monitors are used to solve the Dining-Philosopher problem.

9. How can we use Monitors in Synchronization?
10. What is a bounded-buffer problem? Explain its solution using mutex locks.
11. Explain about solution to Dining-philosophers problem using wait() and signal() operations?

SECTION-C

Previous GATE/NET questions

1. A critical section is a program segment **GATE-1996**[ ]

a) which should run in a certain specified amount of time
b) which avoids deadlocks
c) where shared resources are accessed
d) which must be enclosed by a pair of semaphore operations, P and V

2. A solution to the Dining Philosophers Problem which avoids deadlock is

a) ensure that all philosophers pick up the left fork before the right fork
b) ensure that all philosophers pick up the right fork before the left fork
c) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork
d) None of the above **GATE-1996 [ ]**

3. Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared boolean variables S1 and S2 are randomly assigned.

Method Used by P1 **GATE-2010**

Method Used by P2

while (S1 == S2) ;

Critica1 Section

S1 = S2;

while (S1 != S2) ;

Critica1 Section

S2 = not (S1);

Which one of the following statements describes the properties achieved? [ ]

a) Mutual exclusion but not progress
b) Progress but not mutual exclusion
c) Neither mutual exclusion nor progress
d) Both mutual exclusion and progress

4. A counting semaphore was initialized to 10. Then 6 P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is**GATE-1998**

[ ]

a) 0 b)8 c)10 d)12

5. Let m[0]...m[4] be mutexes (binary semaphores) and P[0] .... P[4] be processes. **GATE-2000**

Suppose each process P[i] executes the following:

wait (m[i]);wait (m[(i+1) mode 4]);

.....

release (m[i]); release (m[(i+1)mod 4]);

This could cause [ ]

a) Thrashing
b) Deadlock
c) Starvation, but not deadlock
d) None of the above

6. The enter_CS() and leave_CS() functions to implement critical section of a process are realized using test-and-set instruction as follows:**GATE-2009**

```
void enter_CS(X)
{
while test-and-set(X) ;
}
void leave_CS(X)
{
X = 0;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements:

I. The above solution to CS problem is deadlock-free

II. The solution is starvation free.

III. The processes enter CS in FIFO order.

IV More than one process can enter CS at the same time.

Which of the above statements is TRUE? [ ]

a) I only
b) I and II
c) II and III
d) IV only

7. The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0=1, S1=0, S2=0.

Process P0	Process P1	Process P2

while (true)
{
wait (S0);
print (0);
release (S1);
release (S2);
}
	wait (S1);
	Release
	(S0);
		wait (S2);
		release (S0);

How many times will process P0 print '0'? **GATE-2010** [ ]

a) At least twice b)Exactly twice

c) Exactly thrice d)Exactly once

8. Fetch_And_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available. **GATE-2012**

```
AcquireLock(L){
while (Fetch_And_Add(L,1))
        L = 1;
}
 ReleaseLock(L){
                L = 0;
 }
```

This implementation [ ]

a) fails as L can overflow

b) fails as L can take on a non-zero value when the lock is actually available.

c) works correctly but may starve some processes

d) works correctly without starvation

UNIT-VI

File system Interface

Objectives:

Students will be able

- To explain concepts of a file
- To discuss file access methods, file sharing, and directory structures
- To explore file-system protection

Syllabus: File system Interface

Concept of a file (File attributes, file operations), Access Methods (Sequential access, direct access), Directory structure (overview, single-level, two-level, tree structured, acyclic-graph), File system mounting, files sharing (multiple users, remote file systems) and protection

Outcomes:

Students will be able to

- Understand about file operations, file attributes.
- Know the file structure and directory structure.
- Learn about various types of directories and file sharing.

Concept of File

- A file is a collection of related information that is recorded on secondary storage.
- A file is a smallest allotment of logical secondary storage that is data can't be written to secondary storage unless they are within a file.
- File represent programs (both source and object forms) and data.
- Data file may be numeric, alphabetic, alphanumerical, or binary.
- A file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by file's creator and user.
- The information of a file is defined by its creator.
- Many different type of information may be stored in file.

➢ Examples: source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recording and so on.

1. File Attributes:

- **Name:** The symbolic file name is the only information kept in human readable code.
- **Identifier**: This is unique tag, usually a number, identifies the file within the file system. It is the non human readable name for the file.
- **Type**: This information is needed for those systems that support different type.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** the current size of the file and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time & Date:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security and usage monitoring.

2. File Operations:

A file is an abstract data type. Operating system must do for each of the six basic file operations.

- **Creating a file:** Two steps are required to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** to write a file, we make a system call specifying both the name of the file and the information to be written to the file.
- **Reading a file:** to read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. Again, the directory is searched for the associated directory entry, and the system needs to keep a read pointer to the location in the file

where the next read is take place. Once the read has taken place, the read pointer is updated.

- **Repositioning with in a file:** The directory is searched for the appropriate entry, and the current file position is set to a given value.
- **Deleting a file:** whatever files want to delete from directory, those files have to found in directory. If file is found then it will be deleted. Deleted file space used for store the next file.
- **Truncating a file:** The file attributes are not changed but the content of file deleted.
- There are several issues are associated with an open file.

❖ **File pointer:** it is a unique pointer for each process operating on file.

❖ **File open count:** how many times has the current file has been opened and not yet closed. When this counter reaches zero the file can be removed from the table.

❖ **Disk location of the file:** most file operations required to modify data within a file.

❖ **Access right:** Each process can open a file in access mode. This information stored on pre process table. So that the operating system can allow or deny subsequent I/O operations.

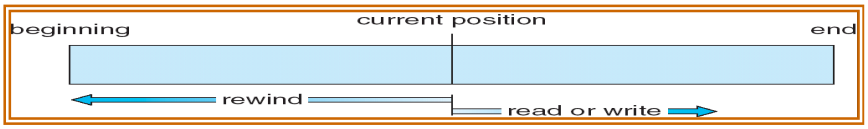
File Access Methods

Files contain information. When it is used, this information must be accessed and read into computer memory. The information of a file can be accessed in different ways.

1. Sequential Access:

- It is the simplest access method and it is sequential.
- Information is processed one after another in sequential.
- This mode of access is common and used in editors and compilers.
- The general operations on files are read and write.
- A read operation reads the next portion of the file and automatically advances the file pointer, which tracks the I/O location.

- A write operation appends to end of the file and advances to the end of newly written material.



2. Direct Access:

- Direct access method is also called as relative access.
- A file is made up to a fixed size logical records that allow programs read and write records in any order.
- Databases and airline reservations are example for this mode.
- The direct access is based on the disk model of a file since disk allows random access to any file block.
- For direct access, the file is viewed as a numbered sequence of block or record.
- Thus, we may read block 14 then block 59 and then we can write block 17.
- There is no restriction on the order of reading and writing for direct access file.
- A block number provided by the user to the operating system is normally a relative block number, the first relative block of the file is 0 and then 1 and so on.

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

3. Other Access Methods:

- Other access methods can be built on top of a direct access method.
- These methods constructs index for files.

- This index containing pointer to the various blocks.
- With large files, the index file itself may become too large to be kept in memory.
- There is a solution to create an index for the index file.
- The primary index file would contain pointer to secondary index files, which would point to the actual data items.
- Example: Indexed sequential access method(ISAM)

Directory Structure

- The file systems are extensive in computers.
- Some systems store some millions of files on disks.
- To manage all these data, we need to organize them in two parts.
- **First,** disks are split into one or more partitions also known as mini disks.
    - Each disk on system contains at least one partition, which is low level structure in which files and directories reside.
- **Second**, each partition contains information about files within it.
    - This information is kept in entries in a device directory or volume table of contents.
    - The device directory records information such as name, location, size and type for al files on this partition.

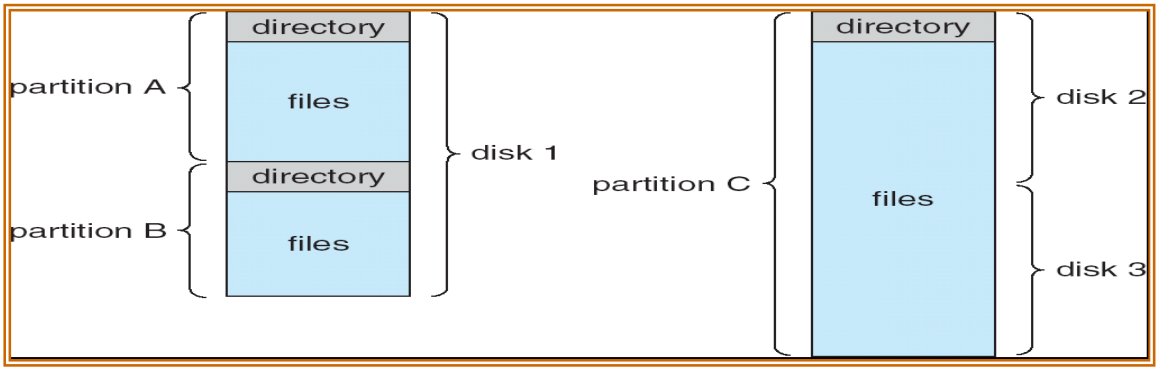


Fig: A typical file system organization

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory.

- **Search for a file:** we need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names and similar names may indicate a relationship between files, we may want to be able to find all files whose names match a particular pattern.
- **Create a file:** new files needed to be crated and added to the directory.
- **Delete a file:** when a file is not needed then it removed from directory.
- **List a directory:** we need to be able to list the files in a directory, and the contents of the directory entry for each file in a list.
- **Rename a file:** the name of a file represents its content to its user, the name must be changeable when the contents or use of the file changes.
- **Traverse the file system:** In file system, every file and every directory accessed by the user. Save files content and its structure at each regular interval time. if we are saving like regular interval times then it is easy to backup in case of system failure.

1 Single-Level Directory

- All files are contained in single directory.
- It has a limitation when number of users is more than one.
- In single level directory all file names are unique.
- If two users call their data file name as test, then unique-name rule is violated.

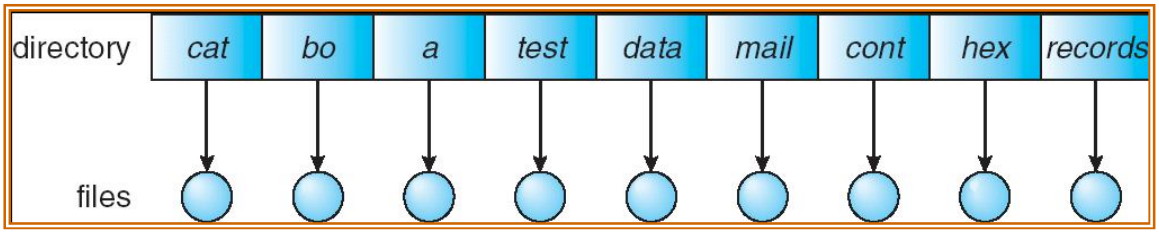


Fig: Single-level directory.

2 Two-Level Directory

- In single-level directory has a problem if two files are same name.
- To overcome the above problem use Two-level directory for each user.
- In Two-level directory structure, each user has own directory called as user file directory (UFD).
- Each UFD has a similar structure but list of files are different from one UFD to other UFD.
- When a user job is started, then the system searches in Master File Directory (MFD).
- In MFD each user has their name and account number.
- When a user wants to refer a particular file then he must search in his own directory or UFD.
- If a user create or delete files then those operations are done from their local UFD's.

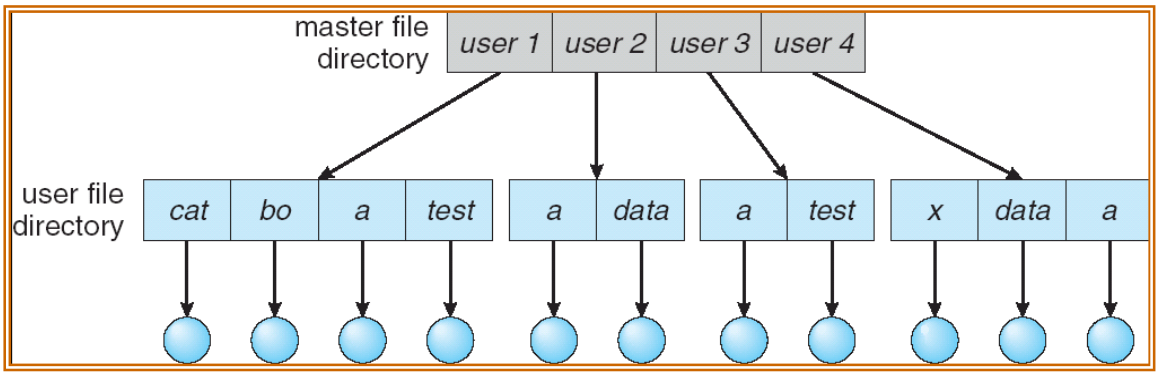


Fig: Two-level Directory Structure

3 Tree-Structured Directories

- The directory structure is a tree with arbitrary heights.
- The tree has a root directory and every file path is unique.
- A directory contains set of files or subdirectories.
- The internal format of each directory is same.
- We are using bits to represent files and directories.
- '1' represents for directories and '0' represents file.
- Tree structure directories are very efficient in search operation.

- An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory.
- Each file has specific path. These paths are divided into two parts. First one is absolute path, second is relative path.
- Absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A relative path name defines a path from the current directory.
- An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory.
- If a directory is empty, its entry in its containing directory can simply be deleted.
- Suppose the directory to be deleted is not empty, but contains several files or subdirectories then one of two approaches can be taken.
- Some systems, such as MS-DOS, will not delete a directory unless it is empty.
- Thus, to delete a directory, the user must delete all the files in that directory.
- if any subdirectories exist , this procedure must be applied recursively to them, so that they can be deleted.
- The other approach taken by UNIX rm command is to provide the option that, when a request is made to delete a directory, that entire directory's files and subdirectories are also to be deleted.

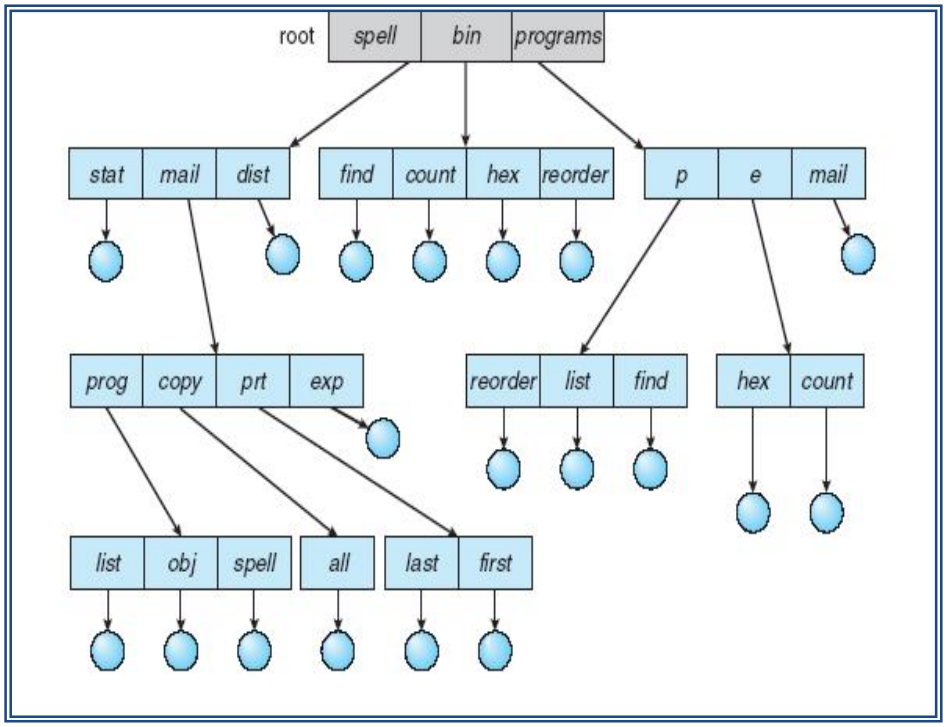


Fig: Tree-structured directory structure

4 Acyclic-Graph Directories

- An acyclic graph is, a graph with no cycles that allows directories to share subdirectories and files.
- The *same* file or subdirectory may be in two different directories. The acyclic graph is a natural generalization of the tree-structured directory scheme.
- An acyclic-graph directory structure is more flexible than is a simple tree
- Structure, but it is also more complex.
- Several problems are there in acyclic graph:
- First one is a file may have multiple absolute path names. Consequently, distinct file names may refer to the same file.
- This situation is similar to the aliasing problem for programming languages.
- If we are trying to traverse the entire file system to find a file, to accumulate statistics on all files, or to copy all files to backup storage.

- This problem becomes significant, since we do not want to traverse shared structures more than once.
- Another problem involves deletion. When can the space allocated to a shared file be de allocated and reused?
- One possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointers to the now-nonexistent file.
- If the remaining file pointers contain actual disk addresses, and the space is subsequently reused for other files, these dangling pointers may point into the middle of other files.

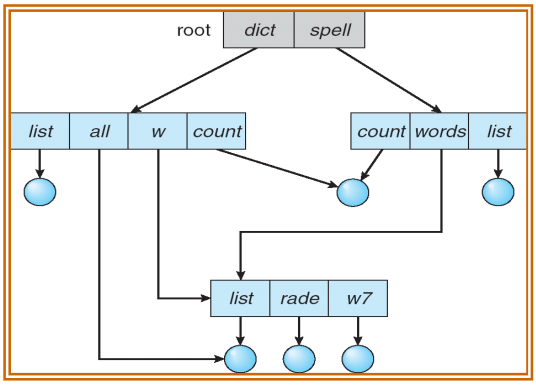


Fig: Acyclic-graph directory structure

5 General Graph Directory

If cycles are allowed in the graphs, then several problems can arise:

- Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms.
- Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero.
- Periodic garbage collection is required to detect and resolve this problem.

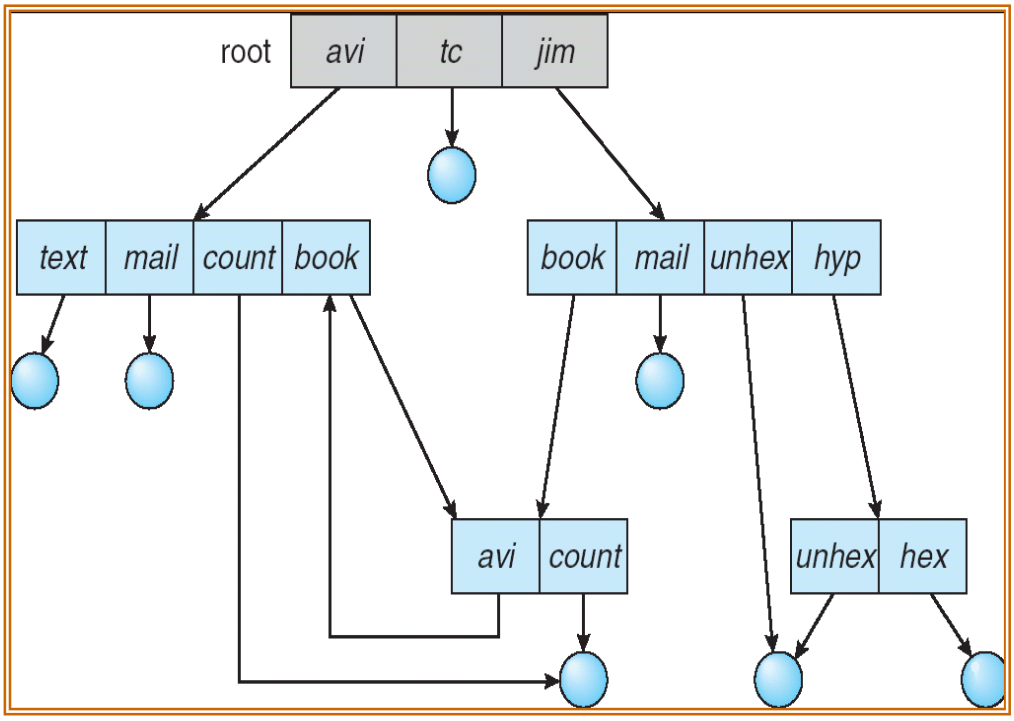
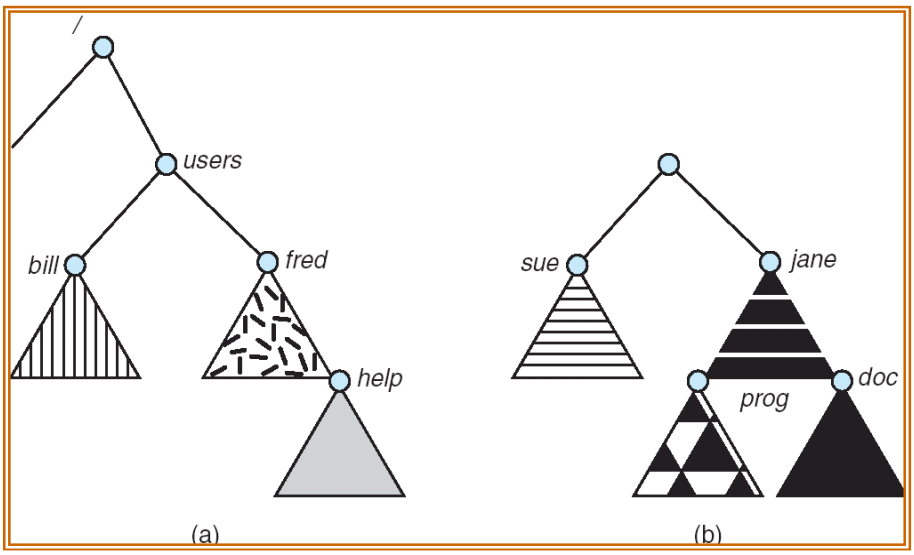


Fig: General graph directory

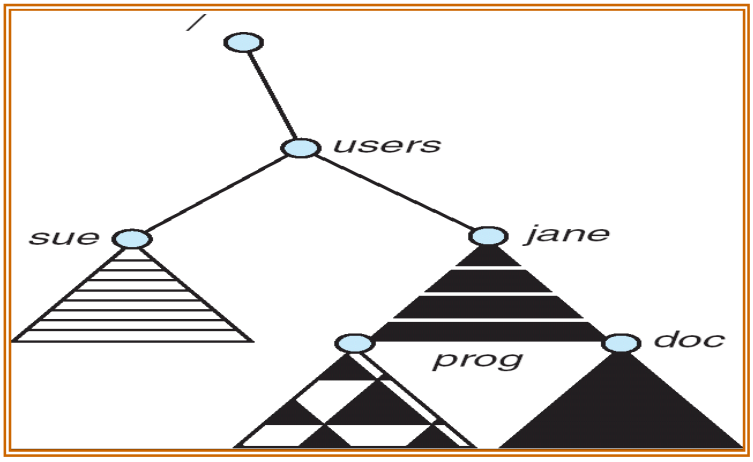
File System Mounting

- Combining two or more number of files into a large tree structure is the basic idea of file system mounting.
- In the file system we are using mount command for the purpose of at which point file has to mount.
- That means it provide a mount point (directory) on which to attach it.
- Once a file system is mounted onto a mount point, any further references to that directory actually refer to the root of the mounted file system.
- Any files (or sub-directories) that had been stored in the mount point directory prior to mounting the new file system are now hidden by the mounted file system, and are no longer available.
- For this reason some systems only allow mounting onto empty directories.
- File systems can only be mounted by root, unless root has previously configured certain file system to be mountable onto certain pre-determined mount points.

- Anyone can run the mount command to see what file systems is currently mounted. File systems may be mounted read-only, or have other restrictions imposed.



(a) Existing. (b) Unmounted Partition



Mount Point

- The traditional Windows OS runs an extended two-tier directory structure, where the first tier of the structure separates volumes by drive letters, and a tree structure is implemented below that level.

- Macintosh runs a similar system, where each new volume that is found is automatically mounted and added to the desktop when it is found.
- More recent Windows systems allow file systems to be mounted to any directory in the file system, much like UNIX.

File Sharing

1. Multiple Users

- On a multi-user system, more information needs to be stored for each file:
- The owner (user) who owns the file, and who can control its access.
- The group of other user IDs that may have some special access to the file.
- What access rights are afforded to the owner (User), the Group, and to the rest of the world.
- Some systems have more complicated access control, allowing or denying specific accesses to specifically named users or groups.

2. Remote File Systems:

- The advent of the Internet introduces issues for accessing files stored on remote computers
- The original method was ftp, allowing individual files to be transported across systems as needed. Ftp can be either account or password controlled, or anonymous, not requiring any user name or password.
- Various forms of distributed file systems allow remote file systems to be mounted onto a local directory structure, and accessed using normal file access commands.
- The WWW has made it easy once again to access files on remote systems without mounting their file systems, generally using (anonymous) ftp as the underlying file transport mechanism.

a) The Client-Server Model

- When one computer system remotely mounts a file system that is physically located on another system, the system which physically

owns the files acts as a server, and the system which mounts them is the client.

- User IDs and group IDs must be consistent across both systems for the system to work properly
- The same computer can be both a client and a server.
- There are a number of security concerns involved in this model:
- Servers commonly restrict mount permission to certain trusted systems only. Spoofing (a computer pretending to be a different computer) is a potential security risk.
- Servers may restrict remote access to read-only.
- Servers restrict which file systems may be remotely mounted. Generally the information within those subsystems is limited, relatively public, and protected by frequent backups.
- The NFS (Network File System) is a classic example of such a system.

b) Distributed Information Systems

- The Domain Name System, DNS, provides for a unique naming system across the entire Internet.
- Domain names are maintained by the Network Information System, NIS, which unfortunately has several security issues.
- NIS+ is a more secure version, but has not yet gained the same widespread acceptance as NIS.
- Microsoft's Common Internet File System, CIFS, establishes a network login for each user on a networked system with shared file access.
- Older Windows systems used domains, and newer systems (XP, 2000), use active directories.
- User names must match across the network for this system to be valid.
- A newer approach is the Lightweight Directory-Access Protocol, LDAP, which provides a secure single sign-on for all users to access all resources on a network.

- This is a secure system which is gaining in popularity, and which has the maintenance advantage of combining authorization information in one central location.

Protection

- Files must be kept safe for reliability and protection.
- The former is usually managed with backup copies.
- One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

Types of Access

The following low-level operations are often controlled:

- Read - View the contents of the file
- Write - Change the contents of the file.
- Execute - Load the file onto the CPU and follow the instructions contained therein.
- Append - Add to the end of an existing file.
- Delete - Remove a file from the system.
- List -View the name and other attributes of files on the system.
- Higher-level operations, such as copy, can generally be performed through combinations of the above.

Access Control

- One approach is to have complicated Access Control Lists, ACL, which specify exactly what access is allowed or denied for specific users or groups.
- The AFS uses this system for distributed access.
- Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown.
- UNIX uses a set of 9 access control bits, in three groups of three.
- These correspond to R, W, and X permissions for each of the Owner, Group, and Others. The RWX bits control the following privileges for ordinary files and directories:

bit	Files	Directories
R	Read (view) file contents.	Read directory contents. Required to get a listing of the directory.
W	Write (change) file contents.	Change directory contents. Required to create or delete files.
X	Execute file contents as a program.	Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.

In addition there are some special bits that can also be applied:

- The set **user ID (SUID) bit** and/or the set group ID (SGID) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program.
- The **sticky bit** on a directory modifies write permission, allowing users to only delete files for which they are the owner.
    - This allows everyone to create files in /tmp, for example, but to only delete files which they have created, and not anyone else's.
- The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for executes permission for the user, group, and others, respectively.
    - If the letter is lower case, (s, s, t), then the corresponding execute permission is not also given.
    - If it is upper case, (S, S, T), then the corresponding execute permission is given.

Other Protection Approaches and Issues

- Some systems can apply passwords, either to individual files, or to specific sub-directories, or to the entire system.
- There is a trade-off between the number of passwords that must be maintained and the amount of information that is vulnerable to a lost or forgotten password.
- Access to a file requires access to all the files along its path as well.
- In a cyclic directory structure, users may have different access to the same file accessed through different paths.

UNIT-VI
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. _____ is a unique tag, usually a number, identifies the file within the file system. [ ]

a) File identifier b)File name c) File type d) None of the mentioned

2. Reliability of files can be increased by : [ ]

a) keeping the files safely in the memory

b) making a different partition for the files

c) by keeping them in external storage

d) by keeping duplicate copies of the file

3. The main problem with access control lists is : [ ]

a) their maintenance c) their length

b) their permissions d) all of the mentioned

4. Many systems recognize three classifications of users in connection with each file (to condense the access control list) : [ ]

a) Owner b) Group c) Universe d) All of the mentioned

5. To create a file [ ]

a) allocate the space in file system

b) make an entry for new file in director

c) allocate the space in file system & make an entry for new file in directory

d) none of the mentioned

6. File type can be represented by [ ]

a) file name c) file extension

b) file identifier d) none of the mentioned

7. What is the mounting of file system? [ ]

a) crating of a file system

b) deleting a file system

c) attaching portion of the file system into a directory structure

d) removing portion of the file system into a directory structure

8. Which one of the following explains the sequential file access method?

a) random access according to the given byte number [ ]

b) read bytes one at a time, in order

c) read/write sequentially by record

d) read/write randomly by record

9. Sequential access method _____ on random access devices.

a) works well [ ]

b) doesnt work well

c) maybe works well and doesnt work well

d) none of the mentioned

10. The direct access method is based on a _____ model of a file, as _____ allow random access to any file block. [ ]

a) magnetic tape, magnetic tapes c) tape, tapes

b) disk, disks d) all of the mentioned

11. For a direct access file : [ ]

a) there are restrictions on the order of reading and writing

b) there are no restrictions on the order of reading and writing

c) access is restricted permission wise

d) access is not restricted permission wise

12. A relative block number is an index relative to :

a) the beginning of the file [ ]

b) the end of the file

c) the last written position in file

d) none of the mentioned

13. For large files, when the index itself becomes too large to be kept in memory : [ ]

a) index is called

b) an index is created for the index file

c) secondary index files are created

d) all of the mentioned

14. The directory can be viewed as a _____ that translates file names into their directory entries. [ ]

a) symbol table b) partition c) swap space d) cache

15. In the single level directory : [ ]

a) All files are contained in different directories all at the same level
b) All files are contained in the same directory
c) Depends on the operating system
d) None of the mentioned

16. In the two level directory structure : [ ]
a) each user has his/her own user file directory
b) the system doesn't its own master file directory
c) all of the mentioned
d) none of the mentioned

17. The disadvantage of the two level directory structure is that :
a) it does not solve the name collision problem [ ]
b) it solves the name collision problem
c) it does not isolate users from one another
d) it isolates users from one another

18. In the tree structured directories: [ ]
a) the tree has the stem directory
b) the tree has the leaf directory
c) the tree has the root directory
d) all of the mentioned

19. Path names can be of two types : [ ]
a) absolute & relative c) local & global
b) global & relative d) relative & local

20. When keeping a list of all the links/references to a file, and the list is empty, implies that : [ ]
a) the file has no copies
b) the file is deleted
c) the file is hidden
d) none of the mentioned

SECTION-B

Descriptive Questions

1. Explain different directory structures.
2. What are the operations that can be performed on a file?

3. How Access to files is controlled?
4. What is direct access method for files?
5. Explain various file accessing methods.
6. Write about single level and two level directory Structures.
7. What is a File? Explain about Files Sharing and Protection.
8. Discuss about the Single level directory structure.
9. Discuss about the two level directory structure.
10. Explain about different file attributes?
11. Briefly explain about file system mounting?
12. Explain about file system protection?