ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

(ACADEMIC YEAR 2023-24)

LESSON NOTES

SUBJECT: OPERATING SYSTEMS

SUB CODE: BCS303
SEMESTER: III

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
    - To strive to attain ever-higher benchmarks of educational excellence.

Department of Computer Science & Design

Vision of the Department

- To develop highly talented individuals in Computer Science and Engineering to deal with real world challenges in industry, education, research and society.

Mission of the Department

- To inculcate professional behaviour, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.
- Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research

Program Educational Objectives (PEO'S):

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in related areas.

Program Specific Outcomes (PSOs)

PSO1: Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.

PSO2: Apply knowledge in the field of computer networks for building network and internet based applications.

OPERATING SYSTEMS
(Effective from the academic year 2023 -2024) SEMESTER – III

Subject Code	BCS303	IA Marks	40
Number of Lecture Hours/Week	3:0:2	Exam Marks	60
Total Number of Lecture Hours	40+20	Exam Hours	03

CREDITS – 4

Course objectives: This course will enable students to

- Introduce concepts and terminology used in OS
- Explain threading and multithreaded systems
- Illustrate process synchronization and concept of Deadlock
- Introduce Memory and Virtual memory management, File system and storage techniques

Module – 1	**Teaching Hours**
Introduction to operating systems, System structures: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management; Protection and Security; Distributed system; Special-purpose systems; Computing environments. **Operating System Services:** User - Operating System interface; System calls; Types of system calls; System programs; Operating system design and implementation; Operating System structure; Virtual machines; Operating System debugging, Operating System generation; System boot.	**08 Hours**
Module – 2	
Process Management: Process concept; Process scheduling; Operations on processes; Inter process communication **Multi-threaded Programming:** Overview; Multithreading models; Thread Libraries; Threading issues. **Process Scheduling:** Basic concepts; Scheduling Criteria; Scheduling Algorithms; Thread scheduling; Multiple-processor scheduling,	**08 Hours**
Module – 3	
Process Synchronization: Synchronization: The critical section problem; Peterson's solution; Synchronization hardware; Semaphores; Classical problems of synchronization; **Deadlocks:** System model; Deadlock characterization; Methods for handling deadlocks; Deadlock prevention; Deadlock avoidance; Deadlock detection and recovery from deadlock.	**08 Hours**
Module – 4	

Memory Management: Memory management strategies: Background; Swapping; Contiguous memory allocation; Paging; Structure of page table; Segmentation. **Virtual Memory Management:** Background; Demand paging; Copy-on-write; Page replacement; Allocation of frames	**08 Hours**
Module – 5	
File System, Implementation of File System: File system: File concept; Access methods; Directory and Disk structure; File system mounting; **File sharing; Implementing File system:** File system structure; File system implementation; Directory implementation; Allocation methods; Free space management. Secondary Storage Structure, **Protection:** Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management; Protection: Goals of protection, Principles of protection, Domain of protection, Access matrix.	**08 Hours**

Course outcomes: The students should be able to:

- Demonstrate need for OS and different types of OS
- Apply suitable techniques for management of different resources
- Use processor, memory, storage and file system commands
- Realize the different concepts of OS in platform of usage through case studies

Question paper pattern:

The question paper will have TEN questions. There will be TWO questions from each module.
Each question will have questions covering all the topics under a module.
The students will have to answer FIVE full questions, selecting ONE full question from each module.

Text Books:

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Principles 8th edition, Wiley-India, 2015

Reference Books

1. Ann McHoes Ida M Fylnn, Understanding Operating System, Cengage Learning, 6th Edition 2
2. D.M Dhamdhere, Operating Systems: A Concept Based Approach 3rd Ed, McGraw- Hill, 2013.
3. P.C.P. Bhatt, An Introduction to Operating Systems: Concepts and Practice 4th Edition, PHI(EEE), 2014. n.

MODULE 1
INTRODUCTION TO OPERATING SYSTEMS, STRUCTURES

Introduction

Objective

What Operating System Do.

Computer System Organization.

Computer System Architecture.

Operating System Structure.

Operating System Operations.

Process Management.

Memory Management.

Storage Management.

Protection and Security.

Distributed System.

Special-Purpose Systems.

Computing Environments.

Operating System Services.

User-Operating System Interface.

System Calls, Types Of System Calls.

System Programs.

Operating System Design and Implementation.

Operating System Structure.

Virtual Machines and System Boot.

Operating System debugging.

Operating System Generation.

Introduction

This unit gives the overview of what OS do, computer system organization, computer system architecture, operating system structure and operating system operations. The different computing environments are discussed in detail. The introduction to system calls is given and types of system calls are discussed. The concept of virtual machines in discussed.

Objective:

- Understand the need of OS
- Understand process, memory and storage management
- Understand the concept of virtual machines, security, and system calls etc.
- Understand different process scheduling algorithms.
- Understand the concept of inter process communication.

What operating systems do

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- The purpose of an OS is to provide a environment in which the user can execute the program in a convenient & efficient manner.
- OS is an important part of almost every computer systems.

Computer system components

Computer system can be divided into four components-

1. ***Hardware:*** provides basic computing resources.Ex: CPU, Memory, I/O devices.
2. ***Operating system:*** controls and coordinates use of hardware among various applications and users.
3. ***Application Programs:*** defines the ways in which the system resources are used to solve the computing problems. Ex: word processors, compilers, web-browsers, database system.
4. ***Users***: Ex: people, machines ,other components

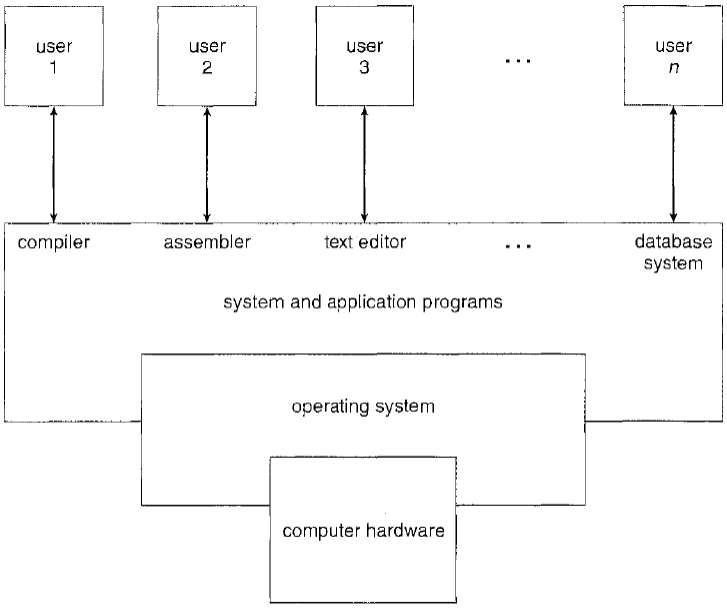


Figure 1.1 Abstract view of the components of a computer system.

Role of operating System with user and system view points

Operating system can be explored from two view points-

- User view

- System view

The user's view of the computer varies according to the interface being used.

Single user: Most users sit in front of a PC, consisting of a monitors, keyboard , mouse and system unit .Here the goal is to maximize the work.

Mainframe: In other cases user sits at a terminal connected to a mainframe or mini-computer. Other users are accessing the same through other terminals. Resource sharing is main goal here.

Handheld computer: Many types of handheld computer are used for easy of use . some are connected to network through wired or wireless medias embedded computers are used to run without user intervention.

Embedded computers are used to run without user intervention.

System view

Resource allocator:From the computer point of view operating system is viewed as a resource allocator . OS acts as a manager of resources like CPU, memory, files etc.

Control program: OS also viewed as a control program it manages the execution of user of computers.

Operating System Definition:

Operating system is the one program running at all times on the computer (called as kernel). Everything else is either a system program or an application program.

Computer System Organization

Computer System Operation

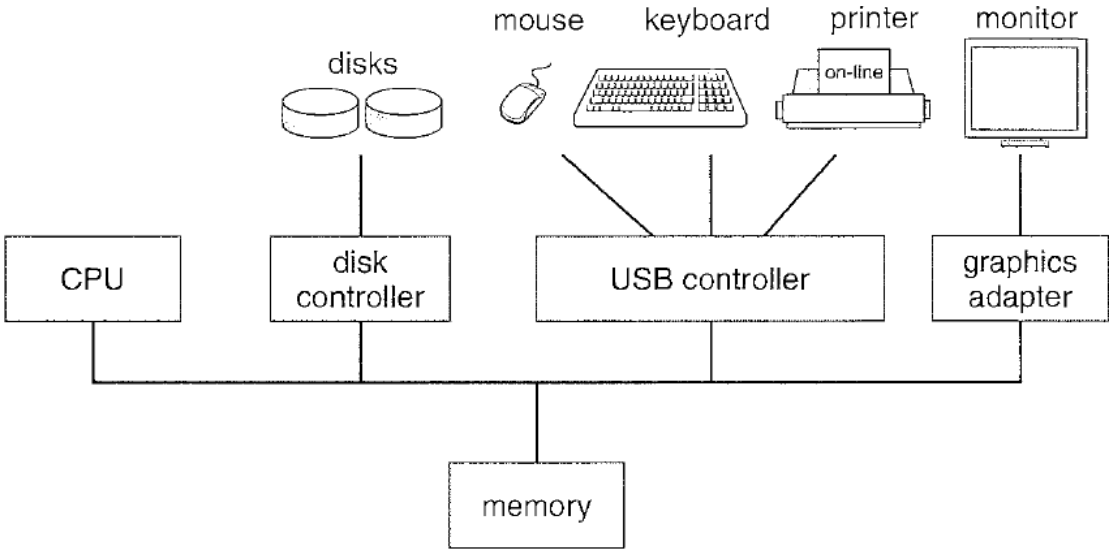


Figure 1.2 A modern computer system.

- One or more CPU, device controllers connect through common bus providing access to shared memory.
- The CPU, and device controllers can execute concurrently computing for memory cycles.
- For a computer to start running ***bootstrap program*** is required. It initializes all aspects of the system. It locates and loads the operating system kernel to memory. The OS then starts executing the first process such as "init" and waits for some event to occur.
- Occurrence of an event is signaled by an ***interrupt*** from either hardware or software.
  - Hardware triggers an interrupt by sending a signal to the CPU, by way of the system bus.
  - Software triggers an interrupt by executing a special operation called a system call

Common Function of Interrupts

- Interrupt transfers control to the interrupt service outline through the interrupt vectors, which contains the addresses of all service routines.
- Address of the interrupted instruction must be saved in interrupt architecture.

- A trap is a software generated interrupt caused either by an error or a user request.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- It determines which type of interrupt has occurred i.e. polling and vectored interrupt system
- Separate code segments are used to determine type of action that has to be taken for each type of interrupt.

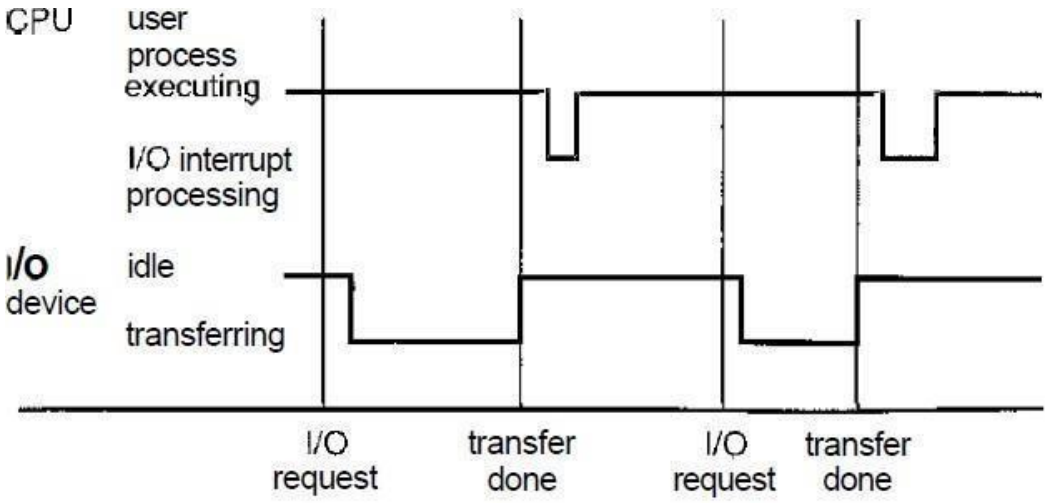


Figure 1.3 Interrupt time line for a single process doing output.

Storage structures

Computer programs must be in main memory or Random-Access Memory or RAM. It is the only large storage media that the CPU can access directly.

Programs and data must reside in main memory, this is not possible because

- ✓ Main memory is too small
- ✓ Main memory is volatile storage.

Thus, to hold large quantities of data most computer system provides secondary storage as an extension of main memory. Most common secondary-storagedevice is a magnetic disk, which provides storage for both programs and data.

Storage system are organized based on

- ✓ Speed
- ✓ Cost
- ✓ Volatility

Higher levels are expensive but fast.

Caching:

Information is copied from slower to faster storage temporarily.Cache is checked first to determine the availability of information. If it is not present information is copied to cache.

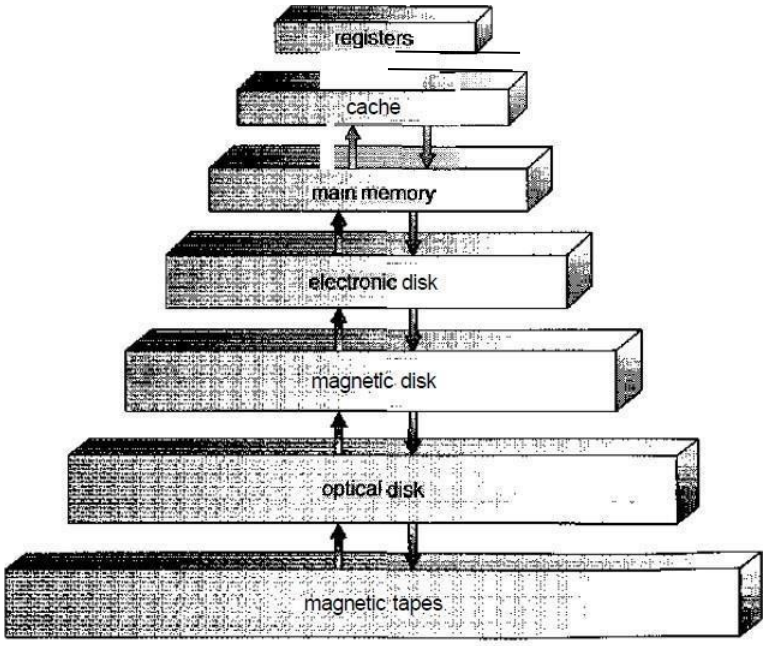


Figure 1.4 Storage-device hierarchy.

I/O structure

Computer system consists of CPU's and multiple device controllers that are connected through a common bus.

Operating systems have a device driver for each device controller which presents a uniform interface to the device to the rest of the operating system.

Interrupt-Driven I/O

To start an I/O operation, the device driver loads the appropriate registers within the device controller.

The device controller examines the contents of these registers, and starts the transfer of data from the device to its local buffer.

Once the transfer of data is complete, the device controller informs the device driver via an interrupt. Device driver returns the control to operating system along with status information.

Direct Memory Access(DMA)

Interrupt-Driven I/O produce high overhead for bulk data movement like disk I/O. To solve this problem DMA is used.

After setting up buffers, pointers and counters for the I/O device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

Only one interrupt is generated per block to tell the device driver that the operation has completed at this time CPU is available to perform other work.

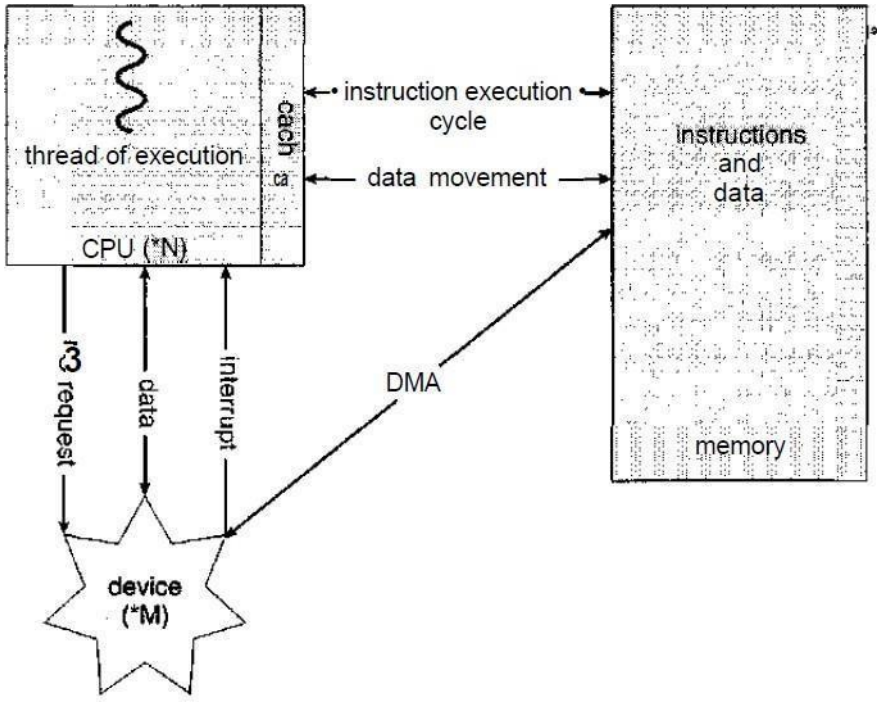


Figure 1.5 How a modern computer system works.

Computer System Architecture

Three different types of Architecture are:-

- Single-processor systems
- Multi-processor systems
- Clustered systems

Single Processor system:

On a single processor system,there is one main cpu capable of executing a general-purpose instruction set.

***Multi processor system (Parallel systems or Tightly coupled systems)*:**

Such systems have two or more processors in close communication sharing the computer bus and memory.

Multiprocessor systems have three main advantages-

- Increased throughput
- Economy of scale.
- Increased reliability.

Two types of multi-processor systems are-

- ***Symmetric Multi processors(SMP):*** In symmetric multi processing, each processors runs an identical copy of OS and they communicate with one another as needed. All the CPU shares the common memory. SMP means all processors are peers i.e. no master slave relationship exists between processors. Each processor concurrently runs a copy of OS.

- ***Asymmetric Multiprocessing:*** Each processor is assigned a specific task. A master process controls the system. Other processors look to the master for instruction. This scheme defines master-slave relationship.

The differences between symmetric & asymmetric multi processing may be result of either H/w or S/w.Special H/w can differentiate the multiple processors or the S/w can be written to allow only master & multiple slaves.

Advantages of Multi Processor Systems:

1. ***Increased Throughput:*** y increasing the Number of processors we can get more work done in less time. When multiple process co-operate on task, a certain amount of overhead is incurred in keeping all parts working correctly.
2. ***Economy of Scale:*** Multi processor system can save more money than multiple single processor, since they share peripherals, mass storage & power supplies. If many programs operate on same data, they will be stored on one disk & all processors can share them instead of maintaining data on severalsystems.
3. ***Increased Reliability:***If a program is distributed properly on several processors, than the failure of one processor will not halt the system but it only slows down.

Operating System Structure

Multiprogramming: It increases CPU utilization by organizing jobs (code and data)so that the CPU always has one to execute. Fig shows memory layout for a multiprogramming system.

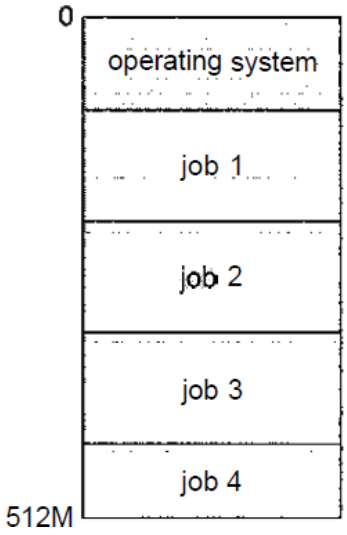


Fig: Memory layout for a multiprogramming system

- In multiprogramming system a subset of total jobs in system is kept in memory.
- One job is selected and run via job scheduling.
- When it has to wait for I/O, OS switches to another job.
- No user interaction with computer system.

Time sharing (Multi Tasking)

- It is a logical extension of multiprogramming in which CPU switches jobs so frequently that users can interact with each job while it is running.(interactive computing)
- Response time should be less than 1 second.
- Each user has at least one program executing in memory process.
- CPU scheduling is used to select a job from the job pool.
- If processes don't fit in memory. Swapping moves them in and out to run.
- Virtual memory allows execution of processes not completely in memory.

Operating System Operations

- Modern operating systems are interrupt driven.
- Events are signaled by the occurrence of an interrupt or a trap.(Trap is a software generated interrupt. ex: divide by zero. It is also called as exception.)

Dual mode

Operation allows OS to protect itself and other system components.

Two different modes are

1. User mode
2. Kernel mode.

Hardware provides mode bit to switch between the modes.

- It provides ability to distinguish when system is running in user mode or kernel mode.
- Some instructions are designed as privileged & only executable in kernel mode.
- System call changes the mode to kernel mode.

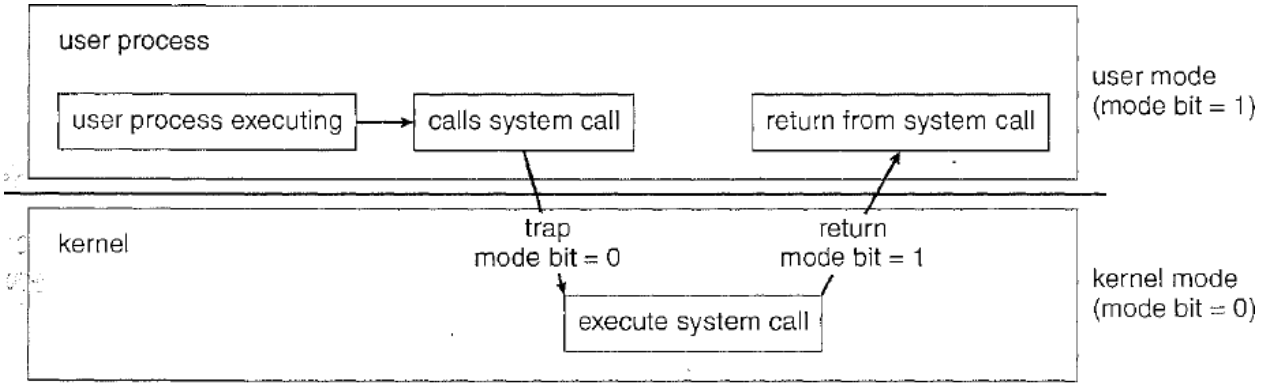


Figure 1.10 Transition from user to kernel mode.

Timer

- Timer is used to prevent a user program from getting stuck in an infinite loop.
- Timer can be set to interrupt the computer after a specified period.
- We can use the timer to prevent a user program from running too long.

Process Management

- A process is a program in execution. It us the unit of work within the system. Program is a passive entity. Process is an active entity.
- Process needs resources to accomplish its task Ex: CPU, Memory, I/O.
- Process termination requires reclaim of any reusable resources.
- Single-threaded process has one program counter specifying the location of next instruction to execute.
- Multi0threaded process has one program counter per thread.

Process Management Activities

The Operating system is responsible for the following activities in connection with process management.

- Creating and deleting both the user and system processes.
- Suspending and resuming process.
- Providing mechanisms for process synchronization and process communication.
- Providing mechanism for deadlock handling

Memory Management

- Main memory is central to the operation of a modern computer system.is a repository of quickly accessible data shared by the CPU and I/O devices.
- Instructions must be in memory for the CPU to execute them.
- For a program to be executed, it must be mapped to absolute addresses and loaded into memory. As the program executes, it accesses program instructions and data from memory by generating these absolute addresses. After program termination, its memory space is declared available, and the next program can be loaded and executed.

Memory Management Activities

- Keeping track of which parts of memory are currently being used and by whom.
- Deciding which processes and data to move into and out of memory.
- Allocating and de-allocating memory space as needed.

Storage Management

- The operating system provides a uniform, logical view of information storage.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, called as file.
- The operating system maps files onto physical media and accesses these files via the storage devices.

File system Management

- Files usually organized into directories.
- Access control mechanisms are used to determine who can access what

File System Management Activities

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

Mass Storage Management

- Main memory is too small to accommodate all data and programs, the computer system must provide secondary storage to back up main memory.
- Usually disks used to store data that does not fit in main memory or data that must be kept for a long time.
- Entire speed of computer operation hinges on disk subsystems and its algorithms

Mass storage management Activities

- Free space Management
- Storage Allocation
- Disk scheduling

Protection and security

Protection : Any Mechanism for controlling access of process or users to resources defined by the operating system

Security : Defense of the system against internal and external attacks-like denial-of-service, viruses, identity theft, theft of service etc.

Protection and security require the system to be able to distinguish among all its users.

- User identifiers include name and associated number one per user.
- User ID then associated with all files, processes of that user to determine access control.
- Group identifier allows set of users to be defined and also associated with each process.
- User sometimes needs to *escalate privileges* to gain extra permissions for an activity. Operating system provides various methods to allow privilege escalation. Ex: setuid attribute on UNIX

Distributed Systems

A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide the users with access to the various resources that the system maintains.

Access to a shared resource increases computation speed, functionality, data availability, and reliability.

Special-Purpose Systems

Object of these computer system are to deal with limited computation domains.

Real-Time Embedded Systems

Embedded computers are the most prevalent form of computers in existence.

- Little user interface
- Prefer to spend their time in monitoring and managing hardware devices such as automobile engines and robotic arms.
- Embedded systems almost always run **real-time operating systems.**
- A real-time system is used when rigid time requirements have been placed on the operation of a processor

Multimedia Systems

Multimedia data (audio, video) along with conventional data (text file, word processing systems) must be handled efficiently to satisfy the user requirements.

Handheld Systems

Handheld systems include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones etc. Many of which use special-purpose embedded operating system.

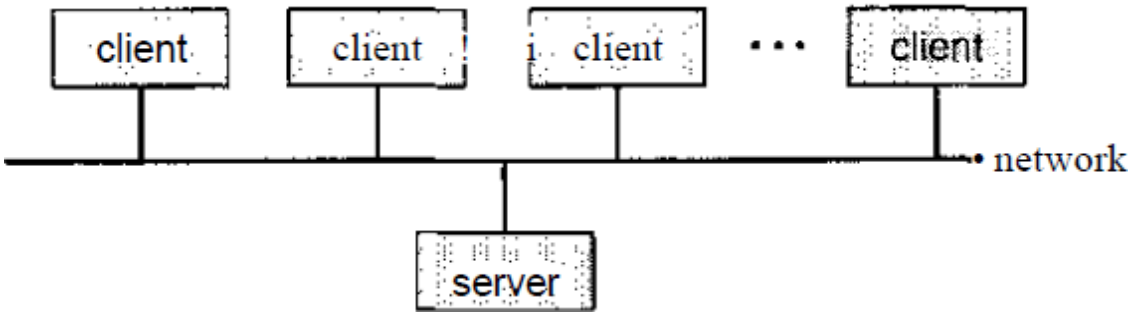
Because of their size most handheld devices will be having small amount of memory, slow processors and display screen. Programmer must consider these limitations while designing applications for handheld devices

Computing Environments

Traditional Computing

Common office environment uses traditional computing normal PC is used in traditional computing.

Client-server computing



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

Peer-to-Peer Computing

In this model, clients and servers are not distinguished from one another all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.

Peer-to-peer systems offer an advantage over traditional client-server systems. In a client-server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.

Web-Based Computing

Web-Based computing has wide range of access devices like PDA's mobiles, PC's and workstations. It focuses on networking. Faster network connectivity is provided to wired and wireless terminals.

Operating system services

Operating system services helpful to users are classified as

a) ***user interface***

All operating systems have a user interface. Several forms of UI are

1. Command- line interface
2. Batch interface
3. Graphical User Interface(GUI)

b) ***Program Execution***

System must be able to load the program into memory and to run that program. System must end the program execution.

c) ***I/O operations***

Operating systems must provide a way to perform Input/output operations.

d) ***File-System Manipulation***

Programs need to read and write files and directories. Some programs include permission management to allow or deny access to files based on file ownership.

e) ***Communications***

One process needs to exchange information with other process.

Communications must be implemented via shared memory or message passing.

f) ***Error Detection***

Errors may occur in different units of computer system. For each type of error OS should take the appropriate action to ensure correct and consistent computing.

OS functions used to ensure the efficient operations of the system are

a) ***Resources Allocation***

When multiple users or multiple jobs running at the same time, resources must be allocated to each of them.

b) ***Accounting***

We must keep track of which user use how much and what kind of resources.

c) Protection and Security

When several separate processes execute concurrently, it should not be possible for one process to interfere with the others.

Protection ensures that all access to system resources is controlled.

Security enforce each user to authenticate himself by means of a password to gain access to system resources.

User Operating System Interface

Two types are

Command Interpreters

Command interpreter is a special program that is running when a job is initiated or when a user first logs on.

On a system with multiple command interpreters it is referred as shells.

Ex: In UNIX→ Bourne Shell, Korn Shell etc.

Main function of the command interpreters is to execute user-specified command.

Ex: DOS commands.

Graphical User Interface

GUI provides a mouse based window and menu system as an interface.

Ex: In desktop mouse is moved to position its pointer on images or icons on the screen.

Depending on mouse pointer location respective program is invoked.

System calls

System calls provide an interface to the services made available by the operating system. These calls are generally available as functions written in c and C++

The run-time support system provides a system call interface that serves as the link to system calls made available by the operating system.

A number is associated with each system call and system call interface involves the intended system call in the operating system kernel and returns the status of the system call. Thus, most of the details of the operating system interface are hidden from the programmer.

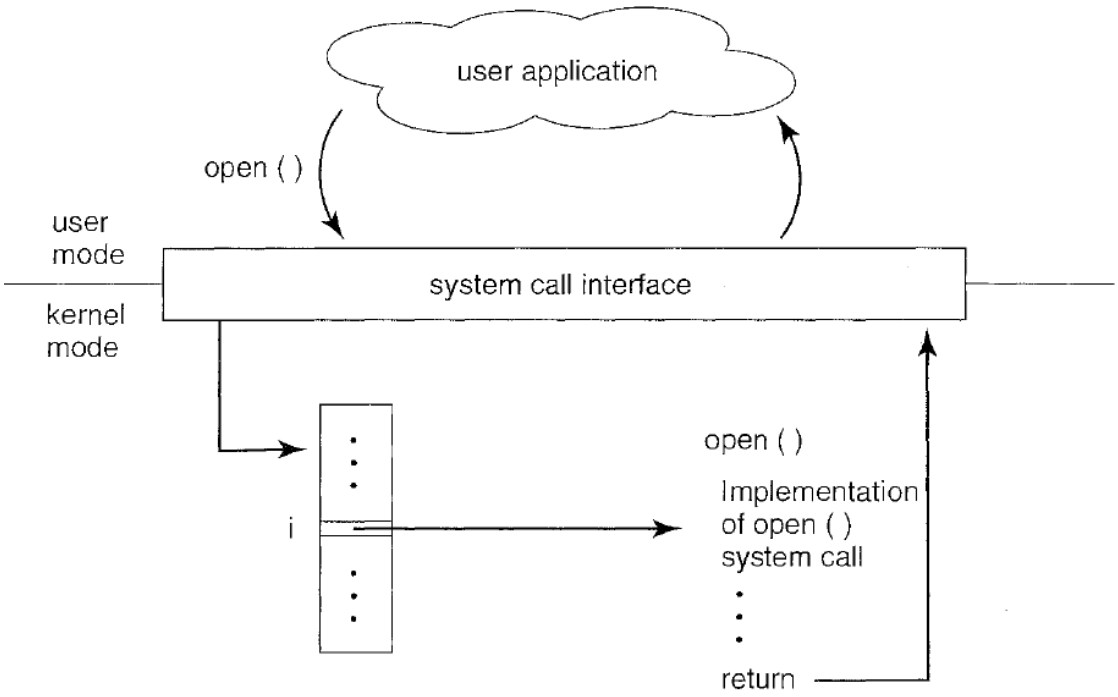


Figure 2.6 The handling of a user application invoking the `open()` system call.

Three general methods are used to pass parameters to the operating system.

- Pass the parameters in registers.
- Passing the address of the block or table.
- Parameters are pushed into the stack by program and popped off the stack by the OS

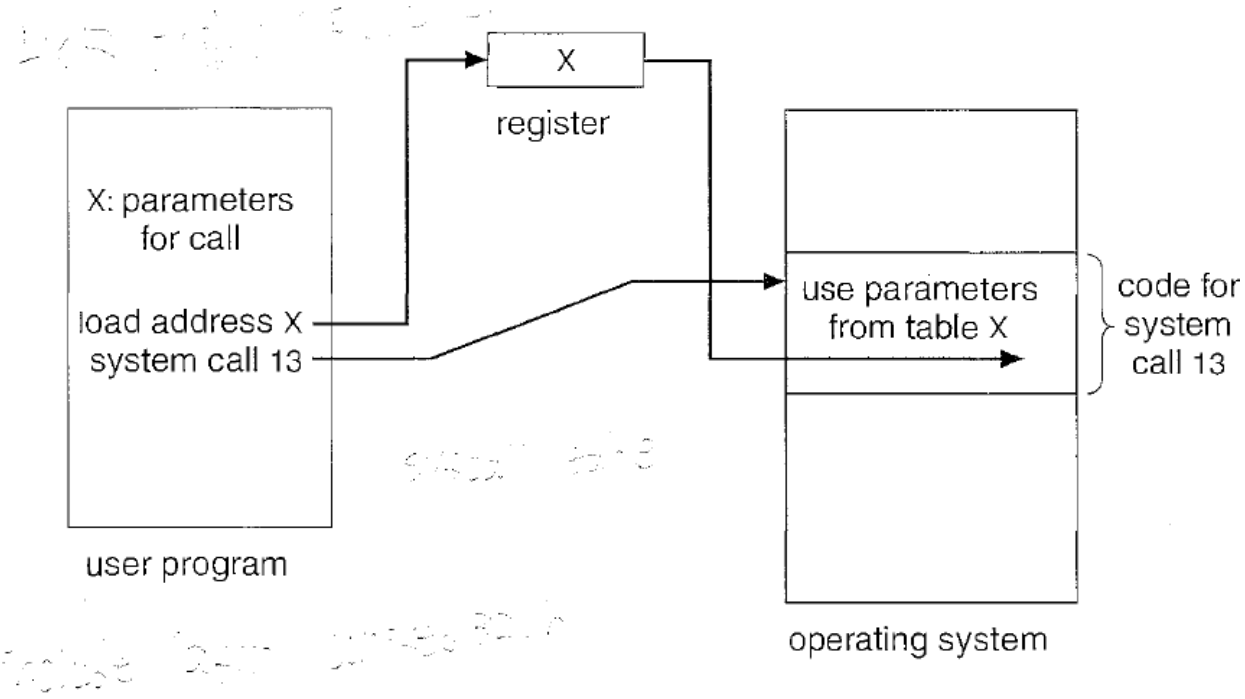


Figure 2.7 Passing of parameters as a table.

Types of System Calls

System calls are grouped into five categories

- ***Process control***
  - End, abort
  - Load ,execute
  - Create process, Terminate process
- ***File Management***
  - Create file, delete file
  - Open, close
  - Read, write.
- ***Device Management***
  - Request device, release device
  - Get device attributes, set device attributes.
- ***Information Maintenance***
  - Get time or date
  - Set time or date
  - Set process, file or device attributes.

- ***Communications***
    - Create, delete communication connection
    - Send, receive messages.

Process Control

- Running programs can be halted by 2 methods
    - Normal
    - Abnormal.
- Operating system transfer control to the command interpreter in the normal or abnormal conditions.
- Some operating system allow control cards to indicate special recovery action in case an error occurs.
- Load and execute system calls are used by process to execute the programs.
- Get process attributes and set process attributes system calls are used to determine and reset the attributes of a job.
- Debugger is used to help the programmer in finding and correcting errors
- Two different types of process control methods are-
    - multitasking and single tasking.
- Ms-Dos operating system is an example of a single tasking system, which has a command interpreter that is involved when computer is started. ms-dos does not create a new process while running one process.

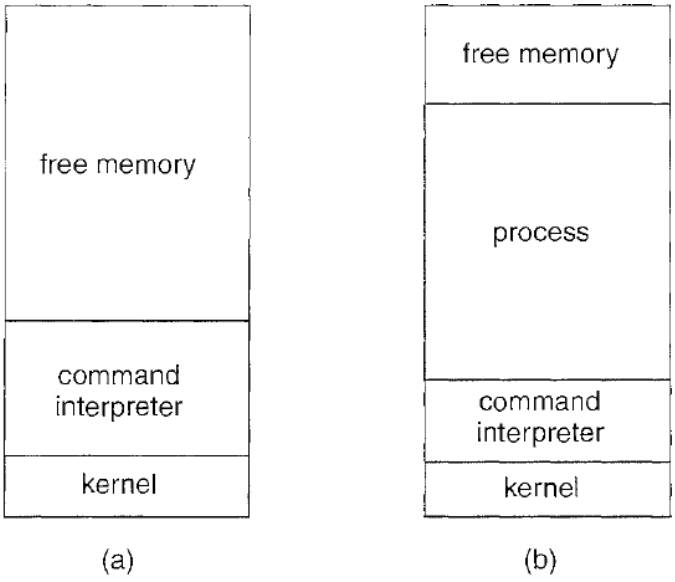


Figure 2.10 MS-DOS execution. (a) At system startup. (b) Running a program.

- Ms-Dos loads the program into memory writing over most of itself to give the program as much memory as possible.
- Ms-Dos does not have multitasking capabilities. Free BSD is an example of multitasking system.
- In free BSD, the command interpreter may continue running while another program is executed. Fork system call is used to create new process.

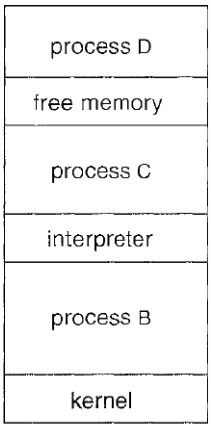


Figure 2.11 FreeBSD running multiple programs.

File Management

Files are created and deleted with the help of system calls. It require name of the file with file attributes for creating and deleting files.

Other operations performed on file is read,write and update. Close system call is used to close a file

Device Management

System calls are also used for accessing a device. When more than one user is accessing a device, request is made before use of the device. After using the device user must release the device. Release system call is used to free the device.

Ms-Dos and UNIX merge the I/O devices and files to form file-device structure. In this structure I/O devices are identified by special file names.

System Programs

System programs can be divided into following categories.

- ***File Management***
  These programs create, delete, copy, list and manipulate files and directories.
- ***Status Information***
  These programs gives details about date, time, amount of available memory etc
- ***File Modification***
  Text editors are used to create and modify the contents of files stored on disk or other storage devices.
- ***Programming Language Support***
  Compilers, Assemblers, Debuggers for common programming languages are provided to the user with the Operating System
- ***Program Loading and Execution***
  Once a program is assembled or compiled it must be loaded into memory to be executed. the system may provide loaders, linkage editors etc.
- ***Communications***
  These programs allows users to send messages, to browse web pages etc. in addition to system programs, most OS are supplied with application programs to solve common problem.
    - ex: word processors
    - games
    - data base systems.

OS Design and Implementation

Design Goals

At the highest level, the design of the system will be affected by the choice of hardware and type of the system like batch, time-shared etc.

Requirements can be divided into two basic groups:-

- user goals
- system goals

According to user the system must be-

- Convenient to use
- Easy to learn and use
- Reliable, safe and fast.

According to system designer the system must be-

- Easy to design , implement and maintain.
- It should be flexible , reliable & error free.

Mechanisms and Policies

- Policy and mechanism are separated from one another.
- Mechanism determine how to do something.
- Policies determine what will be done.
- Separation of policy and mechanism is important for flexibility.

Implementation

- Operating systems are written in higher level language like C or C++.
- The advantage of using higher level language is code can be written faster, is easy to understand and debug.
- The only possible disadvantage of implementing a OS in higher level language are reduced speed and increased storage requirement.

Operating system structure

Simple Structure

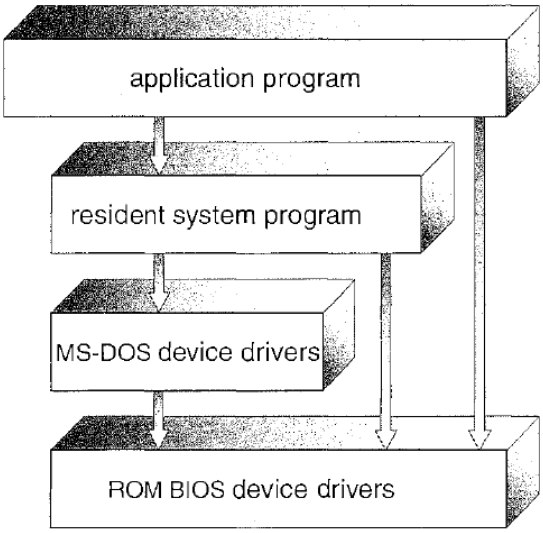


Figure 2.12 MS-DOS layer structure.

MS-DOS is all example for simple structure. It was designed to provide the most functionality in least space, so it was not divided into modules carefully.

In Ms-Dos interface and level of functionality are not well separated. Application programs are able to access the basic I/O routines to the display and disk driver. Due to this entire system may crash if it encounters the malicious program.

Another example of limited structuring is the original UNIX operating system.

It consist of two separable parts-

- The kernel
- The system programs

The kernel is separated into a series of interfaces and device drivers which have been added and expanded over the years as UNIX has evolved. This monolithic structure was difficult to implement and maintain.

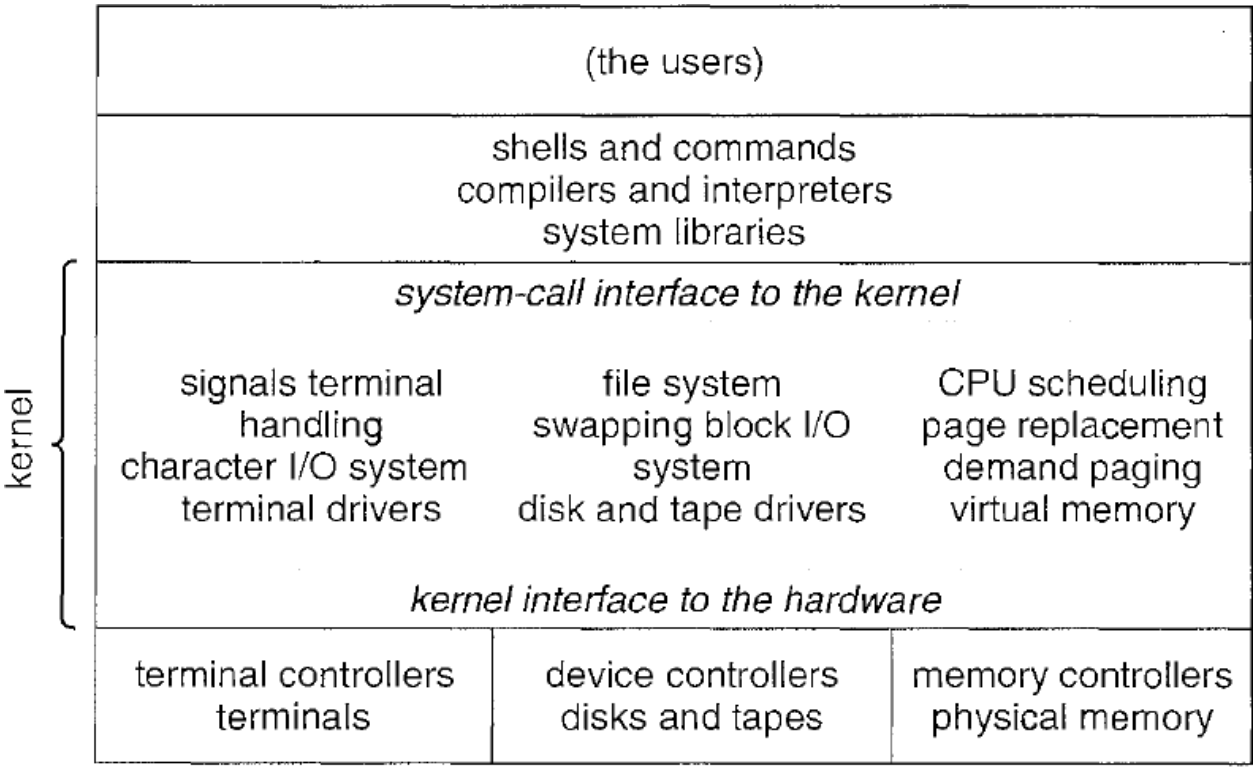


Figure 2.13 Traditional UNIX system structure.

Layered approach

- In this operating system is broken into many layers.(levels)
- The bottom layer is the hardware (layer 0) and the highest (layer N) is the user interface.

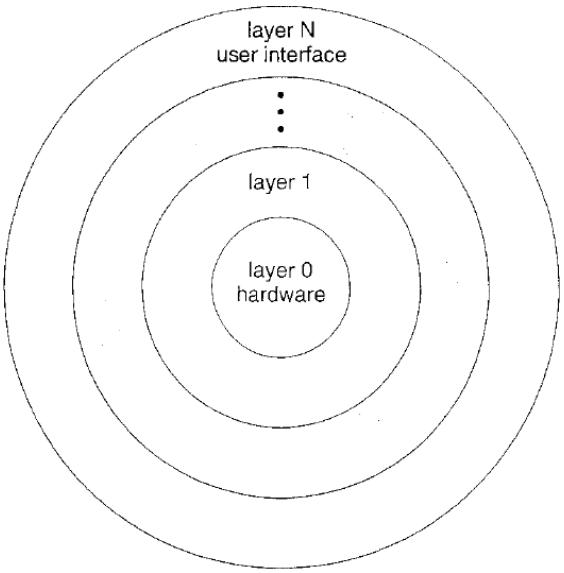


Figure 2.14 A layered operating system.

- Layer is an implementation of an abstract object. (Consist data and Operations to manipulate the data)
- The main advantage of layered approach is simplicity of construction and debugging
- Higher-level layers uses functions and services of only lower-level layers
- The major difficulty with layered approach is defining the various layers.

Micro kernels

- 'Mach' operating system modularized the kernel using the ***Microkernel*** approach.
- This method structure the operating system by removing all non-essential components from the kernel and *implementing* them as system and *user-level programs.* This results in a smaller kernel
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.
- Benefit of microkernel approach is easy of extending the operating system.All new services are added to user space and do not require modification of the kernel.
- Microkernel can suffer from performance decreases due to increased system function overhead.

Modules

- In this object oriented programming technique is used to create a modular kernel.
- Here, kernel has a set of core components and dynamically links additional services either during boot time or during run time.
  Ex: Solaris, Linux, Mac OS
- Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules

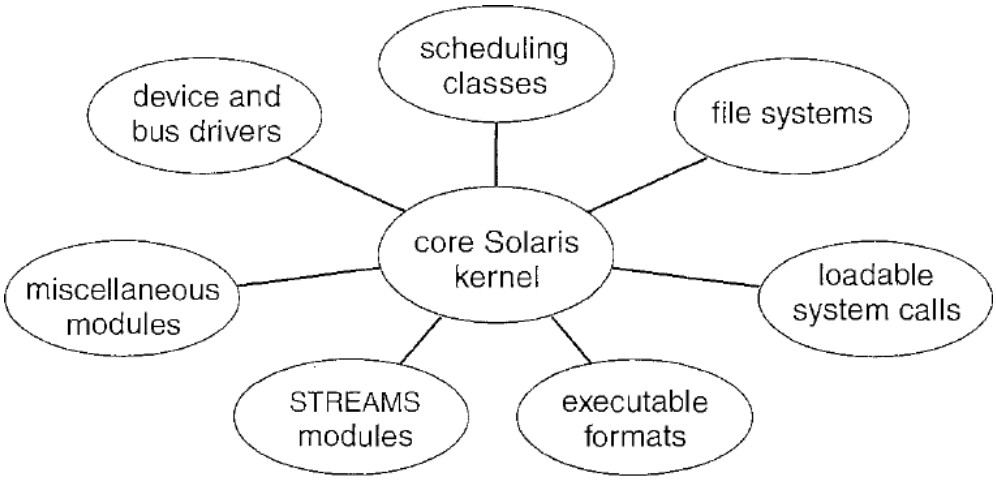


Figure 2.15 Solaris loadable modules.

- Such a design allow the kernel to provide core services yet also allows certain features to be implemented dynamically.
- It is more efficient than micro kernel approach

Virtual machines

- Concept behind a virtual machine is to abstract the hardware of a single computer into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer
- Virtual machine approach provides are interface that is identical to the underlying hardware.

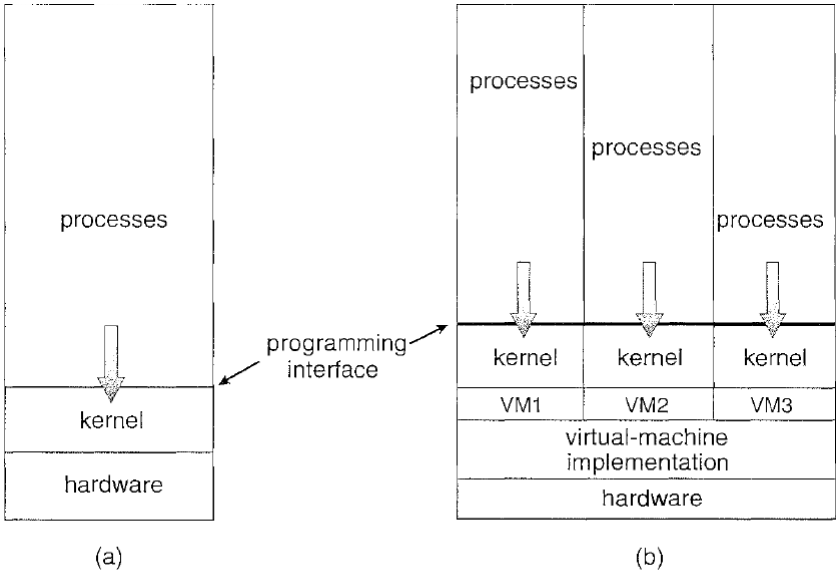


Figure 2.17 System models. (a) Nonvirtual machine. (b) Virtual machine.

- With the help of virtual machines it is possible to share the same hardware among different operating system.
- Major difficulties with virtual m/c s are allocating disk system.
- Virtual machine concept has several advantages.
    - Each VM is completely isolated from all other VM's. So no protection problem
    - There is no direct sharing of resource.
    - Virtual machine system is a perfect vehicle for operating system research and development

Example: **VMware**

- VMware abstracts Intel 8086 hardware into isolated virtual machines
- VMware runs as an application on host operating system such as windows or Linux and allows this host system to concurrently run several different ***quest operating system*** as independent virtual machines.
- In the architecture diagram Linux is running as the host Operating system.
- Free BSD, Windows NT and Windows XP are running as quest operating systems.

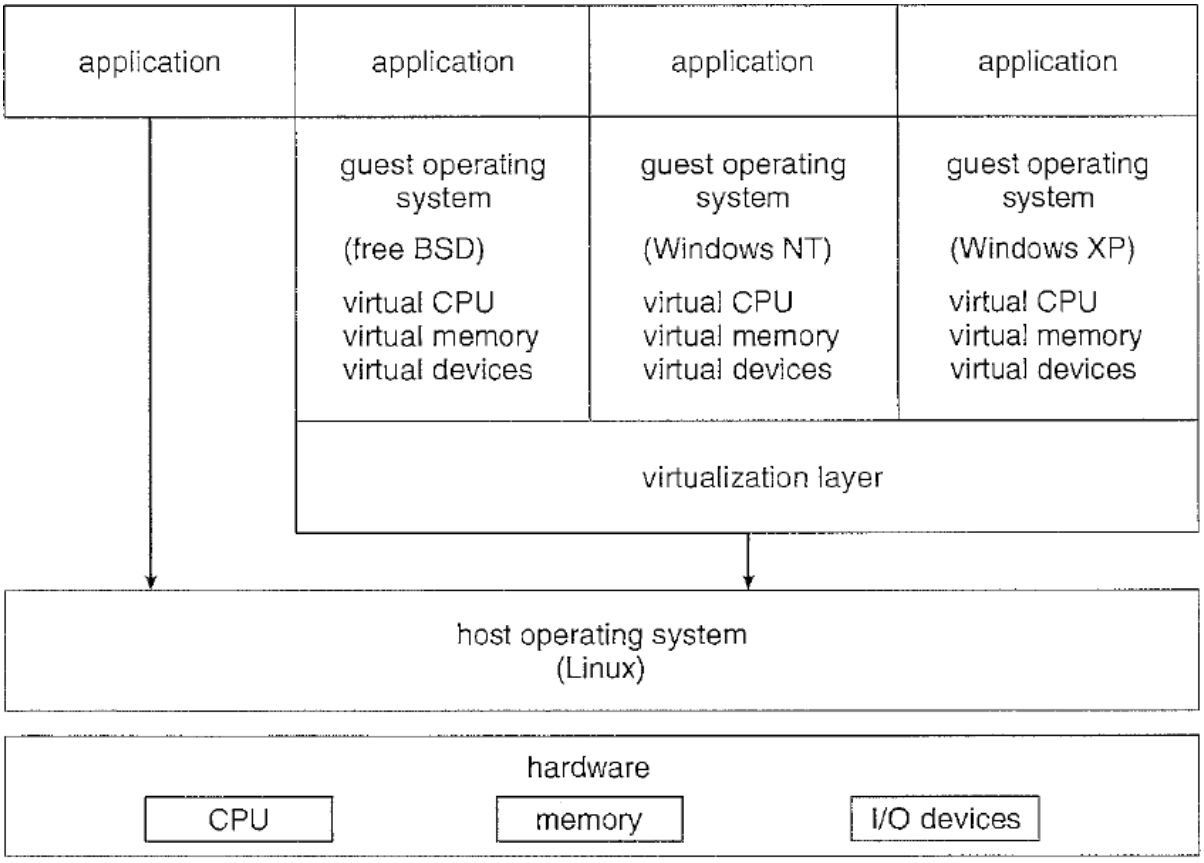


Figure 2.19 VMware architecture.

The Java Virtual Machine

Java is a popular object oriented programming language. For each Java class, the compiler produced an architecture neutral byte code output (.class) file that will run on any implementation of the JVM.

JVM is a specification for an abstract computer. It consist of a class loader and a Java interpreter that executes the architecture neutral byte codes.

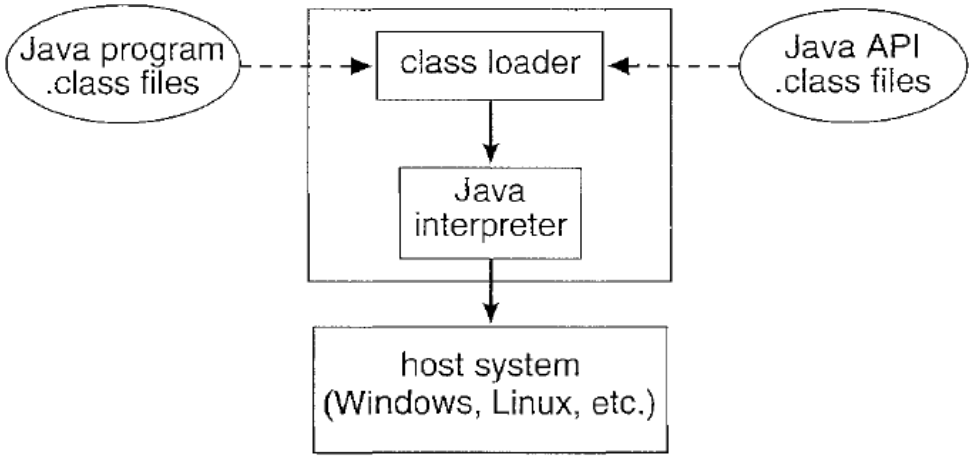


Figure 2.20 The Java virtual machine.

Operating System Generation

- Operating system are designed to run on class of machines at a variety of sites with a variety of peripheral configurations.
- The system must be configured or generated for each specific computer sites.
- This process is referred as system generation. (SYSGEN)
- This process requires the information like
    - Type of CPU used
    - Memory space available in the system
    - Available device list
    - Required operating system option

System Booting

The procedure of starting a computer by loading the kernel is known as booting the system. Bootstrap program or Bootstrap loader locates the kernel loads it into main memory and start its execution.

Bootstrap program is in the form of read only memory (ROM) because the RAM is in unknown state at a system Start up.

All forms of ROM are knows as FIRMWARE. For large OS like Windows, Mac OS, the Bootstrap loaders is stored in firmware and the OS is on disk. Bootstrap has a bit

code to read a single block at a fixed location from disk into the memory and execute the code from that boot block.

Assignment Questions

1. Explain fundamental difference between i) N/w OS and distributed OS ii) web based and embedded computing. . (8) **Dec 07/Jan 08**
2. What do you mean by cooperating process? Describe its four advantages. (6) **Dec 07/Jan 08**
3. What are different categories of system programs? Explain. (6) **Dec 07/Jan 08**
4. Define OS. Discuss its role from different perspectives. (7) **Dec 08/Jan 09.**
5. List different services of OS. Explain. (6))**Dec 08/Jan 09.**
6. Explain the concept of virtual machines. Bring out its advantages. (5) **Dec 08/Jan 09.**
7. Difference between a trap and an interrupt (2) **Dec 08/Jan 09.**
8. Define an operating system. Discuss its role with user and system view points. (06marks) **Dec.09/Jan.10**
9. Give features of symmetric and asymmetric multiprocessing systems (4) **Dec.09/ Jan.10**
10. Briefly explain common classes of services provided by various OS for helping usefor ensuring efficient operation of system. (10) **Dec.09/ Jan.10**
11. Define OS. Explain its two view points (5) **Dec 2010**

12. What are OS operations? Explain (6) **Dec.09/ Jan.10**

13. Define Virtual machine. With diagram, explain its working. What are its benefits? (9)**Dec.09/Jan.10**

14. Distinguish among following terminologies : Multiprogramming systems, multitasking systems, multiprocessor systems. (12) **Dec.09/ Jan.10**

15. What do you mean by PCB? Where is it used? What are its contents? Explain. (8) Dec 07/Jan 08

16. Explain direct and indirect communications of message passing systems. (6)) Dec07/Jan 08

17. Explain the difference between long term and short term and medium term schedulers(6) Dec 07/Jan 08

18. What is process? Draw and explain process state diagram. (5) Dec 08/ Jan 09

19. Discuss operations of process creation and termination in UNIX. (7) Dec 08/ Jan 09

20. Explain different states of a process (6) Dec 09/ Jan 10

Outcome

- Familiarize with OS and it functionality
- Know the application of virtual machines
- Implement different process, memory and storage management techniques.
- Implement different process scheduling algorithms.
- Familiarized with inter process communication

Further Reading

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne: Operating System Principles, 8th edition, Wiley India, 2009. (Listed topics only from Chapters 1 to 12, 17, 21)

2. D.M Dhamdhere: Operating systems - A concept based Approach, 2nd Edition, Tata McGraw- Hill, 2002.

3. P.C.P. Bhatt: Introduction to Operating Systems: Concepts and Practice, 2nd Edition, PHI, 2008.

4. Harvey M Deital: Operating systems, 3rd Edition, Pearson Education, 1990.

5. http://nptel.ac.in/courses/106106144

6. https://en.wikipedia.org/wiki/Operating_system

7. https://www.tutorialspoint.com/operating_system

8. http://www.studytonight.com/operating-system

9. https://www.linux.com

10. https://opensource.com

MODULE 2
PROCESS MANAGEMENT

Process concept

Process scheduling

Operations on processes

Inter process communication

Multi-Threaded Programming.

Overview;

Multithreading Models.

Thread Libraries;

Threading Issues.

Process Scheduling: Basic Concepts.

Scheduling Criteria.

Scheduling Algorithms.

Thread Scheduling.

Multiple-Processor Scheduling.

Assignment Questions

Outcome

Further Reading

Introduction

This unit gives the overview of process concept and operation on process. Different process scheduling is discussed in detail. The concepts of inter process communication and multi threaded programming are highlighted. The different scheduling algorithms with problems are explained. The concept of thread scheduling and multi processor scheduling are introduced.

Objective:

- Understand the concept of multi-threaded programming.
- Understand critical section problem
- Understand Peterson's solution
- Understand semaphores and synchronization

The Process Concept

The Process

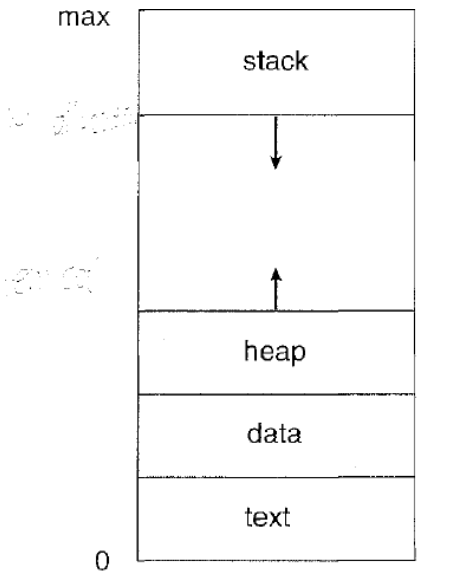


Figure 3.1 Process in memory.

- Process is program in execution.
- Process is also referred as text section which contains program counter value and processor register contents.
- Stack contains temporary data and global variables.
- Process contains heap memory which can be dynamically allocated during process run time.
- Program is a passive entity and process is a active entity.

Process states

Current activity of the process defines state of a process. Each process may be in one of the following states

New : The process being created

Running : Instruction are being executed

Waiting : The process is waiting for some event to occur

Ready : The process is waiting to be assigned to a processor

Terminated : The process has finished execution

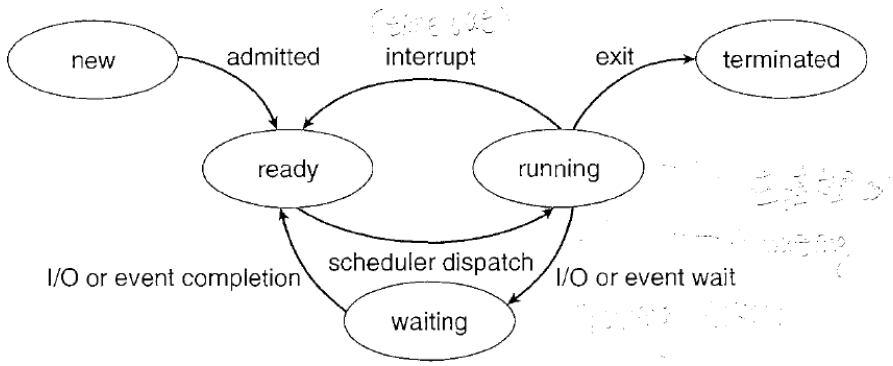


Figure 3.2 Diagram of process state.

The program stored in the boot block loads the entire OS into memory and begins its execution. A disk that has a boot partition is called a *boot disk* or *system disk*.

Process Control Block (Task Control Block)

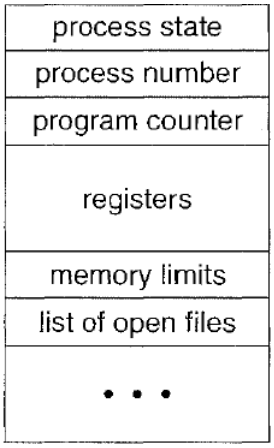


Figure 3.3 Process control block (PCB).

Each process is represented in the operating system by a **process control block (PCB)**
It contains information related to process.

- **Process state:** The state may be new, ready, running, waiting, halted etc.
- **Program counter**: The counter indicates the address of the next instructionto be executed.
- **CPU registers:** It include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward

- **CPU-scheduling information:** It includes a process priority, pointers to scheduling queues etc.
- **Memory-management information:** It includes value of the base and limit registers, the page tables, or the segment tables etc.
- **Accounting information:** It includes details about amount of CPU used, time limits, account numbers, process number etc.
- **I/O status information:** It includes the list of I/O devices allocated to the process, a list of open files etc

Threads

In modern OS Process can have multiple threads of execution to perform more than one task at a time.

Process Scheduling

Process Scheduled selects an available process for execution on the CPU.

Scheduling Queues

- Queuing Diagram is used to represent process scheduling concept.

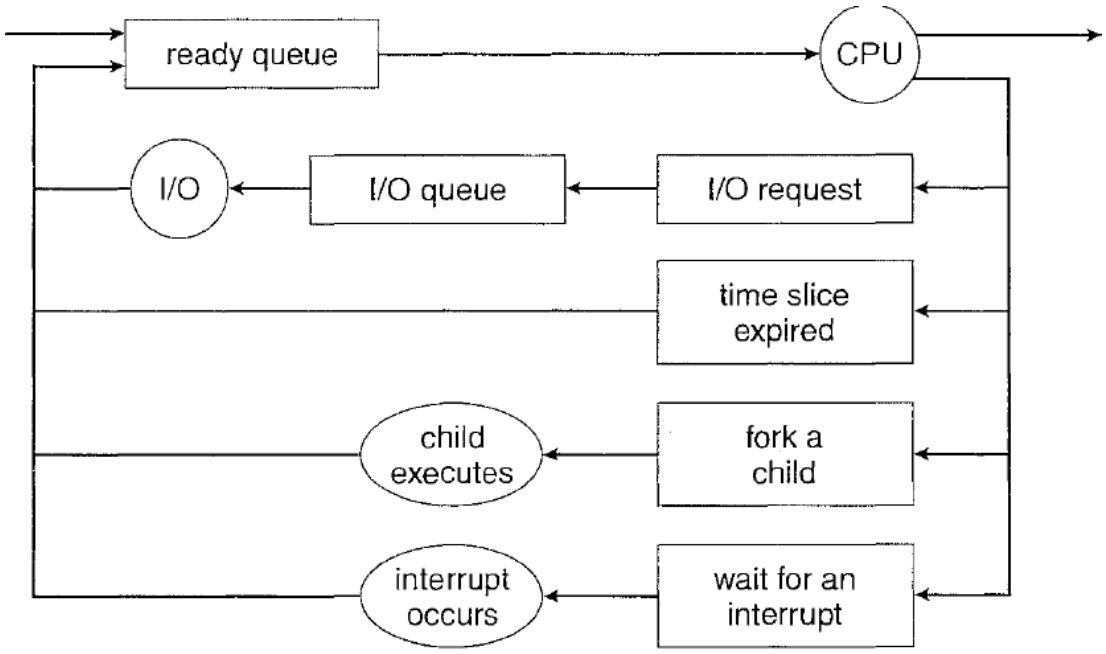


Figure 3.7 Queueing-diagram representation of process scheduling.

- As processes enter the system, they are put into a **job queue.**

- The processes residing in main memory and waiting to execute are kept in a **ready queue.**
- A process waiting for a particular I/O device resides in **device queue.**
- In a Queuing diagram rectangular box represents Queue and circles represents the resources.
- A new process is initially put in the ready queue. Once the process is allocated the CPU will starts executing, several events associated with it are :
    - Process may request I/O and placed in I/O queue
    - Process may create new sub process and wait for sub process termination
    - Process may be removed forcibly from CPU, as a result of an interrupt and put back in the ready queue.

Schedulers

- A process migrates among the various scheduling queues throughout its lifetime.
- Operating system must select, the processes from these queues with the help of scheduler
- Long-term scheduler selects the processes from mass-storage device and loads them into memory for execution.
- Short-term scheduler or CPU scheduler selects among the processes which are ready to execute and allocates the CPU to one of them.
- Short-time scheduler must be fast
- Because of *Longer interval* between executions, the long-term scheduler can take more time to decide which process should be selected for execution
- Long-term scheduler controls the degree of multiprogramming.
- Long-term scheduler must select a good mix of I/O bound and CPU bound process.
- Some Operating system uses medium term scheduler. It removes the process from memory and reduces the degree of multiprogramming. Later, process can be reintroduced into memory and execution can be continued where it left off.

Context Switch

- When an interrupt occurs, the system needs to save the current **context** of the process currently running on the CPU so that it can restore that context when its processing is done.

- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch.**
- Context-switch time is pure overhead, because the system does no useful work while switching.

Operations on Processes

Process Creation

- A process can create several new processes, with the help of create-process system call.
- The creating process is called a **parent** process, and the new processes are called the **children** of that process.
- Each of these new processes may in turn create new processes forming a **tree** of processes.
- Unique **process identifier**(or pid) is used to identify the process.
- A process will need certain resources to accomplish its task. When a process creates a sub process that sub process may be able to obtain its resources directly from the OS or parent may have to partition its resources among its children.
- When a process creates a new process, two possibilities exist in terms of execution:
  1. The parent continues to execute concurrently with its children.
  2. The parent waits until some or all of its children have terminated.

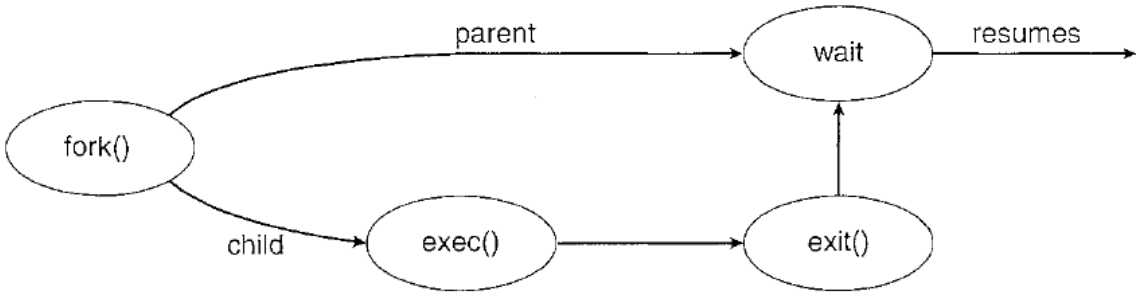


Figure 3.11 Process creation using `fork()` system call.

- **Fork ()** A new process is created by fork system call
- **Exec()** It is used after a fork() system call one of the two processes to replace the process's memory space with a new program.

- **Wait()** Parent can issue a wait() system call to move itself off the ready queue until the termination of the child.
- **Exit ()** Process completes using the exit() system call
- When the child process completes the parent process resumes from the call to wait()

Process Termination

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit () system call.
- Process returns a status value to its parent process.
- All the resources of the process including physical and virtual memory, open files, and I/O buffers are de-allocated by the operating system.
- Parent process can terminate the child process with the help of system call.
- Users can use will command to terminate other users job.
- A parent may terminate the execution of one of its children for a variety of reasons like-
    - The child has exceeded its usage of some of the resources that it has been allocated.
    - The task assigned to the child is no longer required.
    - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

Inter process Communication

Two types of process are-

1. Independent process
2. Co-operating process

Independent process:

A process is **independent** if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.

Co-Operating Process:

A process is **cooperating** if **it** can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

Process Co-Operation is essential because of following reasons-

- **Information sharing:** Since several users may be interested in the same piece of information OS must support information sharing process.
- **Computation speedup:** Multiple processing elements can execute in parallel. It allows faster execution.
- **Modularity:** Dividing the system functions into separate processes or threads, results in modular structure. Different modules must be able to communicate with each other.
- **Convenience:** Even an individual user may work on many tasks at thesame time.

Co-Operating process require an inter process communication mechanism that allow then to exchange data and information.

Two fundamental models of inter process communication are-

- Shared memory
- Message passing

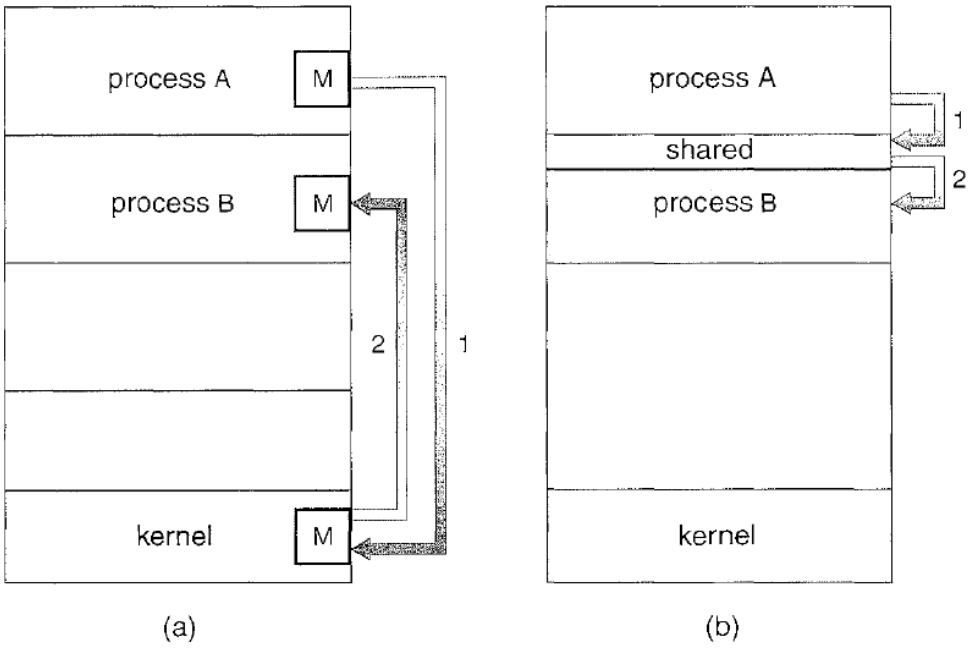


Figure 3.13 Communications models. (a) Message passing. (b) Shared memory.

Shared-Memory Model

In this a region of memory that is shared by Co-Operating process is established. Process can then exchange information by reading and writing data to the shared region.

Message-Passing Model

In this communication takes place by means of messages exchanged between the co-operating processes.

- Message passing is useful for exchanging smaller amounts of data
- Shared memory allows maximum speed
- Message passing system uses system call so kernel assistance is required. Shared memory system does not require it.

Shared-Memory Systems

- Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.
- Shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space.
- Let us consider the producer-consumer problem. A **producer** process produces information that is consumed by a **consumer** process.
- To allow producer and consumer process to run concurrently we must have a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized so that the consumer does not try to consume an item that has not yet been produced.

Message-Passing Systems

- Co-operating processes can communicate with each other via a message-passing system.
- It is useful in distributed environment Ex: *Chat Program* used in a world-wide lab
- A message-passing facility provides two operations-
    a) Send (message)
    b) Receive (message)

- A communication link must exist between a sending process and receiving process.
- Several methods used to implement a logical link are-
  - Direct or indirect communication
  - Synchronous or Asynchronous communication
  - Automatic or Explicit Buffering

Naming

- Processes that want to communicate can use-
- Direct Communication
- Indirect Communication
- In direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication
- In this scheme send() and receive() primitives are defined as-
  - Send (P, message)—Send a message to process P.
  - Receive (Q, message)—Receive a message from process Q.
- With indirect communication, the messages are sent to and received from mailboxes or ports
  - Send(A, message)—Send a message to mailbox A.
  - Receive(A, message)—Receive a message from mailbox A.
- Communication link in direct scheme has the following properties –
  - A link is established automatically between every pair of processes that want to communicate.
  - A link is associated with exactly two processes
- In indirect communication scheme, a communication link has following properties-
  - A link is established between a pair of processes only if both members of the pair have a shared mail box.
  - A link may be associated with more than two processes.

Synchronization

Message passing may be blocking an non-blocking, also known as synchronous and asynchronous.

- ***Blocking send:*** The sending process is blocked until the message is received by the receiving process or by the mailbox.
- ***Non-blocking:*** The sending process sends the message and resumes operation.
- ***Blocking receive:*** The receiver blocks until a message is available
- ***Non-Blocking Receive:*** The receiver retrieves either a valid message or a null

Buffering

Messages exchanged by communicating processes reside in a temporary queue. Such queues can be implemented by three ways-

- ***Zero capacity:*** In this queue has maximum length of zero. Sender must block until the recipient receives the message.
- ***Banded capacity:*** The queue has finite length n. so at most n messages can reside in it. If the queue is not full the message is placed in the queue.
- ***Unbounded capacity:*** the queue length is potentially infinite, any number of message can wait in it. The sender never blocks

Multithreaded Programming

Overview

A thread is a basic unit of CPU utilization; it comprises a thread ID, a programcounter, a register set, and a stack.

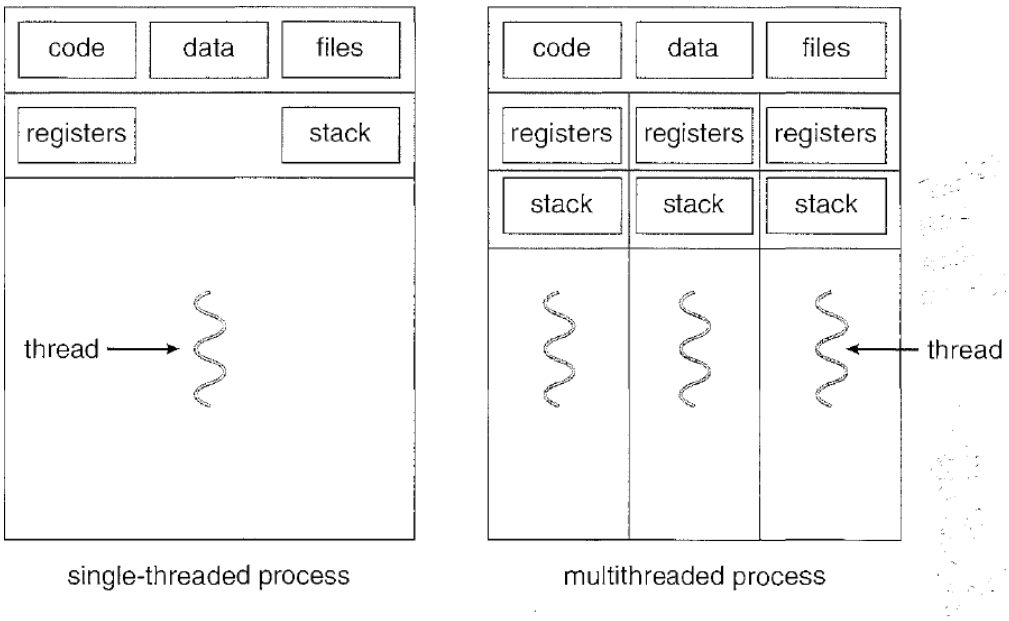


Figure 4.1 Single-threaded and multithreaded processes.

- A traditional (or **heavyweight)** process has a single thread of control.

- In web-server case it is more efficient to use one process that contains multiple threads. So, that the amount of time that a client have to wait for its request to be serviced will be less.
  - Threads also play a important role in remote procedure call.
- If a process has multiple threads of control, itcan perform more than one task at a time.

The benefits of multithreaded programming can be classified into--

- **Responsiveness:** Multithreading allow a program to continue running even if part of it is blocked or is performing a lengthy operation. Ex: multithreaded web browser could still allow user interaction
  - in one thread while an image was being loaded in another thread.
- **Resource sharing:**Threads share the memory and the resources of the process to which they belong.
- **Economy:** Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
- **Utilization of multiprocessor architectures:** A single threaded process can only run on one CPU irrespective of available CPU's Multithreading on a multi-CPU machine increases concurrency.

Multithreading Models

Support for threads may be provided either at the user level, for **user threads,** or by the kernel, for **kernel threads.**

Three ways of establishing connection between user threads and kernel threads are

Many-to-one Model:

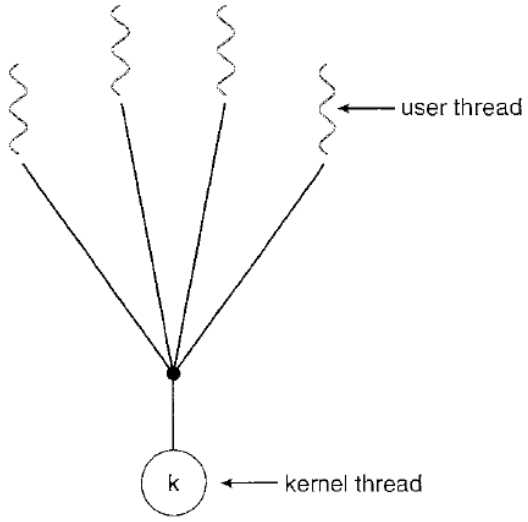


Figure 4.5 Many-to-one model.

- It maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient

- Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

One-to-One Model:

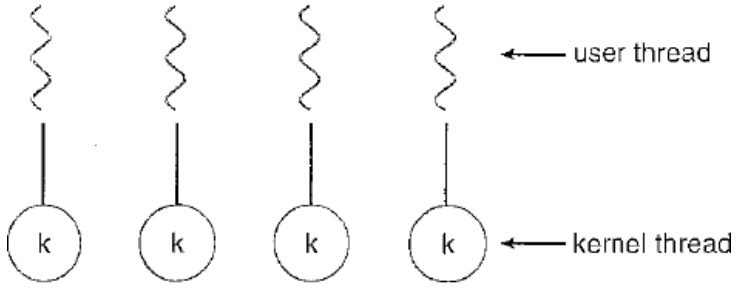


Figure 4.6 One-to-one model.

- The One-to-One model maps each user thread to a kernel thread.
- It allows multiple threads to run in parallel on multiprocessors.

- Drawback of this is model is that creating a user thread requires creating the corresponding kernel thread. Which effects the performance of an application.

Many-to-Many Model:

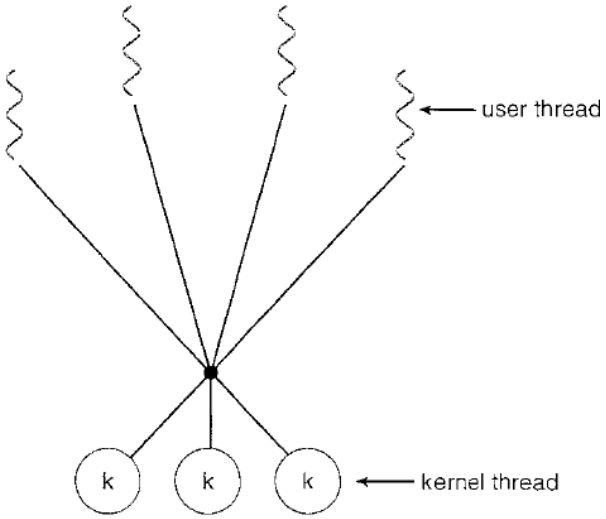


Figure 4.7 Many-to-many model.

- The Many-to-Many model multiplexes many user-level threads to smaller or equal number of kernel threads
- The number of kernel threads may be specific to either a particular application or a particular machine.

Thread Libraries

- **A thread library** provides the programmer an API for creating and managing threads.
  - There are two primary ways of implementing a thread library.
    - The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space.
    - The second approach is to implement a kernel-level library supported directly by the operating system.
  - Three main thread libraries are in use today:
    (1) POSIX threads
    (2) Win32
    (3) Java threads,
- POSIX standard threads or Pthreads may be provided as user or kernel level library
  - Win 32 thread library is a kernel-level library available on windows systems

- Java thread API allows threads creation and management directly in java programs.

Threading Issues

The fork() and exec() System Calls

Fork() system call is used to create a process. If a thread invokes the exec() system call, the program specified in the parameter to exec() will replace the entire process including all threads.

Cancellation

Thread cancellation is the task of terminating a thread before it has completed.
A thread that is to be cancelled is referred as target thread. It may occur in two cases

1. **Asynchronous cancellation:** One thread immediately terminates the target thread.
2. **Deferred cancellation:** The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Signal Handling

- A **signal** is used in UNIX systems to notify a process that a particular event has occurred.
- A signal may be received either synchronously or asynchronously.
- All signals follow the same pattern –
  - A signal is generated by the occurrence of a particular event.
  - A generated signal is delivered to a process.
  - Once delivered, the signal must be handled.
- Every signal may be *handled* by one of two possible handlers:
  1. A default signal handler
  2. A user-defined signal handler
- Signals may be handled in different ways some signals may simply be ignored, others may be handled by terminating the program.

Thread pools

- Idea behind a thread pool is to create a number of threads at process startup and place them into a *pool*, where they sit and wait for work.

- When a server receives a request, it awakens a thread from this pool- if oneis available
- Server passes the request to this thread
- Once the thread completes its service, it returns to the pool and awaits more work.
- If the pool contains no available thread, the server waits until one becomes free.
- Advantages of thread pools are-
    - Servicing a request with an existing thread is usually faster than waiting to create a thread.
    - A thread pool limits the number of threads that exist at any one point.

Thread Specific Data

Each thread might its own copy of certain data such data is called thread Specific data

Scheduler Activations

One scheme for communication between the user-thread library and the kernel is known as scheduler activation.

Process Scheduling

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

In multiprogramming systems several processes are kept in memory at one time. When one process has to wait, the OS takes the CPU away from that process and gives the CPU to another process.

CPU-I/O Burst Cycle

Process execution consists of a **cycle** of CPU execution and I/O wait.
Processes alternate between these two states.

Process execution begins with a CPU **burst.** That is followed by an **I/O burst,** which is followed by another CPU burst, then another I/O burst, and so on. Final CPU burst ends with a system request to terminate execution

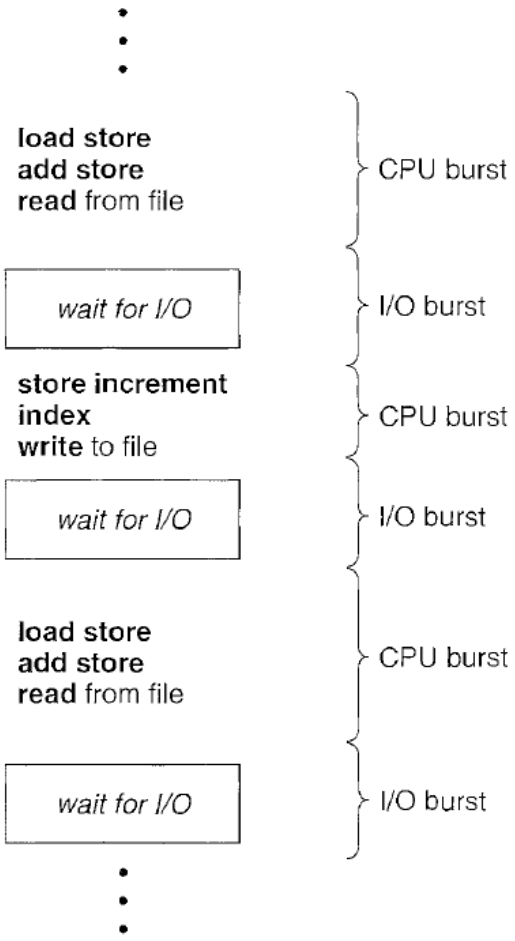


Figure 5.1 Alternating sequence of CPU and I/O bursts.

CPU Scheduler

CPU scheduler or short-term-scheduler selects a process from the processes available in memory that are ready to execute and allocates the CPU to that process.

Preemptive Scheduling and Non-Preemptive Scheduling

CPU-scheduling decisions may take place under the following cases

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state
4. When a process terminates

When scheduling takes place under 1st and 4th case it is called as non preemptive or Co-operative scheduling otherwise it is preemptive.

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

Dispatcher

Dispatcher gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

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

Time required for the dispatcher to stop one process and start another running is known as the **dispatch latency.**

Scheduling Criteria

Many criteria have been suggested for comparing CPU scheduling algorithms, they are

- **CPU utilization:** We want to keep the CPU as busy as possible. It can range from 0 to 100 percent.
- **Throughput:** Number of processes that are completed per time unit.
- **Turnaround time:** The interval from the time of submission of a process to the time of completion is the *turnaround time*. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time:** It is the sum of the periods spent waiting in the ready queue.
- **Response time:** It is the amount of time from the submission of a request until the first response is produced.

Scheduling Algorithms

CPU scheduling deals with the problem of deciding which of the processesin the ready queue is to be allocated the CPU.

First-Come, First-Served Scheduling

- It is the simplest CPU-scheduling algorithm.

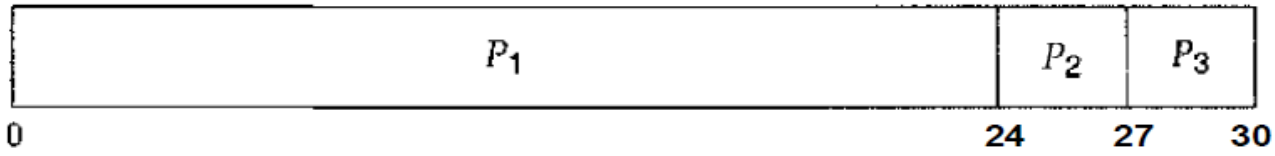
- With this method, the process that requests the CPU first is allocated the CPU first.
- Average waiting time under the FCS policy is quite long.

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

Process arrive in order P_1, P_2, P_3

Gantt Chart



Waiting time for $P_1 = 0$ *ms*

Waiting time for $P_2 = 24$ *ms*

Waiting time for $P_3 = 27$ *ms*

Average waiting time $= \frac{0 + 24 + 27}{3} ms$

$= \frac{51}{3} ms$

$= 17$ *ms*

Turnaround time =Burst time + waiting time

Turnaround time for $P_1 = 24 + 0$ *ms*

Turnaround time for $P_2 = 3 + 24$ *ms*

Turnaround time for $P_3 = 3 + 27$ *ms*

Average turnaround time $= \frac{24+27+30}{3} ms$

$= \frac{81}{3} ms$

$= 27 ms$

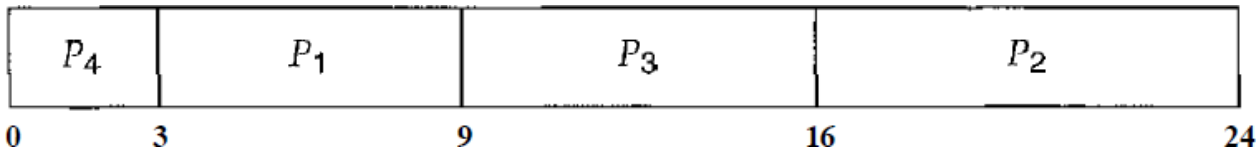
Shortest-Job-First Scheduling

(shortest-job-first (SJF) schedulingalgorithm.)

This algorithm associates with each process the length of theprocess's next CPU burst. When the CPU is available, it is assigned to the processthat has the smallest next CPU burst.

Here scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Process	**Burst Time**
P_1	6
P_2	8
P_3	7
P_4	3



Waiting time for Process P_1 = 3 *ms*

Waiting time for Process P_2 = 16 *ms*

Waiting time for Process P_3 = 9 *ms*

Waiting time for Process P_4 = 0 *ms*

Average waiting time = $\frac{03+16+9+0}{4} ms$

$= \frac{28}{4} ms$

$= 7\ ms$

Difficulty in SJF is knowing the length of the next CPU request SJF algorithm can be either preemptive or non-preemptive.

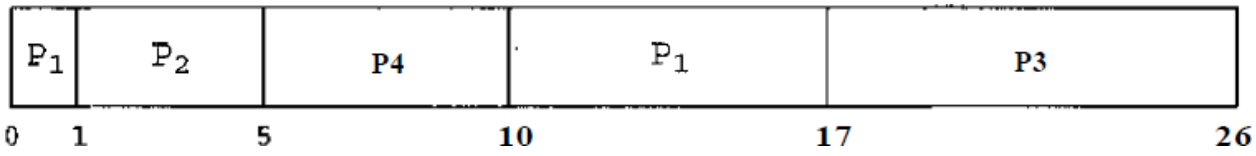
Preemptive SJF (Shortest-remaining time-first-scheduling): Whenever a new process arrives in the system the next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process. A preemptive SJF will preempt the currently executing process.

Non Preemptive SJF: Non Preemptive SJF will allow the currently running process to finish its CPU burst.

Preemptive SJF problem

Process	Arrival Time	SJF Problem
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

If the process arrive at the times shown then preemptive SJF is given like



Waiting time for P1 = (10 – 1)ms

=(1 - 1) ms

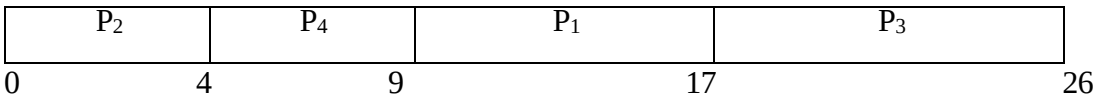
(P2 is arrived at 1ms →check arrival time CPU is allocated immediately)

P3 =(17 – 2) ms

P4 =(5 – 3)ms

Average waiting time = $\frac{9+0+15+2}{4}$ = 6.5 ms

Non Preemptive



Waiting time for P_1 = 9 ms

Waiting time for P_2 = 0 ms

Waiting time for P_3 = 17 ms

Waiting time for P_4 = 4 ms

Average waiting time = $\frac{9+0+14+4}{4}$ ms

$=\frac{30}{4}$

=7.5 ms

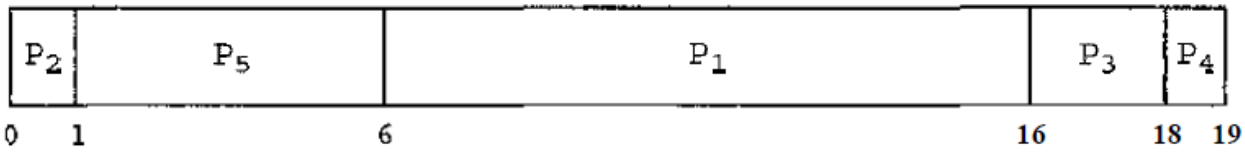
Priority Scheduling

In this a priority is associated with each process, and the CPU is allocated to the process with the highest priority.

Priority is indicated by fixed range of numbers

Assume low numbers represent high priority

Process	Arrival Time	SJF Problem
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_4	5	2



Waiting time for P_1 = 6 ms

Waiting time for P_2 = 0 ms

Waiting time for P_3 = 16 ms

Waiting time for P_4 = 18ms

Waiting time for P_5 = 1ms

Average waiting time = $\frac{6+0+16+18+1}{5}$ ms

$= \frac{41}{4}$

= 8.2ms

Major problem with priority scheduling is indefinite blocking or starvation .

A priority scheduling algorithm can leave some low priority processes waiting indefinitely.

A solution to the problem o indefinite blockage of low-priority process is aging.

Aging is the technique of gradually increasing the priority of processes that wait in the system for a long time.

Round-Robin Scheduling

Round-Robin Scheduling is designed for time-sharing systems. A small unit of time, called a time quantum or time slice is defined.

CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

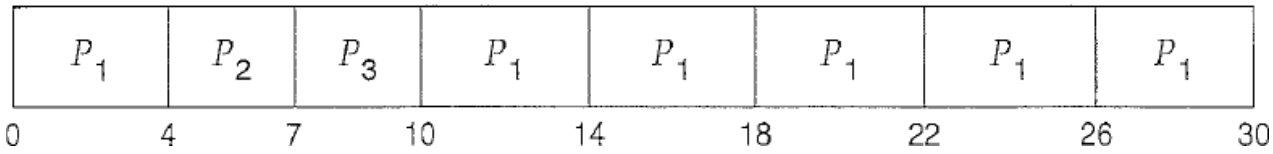
In RR scheduling if a process's CPU burst exceeds 1 time quantum that process is preempted and put back in the ready queue.

Therefore, RR scheduling is preemptive

Consider the process

P_1	24
P_2	3
P_3	3

Time quantum is 4*ms*



Waiting time for $P_1 \rightarrow 10 - 4 = 6$ *ms*

Waiting time for $P_2 \rightarrow = 4$ *ms*

Waiting time for $P_3 \rightarrow = 7$ *ms*

Average waiting time $= \frac{6+4+7}{3} ms$

$= \frac{17}{3}$ □□

$= 5.66$ *ms*

Multilevel Queue Scheduling

- This is used for the processes which can be classified into different groups
- A multilevel Queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue based on some priority.
- Each queue has its own scheduling algorithm

Ex: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.

- There must be scheduling among the queues.
- Ex: Five queues are listed below in the order of priority.
    - System processes
    - Interactive processes
    - Interactive editing processes
    - Batch processes
    - Student processes

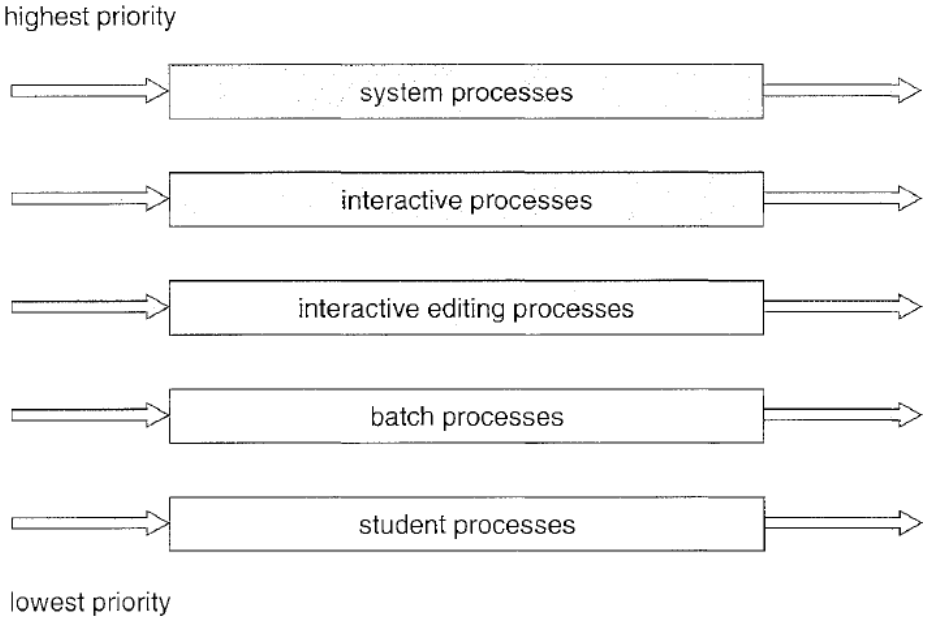


Figure 5.6 Multilevel queue scheduling.

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

Multilevel Feedback-Queue Scheduling

- This allows a process to move between queues
- If a process uses too much CPU time it will be moved to a lower-priority queue.
- This scheme leaves I/O bound and interactive process in the higher-priority queues.

- In this scheme process that waits too long in a lower-priority queue may be moved to a higher-priority queue.
- Example: consider a multilevel feedback-queue scheduler with three queues numbered from 0 to 2

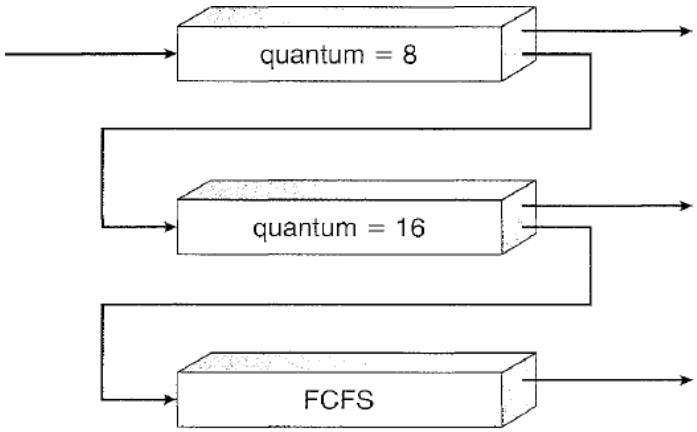


Figure 5.7 Multilevel feedback queues.

- The scheduler first executes all processes in queue 0. Only when queue 0 is empty it will execute process in queue 1.
- Similarly process in queue 2 will be executed if queues 0 and 1 are empty
- A process that arrives for queue 1 will preempt a process in queue 2
- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of the queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.

Multiple-Processor Scheduling

If multiple CPU's are available load sharing becomes possible. We concentrate on **Homogeneous** systems. i.e.., processors are identical. Approaches to Multiple-Processor Scheduling **Asymmetric multiprocessing**

- All scheduling decisions are handled by a single processor the master sever. Other processors execute only user code.
- Here only one processor accesses the system data structures.

Symmetric multiprocessing

- Here each processor is self-scheduling
- Scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity

Because of the high cost, most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor.

Processor affinity, meaning that a process has an affinity for the processor on which it is currently running.

Load Balancing

Load Balancing attempts to keep the workload evenly distributed across all processors in an SMP system.

Two general approaches are

- ***Push migration:*** A specific task checks the load on each processor. In the case of imbalance it evenly distributes the load
- ***Pull migration:*** It occurs when an idle processor pulls a waiting task from a busy processor

Symmetric Multithreading

- It providing multiple physical processors to allow several threads to run concurrently (symmetric multiprocessing provides multiple physical processors)
- It is also called as hyper threading technology
- SMT is a feature provided in hardware not in s/w

- Each logical processor has its own architecture state, which includes general purpose and machine-state registers.
- Each logical processor is responsible for its own interrupt handling

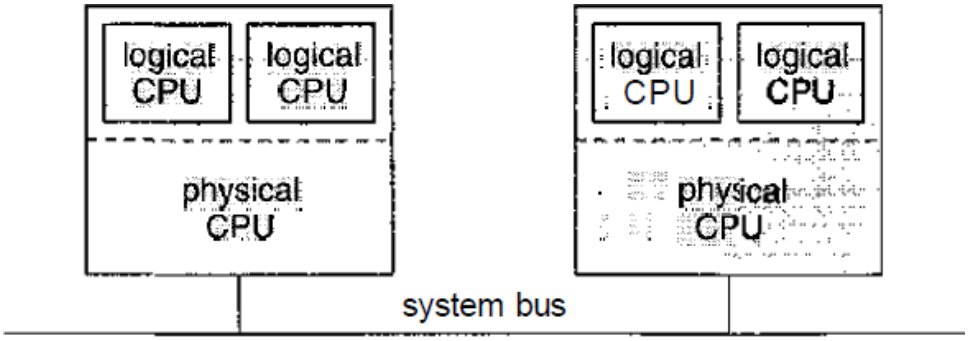


Figure 5.8 A typical SMT architecture

Thread Scheduling

Contention Scope

One distinction between user-level and kernel-level threads lies in how they are scheduled. On systems implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on an available LWP, a scheme known as **process-contention scope (PCS),** since competition for the CPU takes place among threads belonging to the same process.

Os must schedule the kernel thread onto a physical CPU. To decide which kernel thread to schedule onto a CPU, the kernel uses **system-contention scope** (SCS).

Competition for the CPU with SCS scheduling takes place among all threads in the system. Systems using the one-to-one model (such as Windows XP, Solaris 9, and Linux) schedule threads using only SCS.

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- Homogeneous processors within a multiprocessor.
- Load sharing

- Asymmetric multiprocessing – only one processor accesses the systemdata structures, alleviating the need for data sharing.
- Hard real-time systems – required to complete a critical task within a guaranteed amount of time.
- Soft real-time computing – requires that critical processes receive priority over less fortunate ones.

Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines theperformance of each algorithm for that workload.
- Queuing models
- Implementation

Evaluation of CPU Schedulers by Simulation

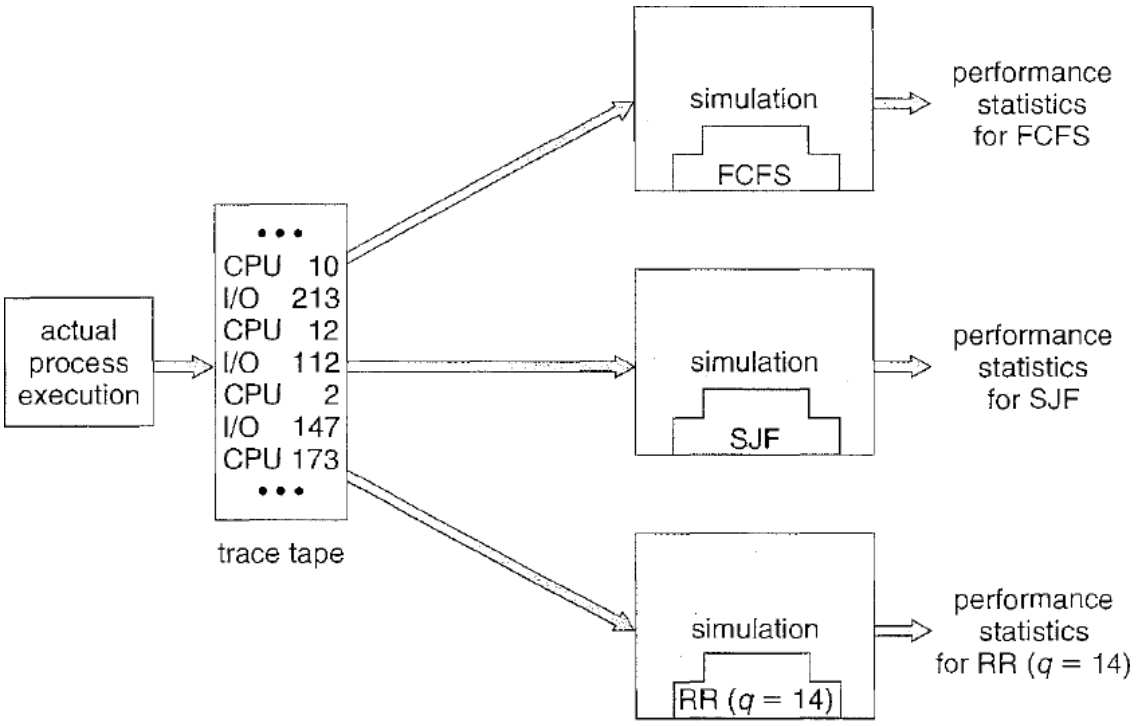


Figure 5.17 Evaluation of CPU schedulers by simulation.

Assignment Questions

1. Differentiate between ling term and short term schedulers (4) Dec 08/ Jan 09
2. Suppose following jobs arrive, each job runs the listed amount of time.

1. Job	2. 1	3. 2	4. 3
5. Arrival Time	6. 0.0	7. 0.4	8. 1.0
9. Burst Time	10. 8	11. 4	12. 1

   i. give grant chart using non preemptive FCFS and SJF alg.

   ii .what is turn around and waiting time of each job?

   iii. compute average turn around time if CPU is idle for first 1 unit and then SJF is used.

3. Define IPC.what are different methods used for logical implementations of messagepassing systems. (6) Dec 2010.
4. Discuss common ways of establishing relationship between user and kernel thread(6) Dec 2010.
5. Draw grant chart using preemptive SJF. Find avg waiting time for following data. (8)Dec 2010.

Process	13. P_1	14. P_2	15. P_3	16. P_4	17. P_5
Arrival Time	18. 0	19. 2	20. 3	21. 6	22. 30
Burst Time	23. 10	24. 12	25. 14	26. 16	27. 5

6. What is process? Draw and explain process state diagram. Explain PCB structure. (10) July 2011
7. Consider 4 jobs with (AT and BT) as (0,5), (0.2,2) (.6,8) (1.2,4). Find avgturnaround time and waiting time using FCGS, SJF and Rp(q=1). (10) July2011
8. What is the meaning of the term *busy waiting*?
9. Explain semaphores with the help of an example.
10. Explain dining philosopher's problem, and how it is solved.
11. What is race condition? Explain how it is handled.
12. Define process synchronization.
13. How is producer-consumer problem with the help of semaphores?

Outcome

- Implement different process scheduling algorithms.
- Familiarized with inter process communication and multi-threaded programming.
- Familiarized with semaphores
- Solve critical section problem
- Implement synchronization
- Implement Peterson's solution.

Further Reading

11. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne: Operating System Principles, 8th edition, Wiley India, 2009. (Listed topics only from Chapters 1 to 12, 17, 21)
12. D.M Dhamdhere: Operating systems - A concept based Approach, 2nd Edition, Tata McGraw- Hill, 2002.
13. P.C.P. Bhatt: Introduction to Operating Systems: Concepts and Practice, 2nd Edition, PHI, 2008.
14. Harvey M Deital: Operating systems, 3rd Edition, Pearson Education, 1990.
15. http://nptel.ac.in/courses/106106144
16. https://en.wikipedia.org/wiki/Operating_system
17. https://www.tutorialspoint.com/operating_system
18. http://www.studytonight.com/operating-system
19. https://www.linux.com
20. https://opensource.com

MODULE 3
Process Synchronization

Process Synchronization

Synchronization

The critical section problem

Peterson's solution

synchronization hardware

semaphores

classical problems of synchronization monitors

Deadlocks

System model

Deadlock characterization

Methods for handling deadlocks

Deadlock prevention

Deadlock avoidance

Deadlock detection and recovery from

deadlock

ASSIGNMENT QUESTION

OUTCOME

FURTHER READING

Introduction

This unit gives the overview on deadlocks and deadlock characterization. Methods for handling deadlocks are discussed. Methods of deadlock detection, prevention and avoidances are highlighted. This unit gives overview of swapping and memory allocations techniques. Paging, structure of page table and segmentation is discussed in detail.

Objective

- Understand what is deadlock.
- Understand the techniques for deadlock detection, prevention and avoidance.
- Understand the techniques to recover from deadlocks.
- Understand swapping
- Understand segmentation, contiguous memory allocation

SYNCHRONIZATION

Since processes frequently need to communicate with other processes therefore, there is a need for a well-structured communication, without using interrupts, among processes.

Race Conditions

In operating systems, processes that are working together share some common storage (main memory, file etc.) that each process can read and write. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions. Concurrently

executing threads that share data need to synchronize their operations and processing in order to avoid race condition on shared data. Only one 'customer' thread at a time should be allowed to examine and update the shared variable. Race conditions are also possible in Operating Systems.

If the ready queue is implemented as a linked list and if the ready queue is being manipulated during the handling of an interrupt, then interrupts must be disabled to prevent another interrupt before the first one completes. If interrupts are not disabled than the linked list could become corrupt.

1. count++ could be
   implemented as
   register1 =
   count register
   1 = register 1 + 1
   count =register 1

2.count--could be implemented
as register
2 = count register
2 = register2 – 1 count = register2

3.Consider this execution interleaving with "count = 5" initially:

S0: producer execute register1 = count {register1 = 5}
S1: producer execute register1 = register1 + 1 {register1 = 6}
S2: consumer execute register2 = count {register2 = 5}
S3: consumer execute register2 = register2 -1 {register2 = 4}
S4: producer execute count = register1 {count = 6 }
S5: consumer execute count = register2 {count = 4}.

THE CRITICAL SECTION PROBLEM

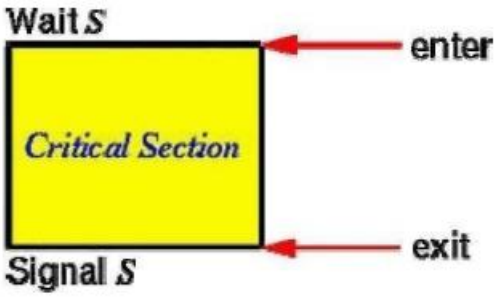
Mutual Exclusion -If process Pi is executing in its critical section, then no other processes can be executing in their critical sections

1. Progress -If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

2.Bounded Waiting -A bound must exist 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 criticalsection and before that request is granted

• Assume that each process executes at a nonzero speed
• No assumption concerning relative speed of the N processes

A. Critical Section



The key to preventing trouble involving shared storage is find some way to prohibit more than one process from reading and writing the shared data simultaneously. That part of the program where the shared memory is accessed is called the Critical Section. To avoid race conditions and flawed results, one must identify codes in Critical Sections in each thread. The characteristic properties of the code that form a Critical Section are Codes that reference one or more variables in a “read-update-write” fashion while any of those variables is possibly being altered by another thread. x Codes that alter one or more variables that are possibly being referenced in “read-updata-write” fashion by another thread. x Codes use a data structure while any part of it is possibly being altered by another thread. x Codes alter any part of a data structure while it is possibly in use by another thread.Here, the important point is that when one process is executing shared modifiable data in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time.

3. MUTUAL EXCLUSION

A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing. Formally, while one process executes the shared variable, all other

processes desiring to do so at the same time moment should be kept waiting; when that process has

finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

Note that mutual exclusion needs to be enforced only when processes access shared modifiable data when processes are performing operations that do not conflict with one another they should be allowed to proceed concurrently.

Mutual Exclusion Conditions

If we could arrange matters such that no two processes were ever in their critical sections simultaneously, we could avoid race conditions. We need four conditions to hold to have a good solution for the critical section problem (mutual exclusion).

- No two processes may at the same moment inside their critical sections.

No assumptions are made about relative speeds of processes or number of CPUs.

- No process should outside its critical section should block other processes.
- No process should wait arbitrary long to enter its critical section.

PETERSON'S SOLUTION

The mutual exclusion problem is to devise a pre-protocol (or entry protocol) and a postprotocol (or exist protocol) to keep two or more threads from being in their critical sections at the same time. Tanenbaum examine proposals for critical-section problem or mutual exclusion problem.

Problem

When one process is updating shared modifiable data in its critical section, no other process should allowed to enter in its critical section.

Proposal 1 -Disabling Interrupts (Hardware Solution)

Each process disables all interrupts just after entering in its critical section and re-enable all interrupts just before leaving critical section. With interrupts turned off the CPU could not be switched to other process. Hence, no other process will enter its critical and mutual exclusion achieved.

Conclusion

Disabling interrupts is sometimes a useful interrupts is sometimes a useful technique within the kernel of an operating system, but it is not appropriate as a general mutual exclusion mechanism for users process. The reason is that it is unwise to give user process the power to turn off interrupts.

Proposal 2 -Lock Variable (Software Solution)

In this solution, we consider a single, shared, (lock) variable, initially 0. When a process wants to enter in its critical section, it first test the lock. If lock is 0, the process first sets it to 1

and then enters the critical section. If the lock is already 1, the process just waits until (lock)

variable becomes 0. Thus, a 0means that no process in its critical section, and 1 means hold your horses -some process is in its critical section.

Conclusion

The flaw in this proposal can be best explained by example. Suppose process A sees that the lock is 0. Before it can set the lock to 1 another process B is scheduled, runs, and sets the lock to 1. When the process A runs again, it will also set the lock to 1, and two processes will be in their critical section simultaneously.

Proposal 3 -Strict Alteration

In this proposed solution, the integer variable 'turn' keeps track of whose turn is to enter the critical section. Initially, process A inspect turn, finds it to be 0, and enters in its critical section. Process B also finds it to be 0 and sits in a loop continually testing 'turn' to see when it becomes 1.Continuously testing a variable waiting for some value to appear is called the *Busy-Waiting*.

Conclusion

Taking turns is not a good idea when one of the processes is much slower than the other. Suppose process 0 finishes its critical section quickly, so both processes are now in their noncritical section. This situation violates above mentioned condition 3.

Using Systems calls 'sleep' and 'wakeup'

Basically, what above mentioned solution do is this: when a processes wants to enter in its critical section , it checks to see if then entry is allowed. If it is not, the process goes into tight loop and waits (i.e., start busy waiting) until it is allowed to enter. This approach waste CPU-time.

Now look at some interprocess communication primitives is the pair of steep-wakeup.

Sleep

- It is a system call that causes the caller to block, that is, be suspended until some other process wakes it up. xWakeup
- It is a system call that wakes up the process.

Both 'sleep' and 'wakeup' system calls have one parameter that represents a memory address used to match up 'sleeps' and 'wakeups' .

The Bounded Buffer Producers and Consumers

The bounded buffer producers and consumers assumes that there is a fixed buffer size i.e., a finite numbers of slots are available

STATEMENT

To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions or lost updates. As an example how sleep-wakeup system calls are used, consider the producer-consumer problem also knownas bounded buffer problem. Two processes share a common, fixed-size (bounded) buffer. The producer puts information into the buffer and the consumer takes information out.

Trouble arises when

1. The producer wants to put a new data in the buffer, but buffer is already full. Solution: Producer goes to sleep and to be awakened when the consumer has removed data.

2. The consumer wants to remove data the buffer but buffer is already empty. Solution: Consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

Conclusion

This approaches also leads to same race conditions we have seen in earlier approaches. Race condition can occur due to the fact that access to 'count' is unconstrained. The essence of the problem is that a wakeup call, sent to a process that is not sleeping, is lost.

SYNCHRONIZATION HARDWARE

1. Many systems provide hardware support for critical section code

2. Uniprocessors – could disable interrupts

- Currently running code would execute without preemption
- Generally too inefficient on multiprocessor systems
- Operating systems using this not broadly scalable
- 3. Modern machines provide special atomic hardware

instructions Atomic = non-interruptable

- Either test memory word and set value
- Or swap contents of two memory words

SEMAPHORES

E.W. Dijkstra (1965) abstracted the key notion of mutual exclusion in his concepts of semaphores.

Definition

A semaphore is a protected variable whose value can be accessed and altered only by the operations P and V and initialization operation called 'Semaphoiinitislize'.Binary Semaphores can assume only the value 0 or the value 1 counting semaphores also called general semaphores can assume only nonnegative values. The P (or wait or sleep or down) operation on semaphores S, written as P(S) or wait (S), operates as follows:

P(S):**IF**S>0

THEN S:= S-1

ELSE (wait on S)

The V (or signal or wakeup or up) operation on semaphore S, written as V(S) or signal (S), operates as follows:

V(S):**IF**(one or more process are waiting on S)

THEN (let one of these processes

proceed) **ELSE** S := S +1

Operations P and V are done as single, indivisible, atomic action. It is guaranteed that once a semaphore operations has stared, no other process can access the semaphore until operation has completed. Mutual exclusion on the semaphore, S, is enforced within **P(S)** and **V(S)**.

If several processes attempt a P(S) simultaneously, only process will be allowed to proceed. The other processes will be kept waiting, but the implementation of P and V

guarantees that processes will not suffer indefinite postponement. Semaphores solve the lost-wakeup problem.

Semaphore as General Synchronization Tool

1. Counting semaphore – integer value can range over an unrestricted domain.

2. Binary semaphore – integer value can range only between and 1; can be simpler to implement Also known as mutex locks.

3. Can implement a counting semaphore S as a binary semaphore.
4. Provides mutual exclusion

- Semaphore S; // initialized to 1
- wait (S);

Critical Section

signal (S);

Semaphore Implementation

1. Must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time
2. Thus, implementation becomes the critical section problem where the wait and signal code are placed in the crtical section. Could now have busy waiting in critical section implementation

- But implementation code is short
- Little busy waiting if critical section rarely occupied
- 3. Note that applications may spend lots of time in critical sections and therefore this is not a good solution.
- Operations P and V are done as single, indivisible, atomic action. It is guaranteed that once a semaphore operations has stared, no other process can access the semaphore until operation has completed. Mutual exclusion on the semaphore, S, is enforced within **P(S)** and **V(S)**.
- If several processes attempt a P(S) simultaneously, only process will be allowed to proceed. The other processes will be kept waiting, but the implementation of P and V guarantees that processes will not suffer indefinite postponement. Semaphores solve the lost-wakeup problem

- If we could arrange matters such that no two processes were ever in their critical sections simultaneously, we could avoid race conditions. We need four conditions to hold to have a good solution for the critical section problem (mutual exclusion).
- xNo two processes may at the same moment inside their critical sections.
- xNo assumptions are made about relative speeds of processes or number of CPUs
- xNo process should outside its critical section should block other processes.
- x No process should wait arbitrary long to enter its critical section
- Note that mutual exclusion needs to be enforced only when processes access shared modifiable data when processes are performing operations that do not conflict with one another they should be allowed to proceed concurrently.

- **Semaphore Implementation with no Busy waiting**

1. With each semaphore there is an associated waiting queue. Each entry in a waiting queue has two data items:

value (of type integer)

pointer to next record in the list

2. Two operations:

block – place the process invoking the operation on the appropriate waiting queue.

wakeup – remove one of processes in the waiting queue and place it in the ready queue. -Implementation of wait:

```
wait (S)
{  value--;
        if (value < 0) { add this process to waiting queue
            block(); }
}
```

->Implementation of signal:

```
Signal (S){

    value++;

    if (value <= 0) {

        remove a process P from the waiting queue

    wakeup(P); }
```

CLASSICAL PROBLEMS OF SYNCHRONIZATION

1. Bounded-Buffer Problem
2. Readers and Writers Problem
3. Dining-Philosophers Problem

Bounded-Buffer Problem

1. *N buffers, each can hold one item*
2. Semaphore mutex initialized to the value 1
3. Semaphore full initialized to the value 0
4. Semaphore empty initialized to the value N.
5. The structure of the producer process while (true) {

```
        // produce an item wait (empty); wait (mutex);

        // add the item to the buffer signal (mutex); signal (full);

}
```

6. The structure of the consumer process

```
while (true) { wait (full); wait

(mutex);

        // remove an item from buffer signal (mutex); signal (empty);

        // consume the removed item }
```

Readers-Writers Problem

1. A data set is shared among a number of concurrent processes

. ○ Readers–only read the data set; they do not perform any updates

. ○ Writers–can both read and write.

2. Problem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time.
3. Shared Data

· ○ Data set

· ○ Semaphore mutex initialized to 1.

· ○ Semaphore wrt initialized to 1.

· ○ Integer readcount initialized to 0.

4. The structure of a writer process while (true) { wait (wrt) ; // writing is performed

signal (wrt) ; }

5. The structure of a reader process

while (true) { wait (mutex) ; readcount ++ ; if (readcount == 1) wait (wrt) ; signal (mutex

// rea ding is

performed wait

(mute x) ;

readcount -

-;

if (readcount== 0) signal (wrt) ; signal (mutex) ;

}

Dining-Philosophers Problem

1. Shared data

. ○ Bowl of rice (data set)

. ○ Semaphore chopstick [5] initialized to 1

2. The structure of Philosopher i: While (true)

{ wait (chopstick[i]); wait (chopStick[

(i + 1) % 5]);

// eat signal (chopstick[i]); signal (chopstick[(i + 1) % 5]); // think

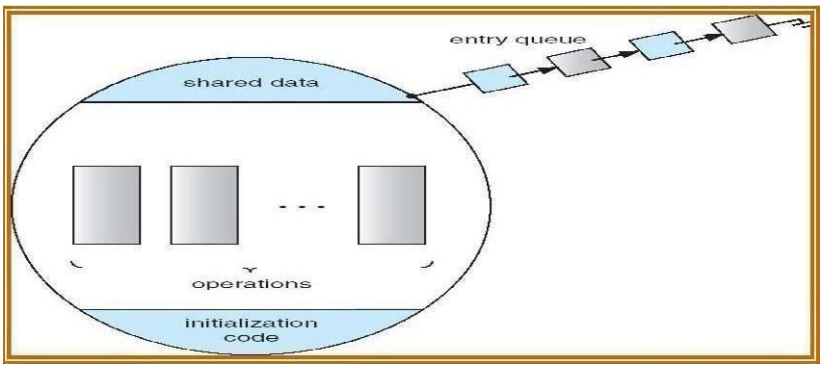
}

Problems with Semaphores

1. Correct use of semaphore operations:
2. . ○ signal (mutex**) ….** wait (mutex)
3. . ○ **wait (mutex) … wa**it (mutex)

4. . ○ Omitting of wait (mutex) or signal (mutex) (or both)

MONITORS



1. high-level abstraction that provides a convenient and effective mechanism for process synchronization
2. Only one process may be active within the monitor at a

time monitor monitor-name { // shared variable

declarations procedure **P1 (…) { ….** }

… **procedurePn (…) {……}**

Initialization code (….) { … } … }

Solution to Dining Philosophers

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 left and

right neighbors 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;
} ->Each philosopher *I* invokes the operations pickup() and putdown() in the

following sequence: dp.pickup (i) EAT dp.putdown (i)

Monitor Implementation Using Semaphores

1. Variables semaphore mutex; // (initially = 1) semaphore next; // (initially = 0) int next-count =

0;

2. Each procedure ***F*** will be replaced by **wait(mutex); …**

body**of F; … if (next**-count > 0

signal(next) else signal(mutex);

1. Mutual exclusion within a monitor is ensured.
2. For each condition variable ***x***, we have: semaphore x-sem; // (initially = 0) int x-count = 0;
3. The operation x.wait can be implemented as: x-count++; if (next-count > 0)

signal(next); else

signal(mutex); wait(x-sem); x-count--;

6. The operation x.signal can be implemented as:

if (x-count > 0) { next-count++; signal(x-sem); wait(next); next-count--;

}

Producer-Consumer Problem Using Semaphores

The Solution to producer-consumer problem uses three semaphores, namely, full, empty and mutex.

The semaphore 'full' is used for counting the number of slots in the buffer that are full. The 'empty' for counting the number of slots that are empty and semaphore 'mutex' to make sure that the producer and consumer do not access modifiable shared section of the buffer simultaneously.

Initialization

xSet full buffer slots to 0. i.e., semaphore Full = 0.xSet empty buffer slots to N. i.e., semaphore empty = N. x Forcontrol access to critical section set mutex to 1. i.e., semaphore mutex = 1.

Producer ( ) WHILE (true) produce-Item ( ); P (empty); P (mutex); enter-Item ( ) V

(mutex) V (full);

Consumer ( )

WHILE (true) P (full) P (mutex); remove-Item ( ); V (mutex); V (empty); consume-Item (Item)
}

DEADLOCK

In a multiprogramming system, several processes may compete for a finite number of resources. If resources are not available process enters into waiting state. Sometimes waiting process is never again able to change the state, because the resources it has requested are held by other waiting processes, this situation is called a deadlock.

SYSTEM MODEL

A system consists of a finite number of resources to be distributed among a number of competing processes.

A process must request a resource before using it and must release the resource after using it. The number of resources requested may not exceed the total number of resources available in the system.

Process may utilize a resource in the following sequence:-

- ***Request*** 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.
- ***Release*** the process releases the resource.

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.

DEADLOCK CHARACTERIZATION

NECESSARY CONDITIONS

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

- ***Mutual exclusion***

  At least one resource must be held in a non-sharable mode, i.e., only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

- ***Hold and wait***

A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

- ***No preemption***

  Resources cannot be preempted, i.e., a resource can be released only voluntarily by the process holding it, after that process has completed its task.

- ***Circular wait.***

  A set {P_0, P_1..., Pn }of waiting processes must exist suchthat $P_0$is waiting for a resource held by P_1, P_1 is waiting for a resourceheld by P_2, $Pn-1$,is waiting for a resource held by Pn, and $Pn$is waitingfor a resource held by Pn.

  All four conditions must hold for a deadlock tooccur..

Resource-Allocation Graph

- Deadlocks can be described in terms of a directed graph calleda **system resource-allocation** graph.
- Graph consists of a set of vertices $V$and a set of edges E. The set of vertices V is partitioned into two different typesof nodes: P = {P_0, P_1..., Pn }the set consisting of all the active processes in thesystem, and R = {R_1, R_2, ...Rm}, the set consisting of all resource types in thesystem.
- A directed edge from process $P_i$to resource type $R_j$is denoted by $P_i \rightarrow R_j$, it signifies that process P_i has requested an instance of resource type R_j, andis currently waiting for that resource. It is called as ***Request edge***
- A directed edge from resource type $R_j$to process P_i is denoted by $R_j \rightarrow P_i$ signifies that are instance of resource type $R_j$has been allocated to process P_i . It is called as ***assignment edge.***
- Process P_i is represented as a circle and each resource type$R_j$as a rectangle.
- Since resource type $R_j$may have more than one instance, werepresent each such instance as a dot within the rectangle.
- Requestedge points to only the rectangle Rj, whereas an assignment edge points to one of the dots within the rectangle.
- Resource allocation graph in the below figure denotes

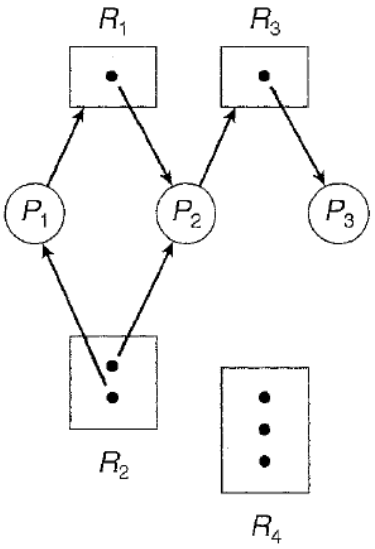


Figure 7.2 Resource-allocation graph.

- The sets *P,* K and *E:*

  $P = \{P_1, P_2, P_3\}$

  $R = \{R_1, R_2, R_3, R_4\}$

  $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

- Resource instance

  $R_1 \rightarrow$ 1 Instance

  $R_2 \rightarrow$ 2Instance

  $R_3 \rightarrow$ 3Instance

  $R_4 \rightarrow$ 4Instance

- 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_2 and an instance of R_1 and iswaiting for an instance of R_3.
  - Process P_3 is holding an instance of R_3.
- If resource-allocation graphcontains no cycles, then no process in the system is deadlocked. Ifthe graph contain a cycle, then a deadlock may exist.
- If the cycle involves set of resource with single instance, then each processinvolved in the cycle is deadlocked. Cycle in the graph is both necessary & sufficient condition for the existence of a deadlock.

- If each resource type has several instance then cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.
- If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is cycle then the system may or may not be in a deadlock state.

Methods for Handling Deadlocks

Three ways used to deal with deadlock problem are:

- ***Protocol*** is used to prevent or avoid deadlocks, ensuring that the system will *never enter* a deadlock state.
- System is allowed to enter a deadlock state, later it is *detected* and *recovered.*
- Problem is *ignored* altogether and pretended that deadlocks never occurred in the system.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock avoidance scheme.

Deadlock Prevention

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

- ***Mutual exclusion***

  Mutual exclusion condition must hold for non-sharable resources. We cannot prevent deadlock by denying the mutual exclusion condition, because some resources are intrinsically non-sharable.

- ***Hold and wait***

  To avoid hold and wait condition the operating system must ensure that, whenever a process requests a resources it will not hold any other resource simultaneously.

  Two possible solutions used to achieve this are:-

  - Whenever a process requests resources, all the resources are allocated to it before it executes.

- Allow a process to request resources only when it does not have any resources. i.e., whenever a process gets some resources it should immediately use them and before making additional request it must release all the resources currently held by it.

- ***No preemption***

  If a process is holding some resources and requests for other resource that cannot be met immediately then all resources held by that process will be preempted.

  The preempted resources are automatically added to the list of free resources. The process may restart only when it gets the old resources and new ones that are required.

- ***Circular wait***

  One way to ensure that this condition never holds is to impose a total ordering of all resources types and to require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

Avoiding deadlocks requires additionalinformation about how resources are to be requested. The system will consider resourcescurrently available, the resources currently allocated to each process, and thefuture requests and releases of each process.

A deadlock-avoidance algorithmexamines the resource-allocation state to ensure that a circularwaitcondition can never exist. The resource-allocation *state* is defined by thenumber of available and allocated resources and the maximum demands ofthe processes.

Safe State

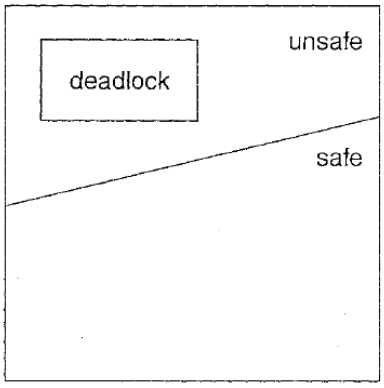


Figure 7.5 Safe, unsafe, and deadlocked state spaces.

- A state is *safe* if the system can allocate resources to each process in some order and still avoid a deadlock. A systemis in a safe state only if there exists a safe sequence.
- A sequence of processes *{P_0, P_1 ..., Pn }* is a safe sequence for the current allocation state if, for each P_i, the resource requests that P_i, can still make can be satisfied by the currently available resources plus the resources held by all P_i , with ***j < i.***
- If no such sequence exists, then the system state is said to be *unsafe.*
- A safe state is not a deadlocked state.
- An unsafe state *may* lead to a deadlock.
- With the concept of safe state we can define avoidance algorithm that ensure that the system will never deadlock.
- Example for safe state

System consist of three processes P_1 , P_2 and P_3 and one resource R_1. Number of units for R_1 is 12. Consider following

	MAXIMUM NEED	CURRENT NEED
P_1	10 (5 Required)	5
P_2	4 (2 Required)	2
P_3	9 (7 Required)	2

Available 3

At time to total allocated resource is 9 and 3 units of R_1 is available.

At time to system is in safe state. Safe sequence is *{P_0, $P_{1.....}$, Pn }*

- i.e., Process P_2 can immediately allocated all its resource (R_1) and then return to the system.
- Available units become 5.
- P_1 gets all resources, finishes its execution and returns 10 units.
- Now P_3 gets 7 units of R_1. Finishes its execution and returns the resources.
- Now 12 units are available in the system.

Resource-Allocation-Graph Algorithm

- In addition to the request and assignment edges, a new type of edge, called a ***claim edge*** is introduced.
- A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j, atsome time in the future. It is represented by dashed line.
- When process P_i requests resource R_j, the claim edge $P_i \rightarrow R_j$ is converted to a request edge. Similarly, when aresource R_j is released by P_i, the assignment edge $R_j \rightarrow P_i$ is reconverted toa claim edge $P_i \rightarrow R_j$.
- Process P_i, requests resource R_j. The request can be grantedonly if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle in the resource-allocation graph. Cycle detection algorithm is used to check the safety.

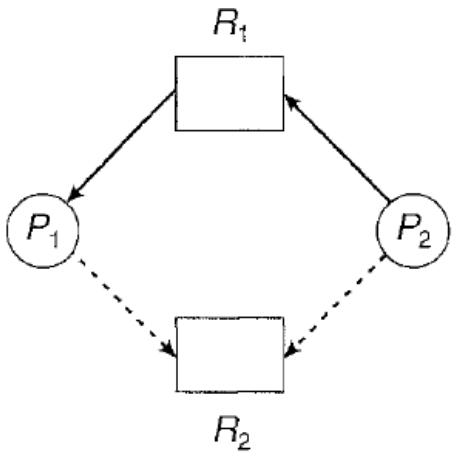


Figure 7.6 Resource-allocation graph for deadlock avoidance.

- In this figure Suppose that 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 indicates that the system is in an unsafe state.
- If P_1 requests R_2, and P_2 requests R_1, then a deadlock willoccur.

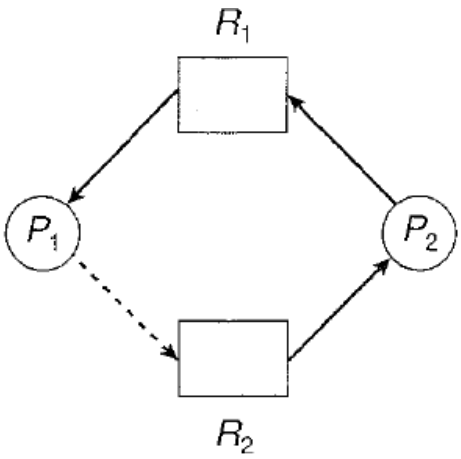


Figure 7.7 An unsafe state in a resource-allocation graph.

Banker's Algorithm

- Resource-allocation-graph algorithm is not applicable to a resourceallocationsystem with multiple instances of each resource type.
- This algorithm could be used in a banking system to ensure that the bank neverallocated its available cash in such a way that it could no longer satisfy theneeds of all its customers.
- When a user requestsa set of resources, the system must determine whether the allocation of theseresources will leave the system in a safe state. If it will, the resources areallocated; otherwise, the process must wait until some other process releasesenough resources.
- Data structures used in this algorithm are-
    - **Available :**A vector of length *m* indicates the number of available resourcesof each type. If *Available [j]* equals *k*, there are *k* instances of resource type$R_i$available.
    - **Max :**An *n* x *m* matrix defines the maximum demand of each process.If Max { i, j}= k , then process P_i may request at most *k* instances of resource type R_j.
    - **Allocation :**An *n* x *in* matrix defines the number of resources of each typecurrently 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 eachprocess. If *Need{ i ,j }* equals *k,* then process P_i may need *k* more instances ofresource type $R_j$to complete its task.
- **Need { *i }{ j }* = Max { *i }{ j }* – Allocation { *i }{ j }* .**

Safety Algorithm

Safety algorithm is used to find the state of the system. i.e., System may be in safe state or unsafe state.

1. Let work and finish be vector of length m and n respectively

   *I*nitialize

   Work = Available

   Finish { *i*} = Fale for *i* = 0,1, ……. n – 1
2. Find an *i* such that both

   a. Finish { *i*}= False

   b. Need $_i \leq$ Work

   If no such exists go to step 4
3. Work = Work + Allocation *i*

   Finish { *i*} = t

   Go to step 2
4. If finish { *i*} = = *true* for all *i* , then the system is in a safe state.

Resource-Request Algorithm

This algorithm determines if requests can be safely granted .let request $_i$ be the request vector for process P_i. If $Request_i$ *[j] == k,* thenprocess P_i,wants *k* instances of resource type *Rj*. When a request for resourcesis made by process P_i, the following actions are taken:

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, sincethe process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i, must wait, since theresources are not available.
3. $Available = Available - Request_i$

   $Allocation_i = Allocation_i + Request_i$

   $Need_i = Need_i - Request_i$

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

Example

System consists of five process (P_1, P_2, P_3, P_4, P_5) and three resources (R_1, R_2, R_3)

Resource type R_1 has 10 instance

Resource type R_2has 5 instance

Resource type R_3has 7 instance

Following snapshot of the system has been taken

	Allocation			**Max**			**Available**	
Process	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2 R_3
P_1	0	1	0	7	5	3	3	3 2
P_2	2	0	0	3	2	2		
P_3	3	0	2	9	0	2		
P_4	2	1	1	2	2	2		
P_5	0	0	2	4	3	3		

Content of need matrix is → **Need = Max – Allocation**

	Need		
Process	R_1	R_2	R_3
P_1	7	4	3
P_2	1	2	2
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Currently the system is in safe state.

Safe sequence

Safe sequence is calculated as follow

1. Need of each process is compared with available.

   If Need $_i \leq$ Available $_i$, then the resources are allocated to that process and process will release the resource.

2. If need is greater than available, next process need us taken for comparison.
3. In the process example

Need of P_1 is (7, 4, 3) and Available is (3,3,2)

Need ≥ Available → False

So system will move for next process

4. Need of P_2 is (1, 2, 2) and Available is (3, 3, 2)

   So Need ≤ Available (work)

   (1, 2, 2) ≤ (3, 3, 2) = True

   Then Finish {*i* } = True

   Request of P_2 is granted and P_2 release the resource to the system.

   Work = Work + Allocation.

   Work = (3, 3, 2) +(2,0,0)

   = (5,3,2)

   This procedure is continued for all process

5. Process P_3 Need (6, 0, 0) Available (5, 3, 2) move to next process
6. Process P_4 Need (1, 1, 1) Available (5, 3, 2) need ≤ Available

   (0,1,1) ≤(5,3,2)=True

   Available = Available + Allocation

   =(5,3,2) + (2,1,1)

   =(7 4 3)

7. P_5 Need (4 3 1) available (7 4 3) Need ≤ Available

   P_5 is granted

   Available = Available + Allocation

   =(7 4 3) + (0 0 2)

   =(7 4 5)

8. Process P_1 Need (7 4 3) and Available (7 4 5). If this request is granted then the system may be in the deadlock state. After granting the request, available resource is (0 0 2) so the system is in unsafe state.
9. Process P_3 Need (6 0 0) Available (7 4 5) Need ≤ Available

   P_3 is granted

   Available = Available + Allocation

   =(7 4 3) + (3 0 2)

   =(10 4 5)

10. P_1 Need (7 4 3) and Available (10 4 7) Need ≤ Available

    (7 4 3) ≤ (10 4 7) = True

P_1 is granted

Available = Available + Allocation

=(10 4 7) + (010)

=(1057)

Safe sequence is { P_2 P_4 P_5 P_3 P_1 }

Disadvantages of Bomker's Algorithm

1. It requires that there be a fixed number of resources to allocate.
2. Algorithm requires that users state their maximum need in advance.
3. Number of users must remain fixed.

Deadlock Detection

- If a system does not employ either a deadlock-prevention or a deadlock-avoidancealgorithm, then a deadlock situation may occur.
- System may provide:
  - An algorithm that examines the state of the system to determine whethera deadlock has occurred
  - An algorithm to recover from the deadlock

Single Instance of Each Resource Type

- If all resources have only a single instance, then wait-for graph is used to detect deadlock.
- Wait-for graph is obtained from resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.
- An edge $P_i \rightarrow P_j$existsin a wait-for graph if and only if the corresponding resource allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$for some resource R_q.
- 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 ***invoke an algorithm*** that searches for a cycle in the graph.

Several Instances of a Resource Type

- The wait-for graph scheme is not applicable to a resource-allocation systemwith multiple instances of each resource type.
- A different deadlock detection algorithm is used for such system
- Data structures used are –

- Available : A vector of length m indicates the number of available resource of each type
- Allocation : An n x m matrix defines the number of resources of each type currently allocated to each process
- Request : An n x m matrix indicates the current request of each process. If request { i } { j } = k, then process P_i is requesting k More instances of resource type R_j.

➢ This algorithm checks every possible allocation sequence for the process that remain to be completed.

1. Let work and Finish be vectors of length m and n respectively
   Initialize Work = Available
   For i =0,1,……., n-1 , if Allocation $_i \neq 0$, then Finish { i }= false;
   other wise, Finsih { i } =true.
2. Find an index i such that both
   a. Finish { i } == False
   b. Request $_i \leq$ work
   c. 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, $0 \leq i < n$, then the system is in adeadlocked state
   If Finish { i } == false, then process P_i is deadlocked.

Recovery From Deadlock

Two Options for breaking a deadlock

➢ Abort one or more process to break the circular wait.

➢ Preempt some resourcesfrom one or more of the deadlocked processes.

Process Termination

➢ ***Abort all deadlocked processes***. This method clearly will break thedeadlock cycle, but at great expense;

➢ ***Abort one process at a time until the deadlock cycle is eliminated.*** In this each process is aborted, a deadlock-detection algorithm must be invoked to determine whether anyprocesses are still deadlocked.

Resource Preemption

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

Three issues need to be addressed are

- **Selecting a victim.** Which resources and which processes are to be preempted? As in process termination, we must determine the order ofpreemption to minimize cost.
- **Rollback.** If we preempt a resource from a process, what should be donewith that process? Clearly, it cannot continue with its normal execution; itis missing some needed resource.
- **Starvation.** How do we ensure that starvation will not occur? That is,how can we guarantee that resources will not always be preempted fromthe same process?

Assignment Questions

1. Why is deadlock state more critical than starvation? Describe resource allocationgraph with a deadlock, with a cycle but no deadlock. (8**) Dec 07/Jan 08**
2. What are two options for breaking deadlock? (7) **Dec 07/Jan 08**
3. What is wait-for graph? How is it useful for detection of deadlock? (5) **Dec 07/Jan 08**
4. Define race condition. List the requirements that a solution to critical section problemmust satisfy. (4) **Dec 08/Jan 09**
5. Define algorithms Test() and set() and swap(). Show that they must satisfy mutualexclusion. (6) **Dec 08/Jan 09**
6.

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

Show the system is not deadlocked by one safe sequence. At t2,p2 makes oneadditional request for type C, show that system is deadlocked if request is granted.(10) **Dec 08/Jan 09.**

7. Define hardware instructions test() , set() and swap(). Give algorithms to implementmutual exclusion with these instructions. (6) **Dec 09/ Jan 10**
8. Describe necessary conditions for a deadlock situation to arise. (4) **Dec 09/ Jan 10**
9. Consider given chart and answer i) what is content of matrix need? i) is systemsafe? iii) if request comes from p, arrives for (0,4,2,0), can it be granted ? (12) **dec 2010, june 2011.**

VTU question paper questions

1. Why is deadlock state more critical than starvation? Describe resource allocationgraph with a deadlock, with a cycle but no deadlock. (8) Dec 07/Jan 08
2. What are two options for breaking deadlock? (7) Dec 07/Jan 08
3. What is wait-for graph? How is it useful for detection of deadlock? (5) Dec 07/Jan 08

4. Define race condition. List the requirements that a solution to critical section problemmust satisfy. (4) Dec 08/Jan 09
5. Define algorithms Test() and set() and swap(). Show that they must satisfy mutual exclusion. (6) Dec 08/Jan 09
6.

	allocation	***request***	***available***
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

Show the system is not deadlocked by one safe sequence. At t2,p2 makes oneadditional request for type C, show that system is deadlocked if request is granted.(10) Dec 08/Jan 09.

7. Define hardware instructions test() , set() and swap(). Give algorithms to implementmutual exclusion with these instructions. (6) Dec 09/ Jan 10
8. Describe necessary conditions for a deadlock situation to arise. (4) Dec 09/ Jan 10
9. Consider given chart and answer
   i. what is content of matrix need?
   ii. Is systemsafe?
   iii. If request comes from p, arrives for (0,4,2,0), can it be granted ? (12) dec 2010, jun 2011.

10 What do you mean by fragmentation? Explain difference between internal and external fragmentation. (6) **Dec 07/Jan 08**

11. For page reference string : 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6, how many pagefaults would occur for LRU and optimal alg. Assuming 2 and 6 frames. (10) **Dec 07/Jan 08**

12. What is the cause of thrashing? How does system detect thrashing? (4) **Dec 07/Jan 08, Dec 08/Jan 09, Dec 09/ Jan 10, june 2011.**

13. Differentiate between internal and external fragmentation. How are they overcome?(4)

14. What is paging and swapping? (4) **Dec 08/Jan 09**.

15. With diagram, discuss steps involved in handling a page fault. (6)) **Dec 09/ Jan 10**

16. What is address binding? Explain with necessary steps, binding instructions and datato memory addresses. (8) **Dec 09/ Jan 10**

Recommended question

1. What is paging ? Give advantages and disadvantages.
2. What is segmentation? Give advantages & disadvantages.
3. Differentiate between paging & segmentation.
4. What are the different methods of implementing page table?
5. How can you ensure protection & sharing in paging?
6. What is fragmentation? Explain different types of fragmentation.
7. Explain swapping .
8. How to satisfy a request of size n from a list of free holes? Explain.
9. Explain the following:
    a) privileged instruction
    b) transient code
    c) 50-percent role
    d) Roll in, roll out
    e) Compaction
10. How can you ensure hardware address protection with base & limit registers?
11. Explain address binding using base & limit registers.
12. What is dynamic loading ? Give advantages.
13. Explain the following:
    a) Frame table
    b) Hit ratio
    c) Re-entrant code
    d) Legal page
    e) TLB
14. Give a brief idea on dynamic linking & shared libraries.

OTHER IMPORTANT QUESTIONS:

1. For the following snapshot of the system find the safe sequence (using Banker'salgorithm).

Process	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	2			

a. Calculate the need of each process?

b. To find safe sequence?

2. Consider the following snapshot of the system and answer the following questions using Banker's

algorithm?

a. Find the need of the allocation?

b. Is the system is in safe state?

c. If the process P1 request (0,4,2,0) resources cam the request be granted immediately?

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	5	2	0
P2	1	0	0	0	1	7	5	0				
P3	1	3	5	4	2	3	5	6				
P4	0	6	3	2	0	6	5	2				
P5	0	0	1	4	0	6	5	6				

3. The operating system contains three resources. The numbers of instances of each resource type are (7, 7,10). The current allocation state is given below.

a. Is the current allocation is safe?

b. find need?

c. Can the request made by the process P1(1,1,0) can be granted?

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	2	2	3	3	6	8
P2	2	0	3	4	3	3
P3	1	2	4	3	4	4

4. Explain different methods to recover from deadlock?

5. Write advantage and disadvantage of deadlock avoidance and deadlock prevention?

Outcome

- Familiarized with deadlock detection, prevention, avoidance and recovery from deadlock.
- Familiarized with swapping
- Familiarized with segmentation, contiguous memory allocation
- Know the segmentation,

Further Reading

21. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne: Operating System Principles, 8th edition, Wiley India, 2009. (Listed topics only from Chapters 1 to 12, 17, 21)
22. D.M Dhamdhere: Operating systems - A concept based Approach, 2nd Edition, Tata McGraw- Hill, 2002.
23. P.C.P. Bhatt: Introduction to Operating Systems: Concepts and Practice, 2nd Edition, PHI, 2008.
24. Harvey M Deital: Operating systems, 3rd Edition, Pearson Education, 1990.
25. http://nptel.ac.in/courses/106106144
26. https://en.wikipedia.org/wiki/Operating_system
27. https://www.tutorialspoint.com/operating_system
28. http://www.studytonight.com/operating-system
29. https://www.linux.com
    https://opensource.com

MODULE 4:
MEMORY MANAGEMENT

Memory Management:

Memory management strategies:

Background

Swapping

Contiguous memory allocation

Paging

Structure of page table

Segmentation.

Virtual Memory Management

Background

Demand paging

Copy-on-write

Page replacement

Allocation of frame

Assignment Questions

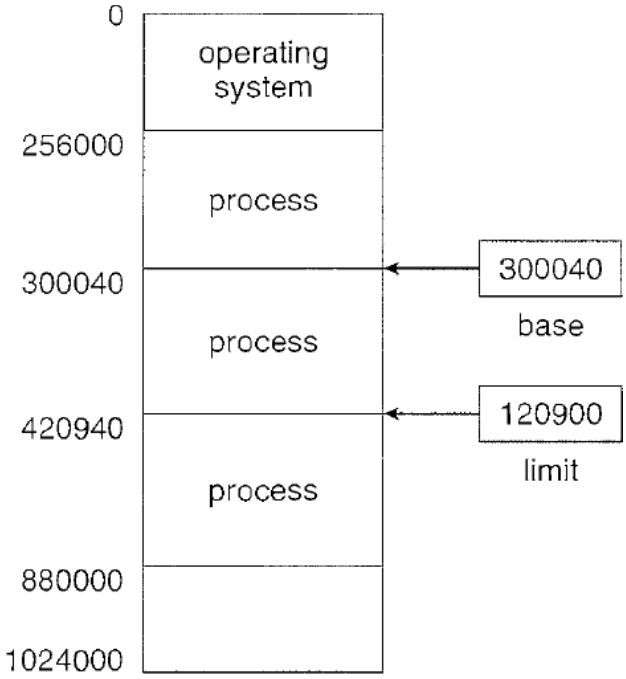
Outcome

Further Reading

Background

Memory consists of a large array of words or bytes, eachwith its own address.A instruction execution cyclefetches an instructionfrom memory. The instruction is then decoded and may cause operandsto be fetched from memory.After the instruction has been executed on theoperands, results may be stored back in memory.

Basic Hardware



• Main memory and the registers are the onlystorage that the CPU can access directly. • Any instructions in execution, and any data used by the instructions,must be in one of these direct-access storage devices. • Cache memory is provided to support faster data access. • Protection must be provided from unauthorized access. To provide this we need to make sure that each process has a separate memory space. • We can achieve this by using two register o Base register holds the smallest legal physical memory address o Limit register specifies the size of the range

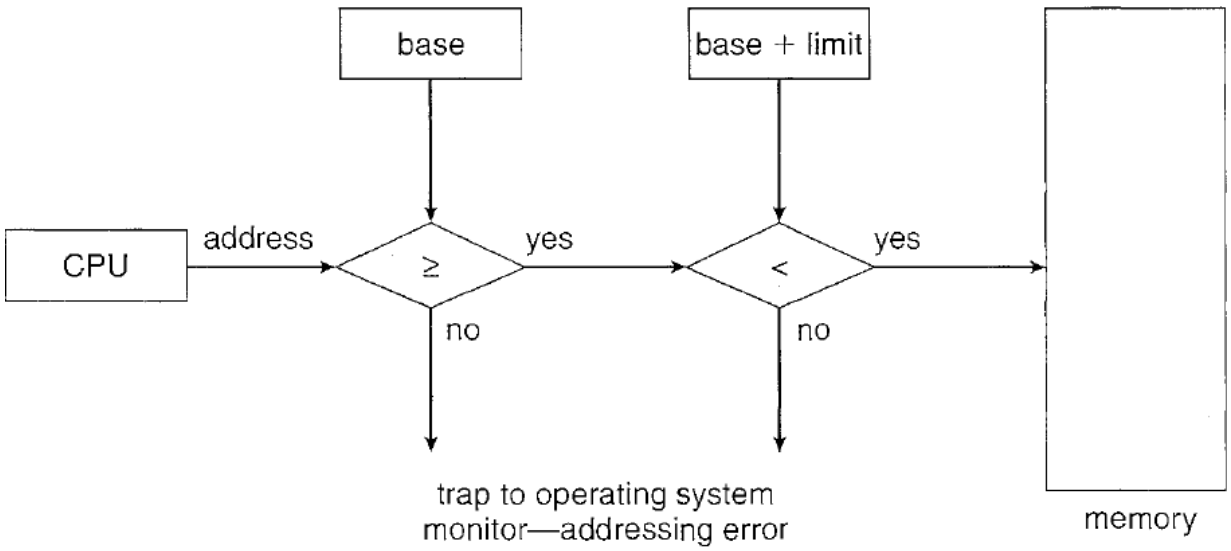


Figure 8.2 Hardware address protection with base and limit registers.

Ex: If the base register holds 30004 and limit register is 12090 then the program can legally access all addresses from 30004 through 42094(inclusive)

- Protection of memory space is accomplished by having the CPU hardwarecompare *even/* address generated in user mode with the registers. Any attemptby a program executing in user mode to access operating-system memory orother users' memory results in a ***trap*** to the operating system, which treats theattempt as a ***fatal error***

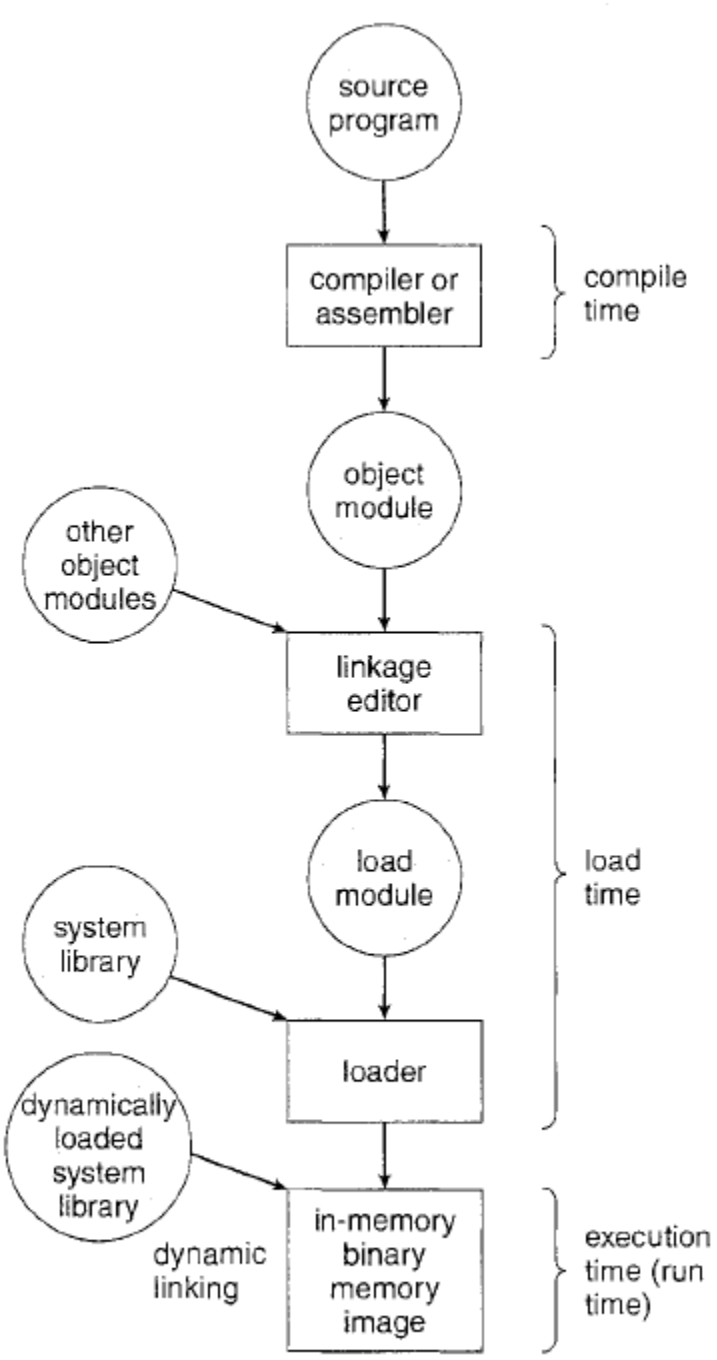


Figure 8.3 Multistep processing of a user program.

Address Binding

- Usually, a program resides on a disk as a binary executable file. To be executed,the program must be brought into memory and placed within a process
- Most systems allow a user process to reside in any part of the physicalmemory. (Thus, although the address space of the computer starts at 00000,the first address of the user process need not be 00000.)

- Addresses in the source program are generally symbolic (such as*count).*
- *A* compiler will typically **bind** these symbolic addresses to relocatableaddresses (such as "14 bytes from the beginning of this module").
- The linkageeditor or loader will in turn bind the relocatable addresses to absolute addresses(such as 74014).
- Each binding is a mapping from one address space to another. Binding canbe doneat following stages:
  - **Compile time**At the time of complication if we know the location of process, then **absolute code** can be generated.
  - **Load time** If it is not known at compile time where the process will residein memory, then the compiler must generate **relocatable code.**
  - **Execution time.** If the process can be moved during its execution fromone memory segment to another, then binding must be delayed until runtime.

Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a ***Logical address*.**
- An address seen by the memory unit—that is, the one loaded intothe **memory- address register** of the memory—is commonly referred to as a***Physical address.***
- The set of all logical addressesgenerated by a program is a **logical** address **space;** the set of all physicaladdresses corresponding to these logical addresses is a ***Physical address space.***
- The execution-time address-bindingscheme the logical and physical addresses space differ. In this case,we usually refer to the logical address as a **virtual address.**
- The run-time mapping from virtual to physical addresses is done by ahardware device called the **memory-management unit (MMU).**
- **Relocation register**is also called base register.
- Value of the relocation register is *added* to every address generated by auser process

at the time it is sent to memory.

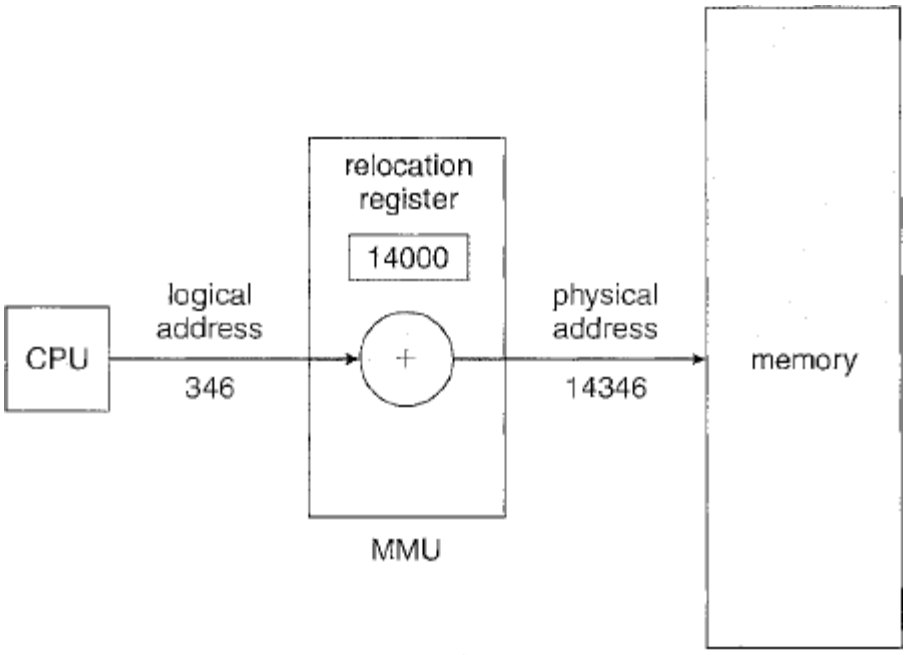


Figure 8.4 Dynamic relocation using a relocation register.

For example, base is at 14000, thenan attempt by the user to address location 0 isdynamically relocated to location 14000; an access to location 346 is mappedto location 14346.

- The user program never sees the *real* physical addresses. It deals with *logical* addresses.

Dynamic Loading

- Dynamic loading is used for better memory space utilization. User program size islarge as compared to memory size, Program or process are dynamically loaded into memory as per required. With dynamic loading, a routine is not loaded untilit is called. All routines are kept on disk in a relocatable load format. The mainprogram is loaded into memory and is executed.
- The advantage of dynamic loading is that an unused routine is neverloaded.
- Dynamic loading does not require special support from the operatingsystem.

Dynamic Linking and Shared Libraries

- In this linking of system libraries are postponeduntil execution time.
- Without this facility, each program on a systemmust include a copy of its language library in the executable image. It wastes both diskspace and main memory.

- With dynamic linking, a *stub* is included in the image for each libraryroutinereference. When it is executed it replaces itself with the address of the routine and executes the routine.
- This feature is also for library updates (such as bug fixes). Alibrary may be replaced by a new version, and all programs that reference thelibrary will automatically use the new version.
- Only programs that are compiled withthe new library version are affected by the incompatible changes incorporatedin it. Other programs linked before the new library was installed will continueusing the older library. This system is also known as **shared libraries.**

Swapping

- A process must be loaded into memory in order to execute. If there is not enough memory to keep all process then A processcan beswapped temporarily out of memory to a **backing store** and then broughtback into memory for continued execution.
- If a higher-priority process arrives and wants service, the memorymanager can swap out the lower-priority process. When the higher-priority process finishes, the
- lower-priority process can be swapped back in and continued. This process is called **roll out, roll in.**
- Normally, a process that is swapped out will be swapped back into thesame memory space it occupied previously.

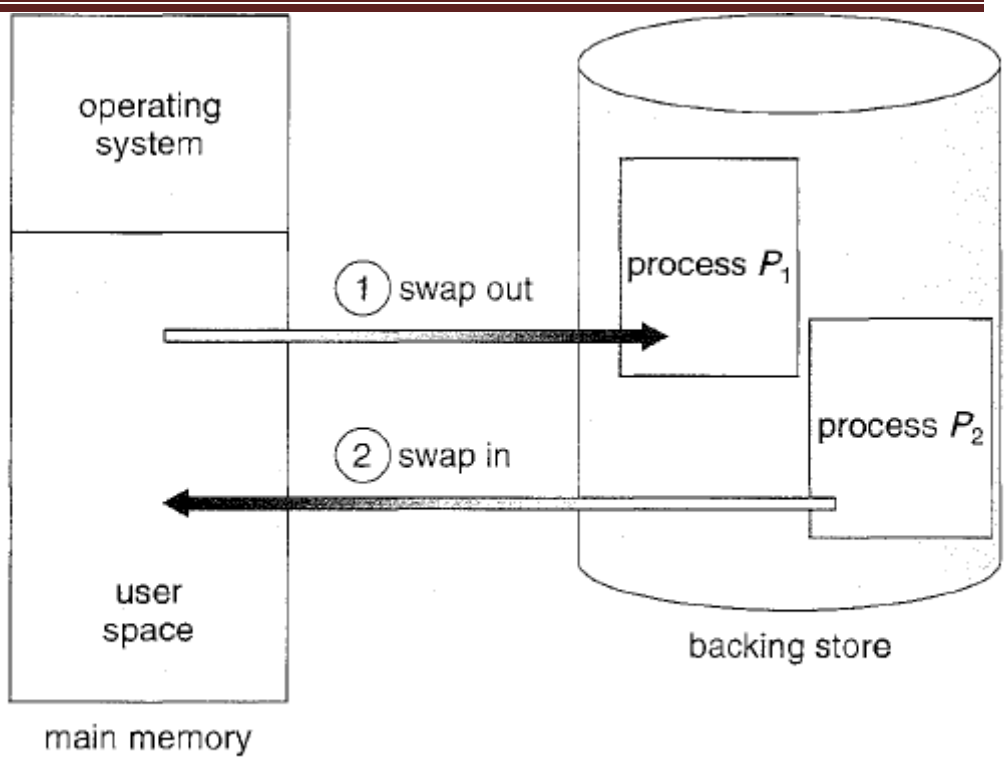


Figure 8.5 Swapping of two processes using a disk as a backing store.

- Backing store is commonly a fastdisk. It must be large enough to accommodate copies of all memory imagesfor all users, and it must provide direct access to these memory images.
- Major part of the swap time is transfer time. The total transfer time is directly proportional to the amount of memory swapped.

Contiguous Memory Allocation

- The main memory must accommodate both the operating system and thevarious user processes.
- In the contiguous memory allocation, each process is contained in a singlecontiguous section of memory.

Memory Mapping and Protection

Relocation register and limit register are used to map the memory. Memory management unit maps the logical address dynamically by adding the value in the relocation register.

Operating system protection by checking every address generated by the CPU against these register values.

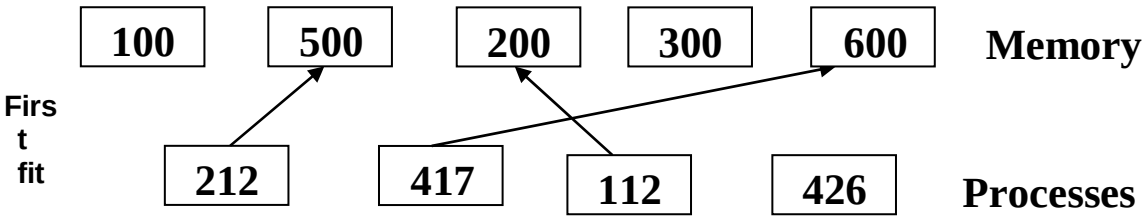
Memory Allocation

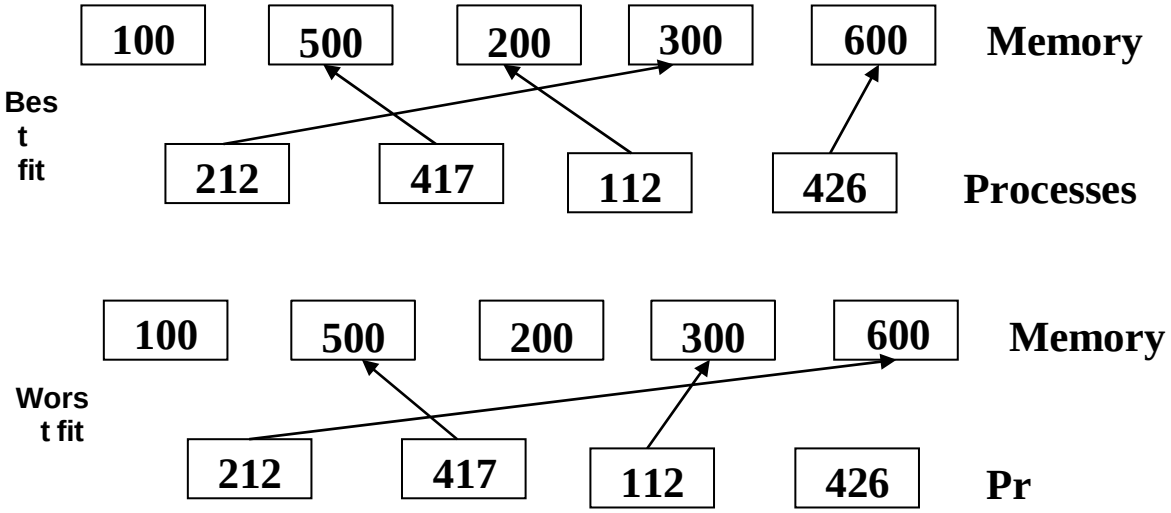
- Memory is divide memory into several fixed-sizedpartitions. Each partition may contain exactly one process.
- The degreeof multiprogramming is bound by the number of partitions.
- In this multiplepartitionmethod, when a partition is free, a process is selected fromthe input queue and loaded into the free partition. When the process terminates, thepartition becomes available for another process.
- In the fixed-partition scheme, the operating system keeps a table indicatingwhich parts of memory are available and which are occupied.
- Initially, allmemory is available for user processes and it is called as**hole.** When a process arrives and needs memory, we searchfor a hole large enough for this process. If any free hole is found, it is allocated to the process.
- The set of holes is searched to determine which hole is best to allocate. Memory management uses three algorithms for selecting free holes
    1) First fit
    2) Best fit
    3) Worst fit

First fit : First fit begins to scan memory from the beginning and chooses the first available block that is large enough.

Best fit : It chooses the block that is closest in size to the request. It allocates the smallest hole that is big enough.

Worst fit : It searches the entire list and allocate the largest hole.




100
500
200
300
600
Memory
Best fit
212
417
112
426
Processes

100
500
200
300
600
Memory
Worst fit
212
417
112
426
Pr

Best fit algorithm makes efficient use of memory.

Fragmentation

- As processes are loaded and removed from memory,the free memory space is broken into little pieces.
- External fragmentation existswhen there is enough total memory space to satisfy a request, but the availablespaces isnot contiguous; storage is fragmented into a large number of smallholes.
- solutionto external fragmentation is **"compaction"**. In this all free memory blocksare grouped together as a large block.
- In **Internal Fragmentation** thethe memory allocated to a process may be slightly larger than the requestedmemory. The difference between these two numbers is **internal fragmentation.**This memory that is internal to a partition but is not being used.
- In internal fragmentation the overhead to keep track of the hole will be larger thanthe hole itself.
- In compaction process move towards one side of memory and holes move in the other direction of memory. It produces one large hole of available memory.

Paging

- Paging permits the physical addressspace of a process to be noncontiguous.

Basic Method

- Physical memory is breakedinto fixed-sized blocks called "**frames**"
- Logical memory is breaked into blocks of same size called **"pages"**
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.

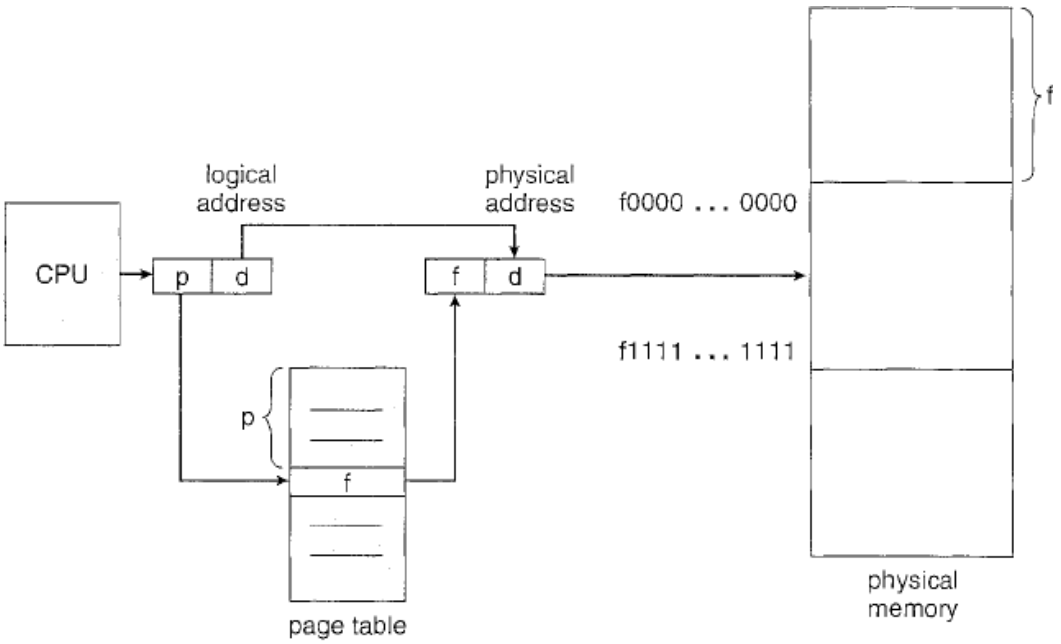


Figure 8.7 Paging hardware.

- Every addressgenerated by the CPU is divided into two parts:

  a**page number (p)** and

  a**page offset (d).**

- The page number is used as an index into a **page table.**
- Thepage table contains the base address of each page in physical memory.
- Base address is combined with the page offset to define the physical memoryaddress.

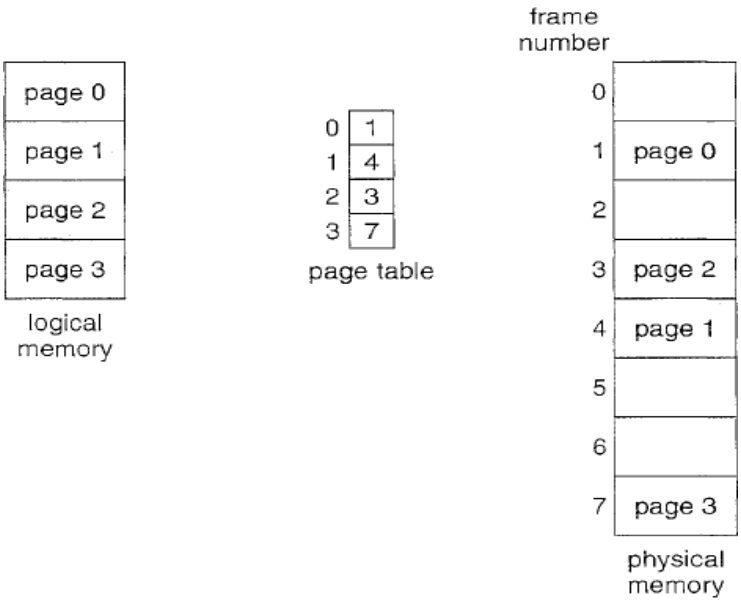


Figure 8.8 Paging model of logical and physical memory.

- When a process arrives in the system, each page of the process needs one frame. If process requires *n* pages, at least *n* frames must be available in memory. The first page of the process is loaded into one of the allocated frames and the frame number is put in the page table for this process. This process is continued for all pages.

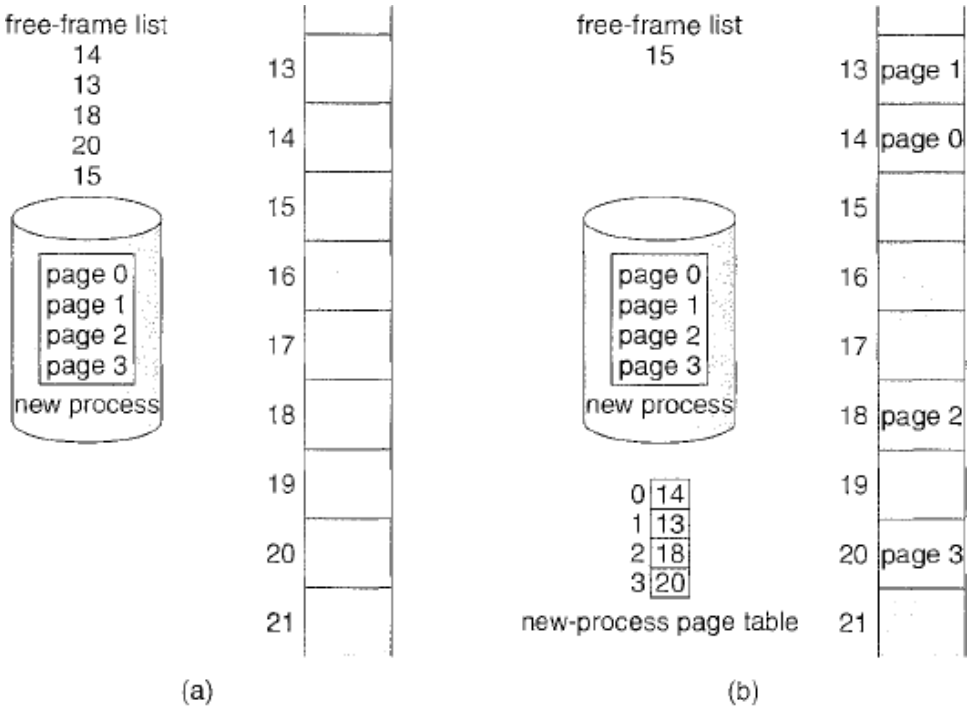


Figure 8.10 Free frames (a) before allocation and (b) after allocation.

- Paging makes clear separation between user's view of memory and actual physical memory.
- Address-translation hardware translates logical address into physical addresses.

Hardware Support

- If page table is small (ex: 256 entries) a set of register are used to implement it.
- In large page-tables a **page-table base register (PTBR)** points to the page table.In this memory access is very slow.
- Solution to this problem is use of **"translation look-aside buffer (TLB)"**
- Each entry in the TLB consists of two parts : a **key** and the **value**

"Working of TLB"

- TLB consists 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. If the page number is found, its frame number is available and is used to access memory. Thisprocess is called **"TLB hit".**
- If the page number is not in the TLB, a memory reference to the page table is made.

When the frame number is obtained, we can use it to access memory. It is referred as **"TLB miss"**.

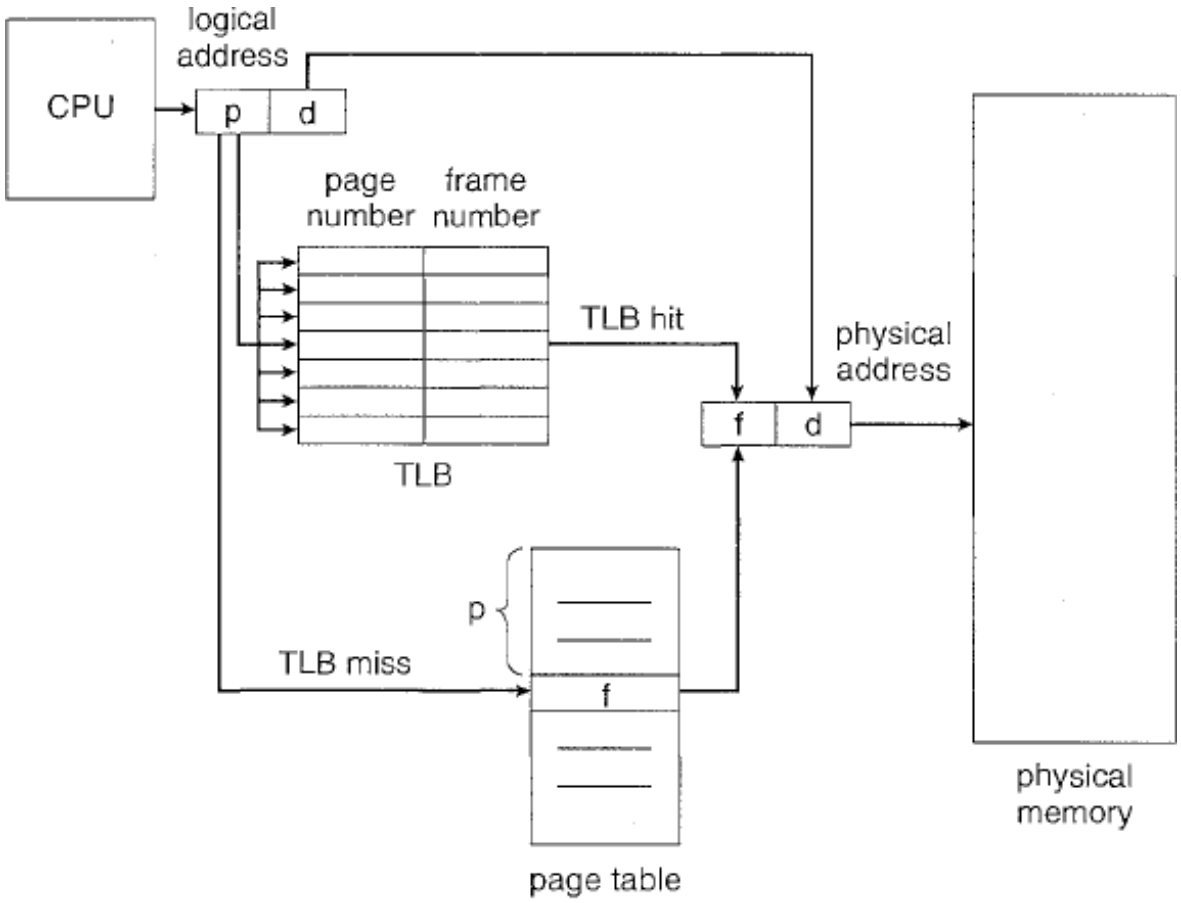


Figure 8.11 Paging hardware with TLB.

- **Hit – Ratio**The percentage of times that a particular page number is found in the TLB iscalled the hit ratio.
- An 80-percent hit ratio means that we find the desired pagenumber in the TLB 80 percent of the time.
- If it takes 20 nanoseconds to searchthe TLB and 100 nanoseconds to access memory, then a mapped-memory accesstakes 120 nanoseconds when the page number is in the TLB.
- If page not present in TLB it takes 20 nanoseconds to search TLB, 100 nanoseconds to access the desired byte for a total of"220" nanoseconds

To find effective access time = 0.80 x 120 + 0.20 x 220

= 140 nanoseconds.

We suffer a 40 – percent slowdown in memory – access time (from 100 to 140 nanoseconds)

Ex2: Percentage slow-down for 98-percent hit ratio

To find effective access time = 0.98 x 120 + 0.20 x 220
= 122 nanoseconds.

Protection

- Protection bits are used to provide read write protection in paged environment. llegal attempts will be trappedto the operating system.
- One additional bit is generally attached to each entry in the page table: a**valid-invalid** bit.When this bit is set to "valid," the page is considered as a valid page. This page is in the process's logical address space.

Shared Pages

- Paging allows sharing of common code.
- Consider asystem that supports 40 users, each of whom executes a text editor. If the texteditor consists of 150 KB of code and 50 KB of data space, we need 8,000 KB tosupport the 40 users.
- If the code is reentrant code (it never changes during execution) it can be shared as shown in the below fig.

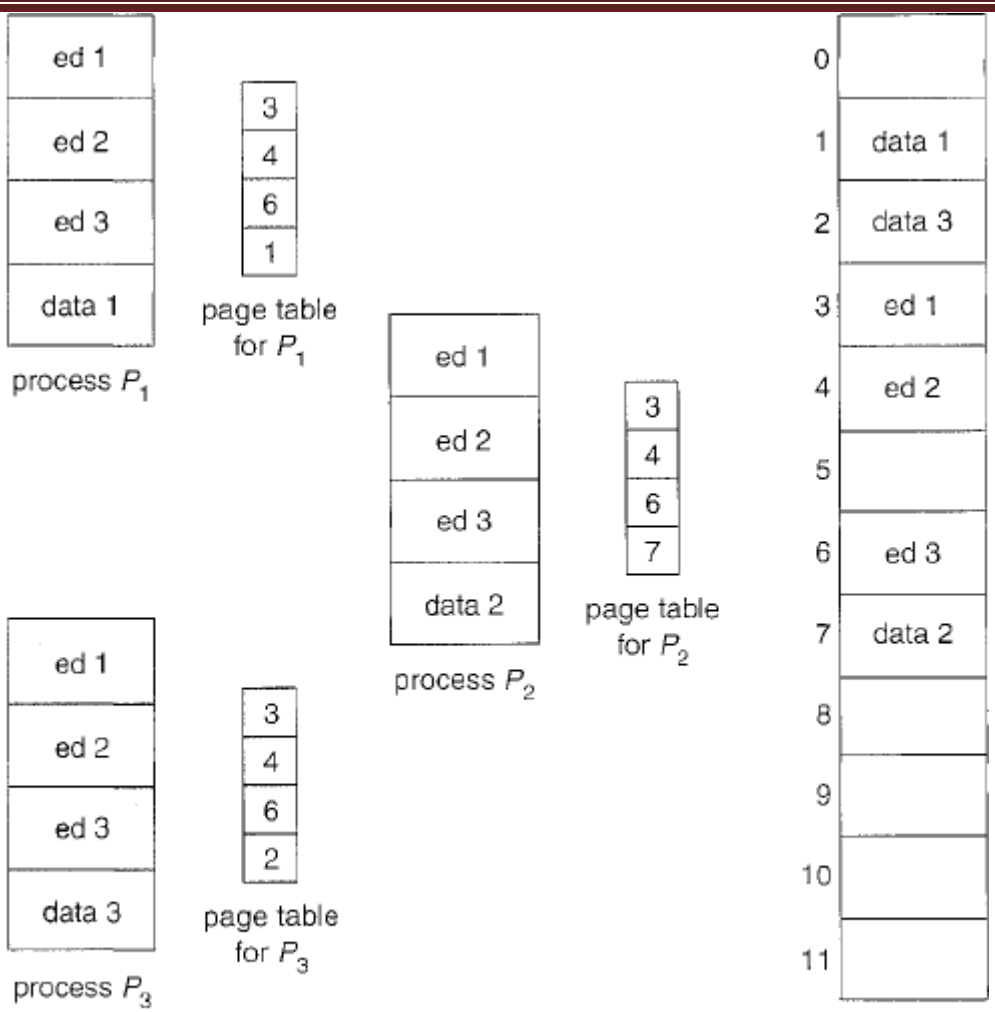


Figure 8.13 Sharing of code in a paging environment.

- In this fig..we see a three-page-editor each page 50kb in size being shared among three processes. Each process has its own data page.
- Thus to support 40 users, we need only one copy of editor(150kb) plus 40 copies of 50kb of data space per user. Total required space is 2150 kb instead 0f 8000kb.

Structure of the Page Table

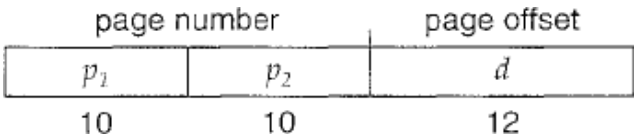
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Paging [Forward – mapped Page table]

- Most modern computer systems support a large logical address space. In such an environment, the page table itself becomes very large.

- In this case we would not want to allocate the page table contiguously inmain memory. One simple solution to this problem is to divide the page tableinto smaller pieces.
- One way is to use a two-level paging algorithm, in which the page tableitself is also paged.
- Ex: Consider a system with a 32-bit logical address space and page size is 4kb.A logical address is divided into a page numberconsisting of 20 bits and a page offset consisting of 12 bits.

  A page number is further divided into a 10-bit page numberand a 10-bit page offset. Thus, a logical address is as follows:



*P1*is an index into the outer page table and *p2* is the displacementwithin the page of the outer page table.

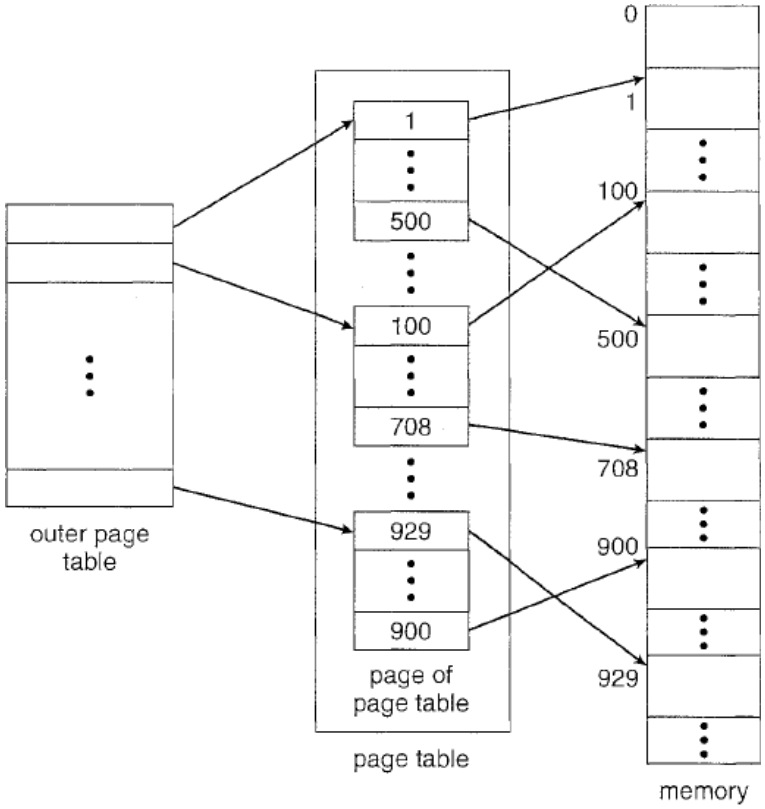


Figure 8.14 A two-level page-table scheme.

Hashed Page Tables

- Hash page table is used to handling address spaces larger than 32 bits.
- Virtual page number is the hash value.

Eachentry in the hash table contains a linked list of elements that hash to the samelocation.

- Each element consists of three fields:
  (1) Virtual page number,
  (2) Value of the mapped page frame, and
  (3) Pointerto the next element in the linked list.

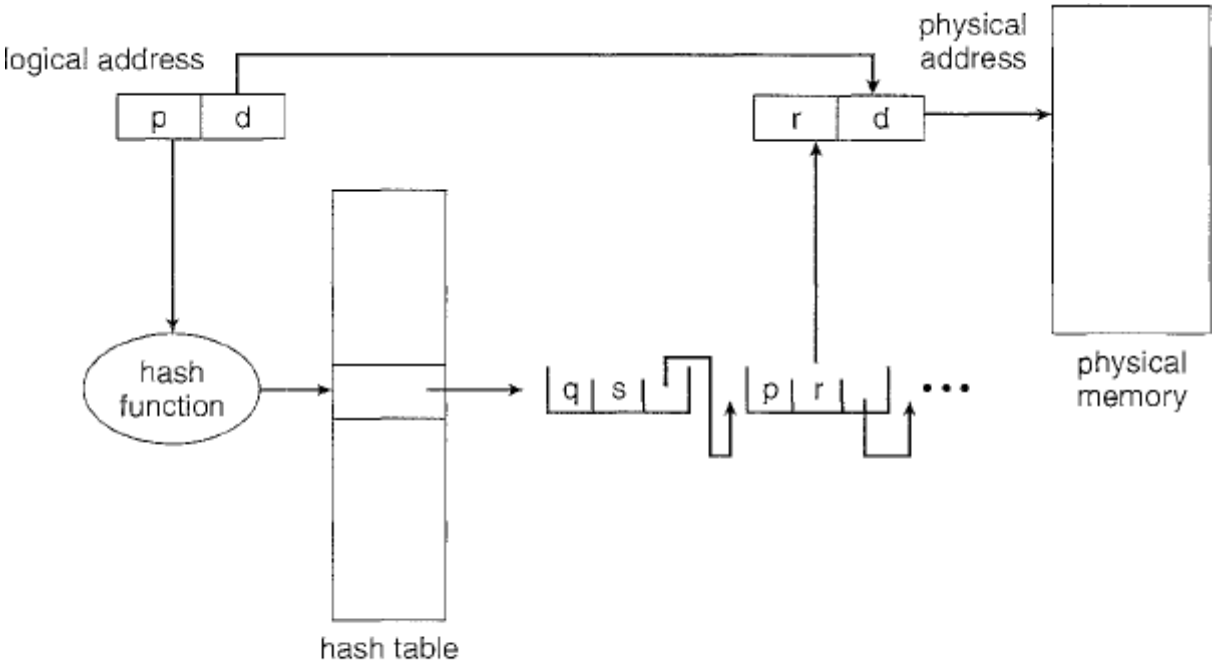


Figure 8.16 Hashed page table.

- The virtual page number in the virtualaddress is hashed into the hash table. The virtual page number is comparedwith field 1 in the first element in the linked list.
- If there is a match, thecorresponding page frame (field 2) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for amatching virtual page number.

Inverted Page Tables

- Inverted page table has one entry for each frame (real page) of memory.
- Each entryconsists of the virtual address of the page stored in that real memory location,with information about the process that owns that page.
- Each virtual address in the system consists of atriple

  <process-id, page-number, offset>.

Each inverted page-table entry is a pair <process-id, page-number>

- When a memoryreference occurs, part of the virtual address, consisting of <process-id, pagenumber>,is presented to the memory subsystem.
- The inverted page tableis then searched for a match. If a match is found at entry *i*then thephysical address <*i*, or'fset> is generated. If no match is found, then an illegaladdress access has been attempted.

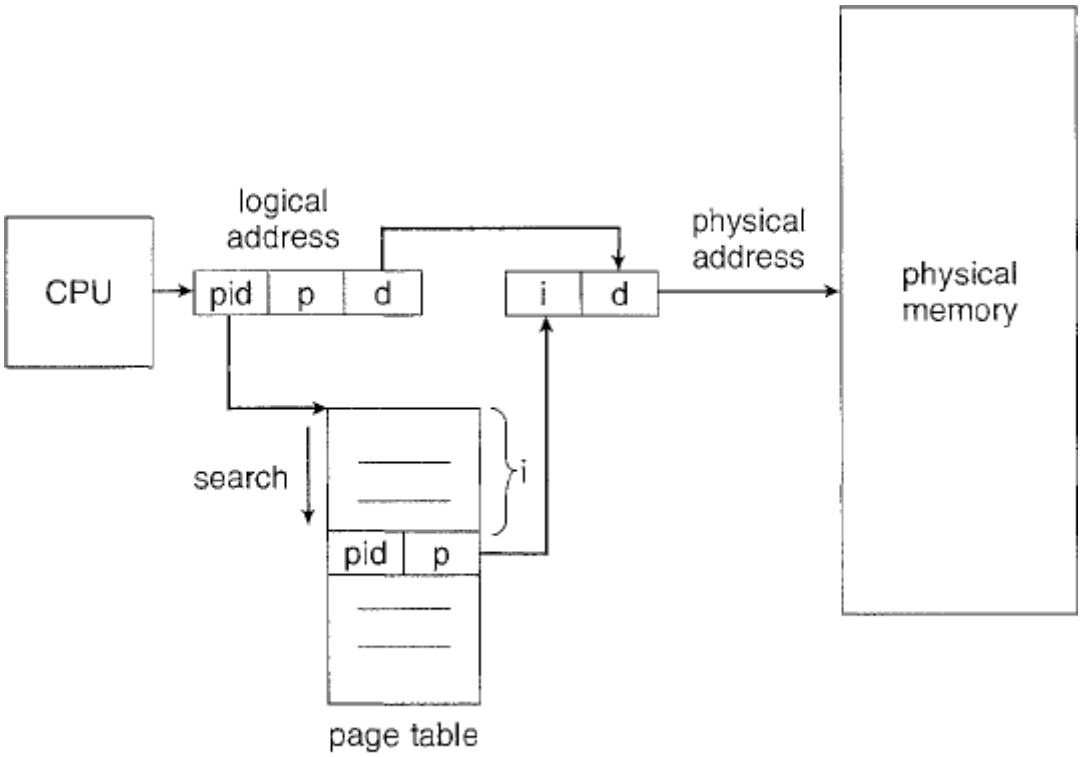


Figure 8.17 Inverted page table.

Segmentation

- Users prefer to view memory as a collection of variable-sized segments., withno necessary ordering among segments

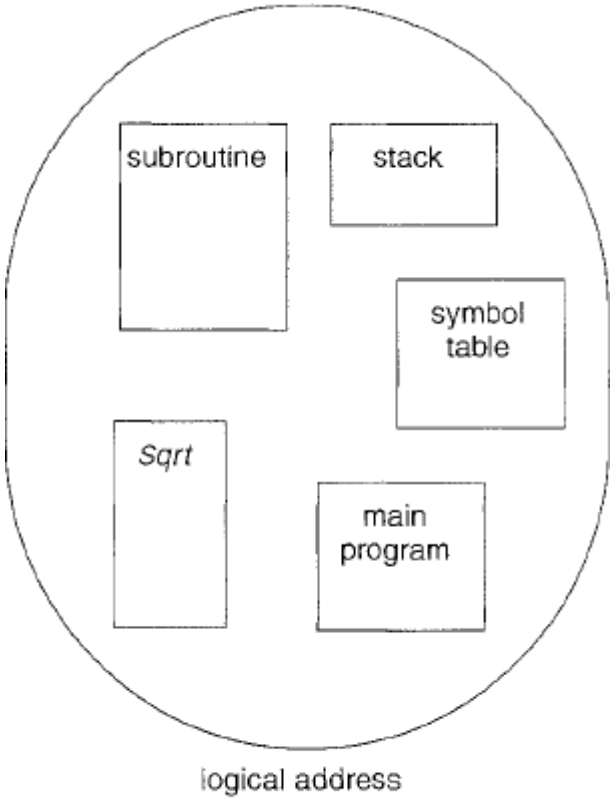


Figure 8.18 User's view of a program.

- Ex:- Program is considered as a collection of main program, set of methods, data structures etc.,
- Eachof these segments is of variable length.
- Elements within a segment areidentified by their offset from the beginning of the segment.
- Segmentation is a memory-management scheme that supports this userview of memory.
- A logical address space is a collection of segments. Eachsegment has a name and a length.
- User specifies each addressby two quantities: a segment name and an offset. In paging user specified only a single address.
- segments are numbered and are referredto by a segment number, rather than by a segment name. Thus, a logical addressconsists of a *two tuple:*

  <segment-number, offset >

Hardware

- Operating system will map two dimension user-defined address into one dimensional physical address. This mapping is effected by a segment table.
- Each entry in the segment table has a segment base and a segment limit

- Segment base contains starting physical address where the segment resides in memory segment limit specifies the length of the segment.

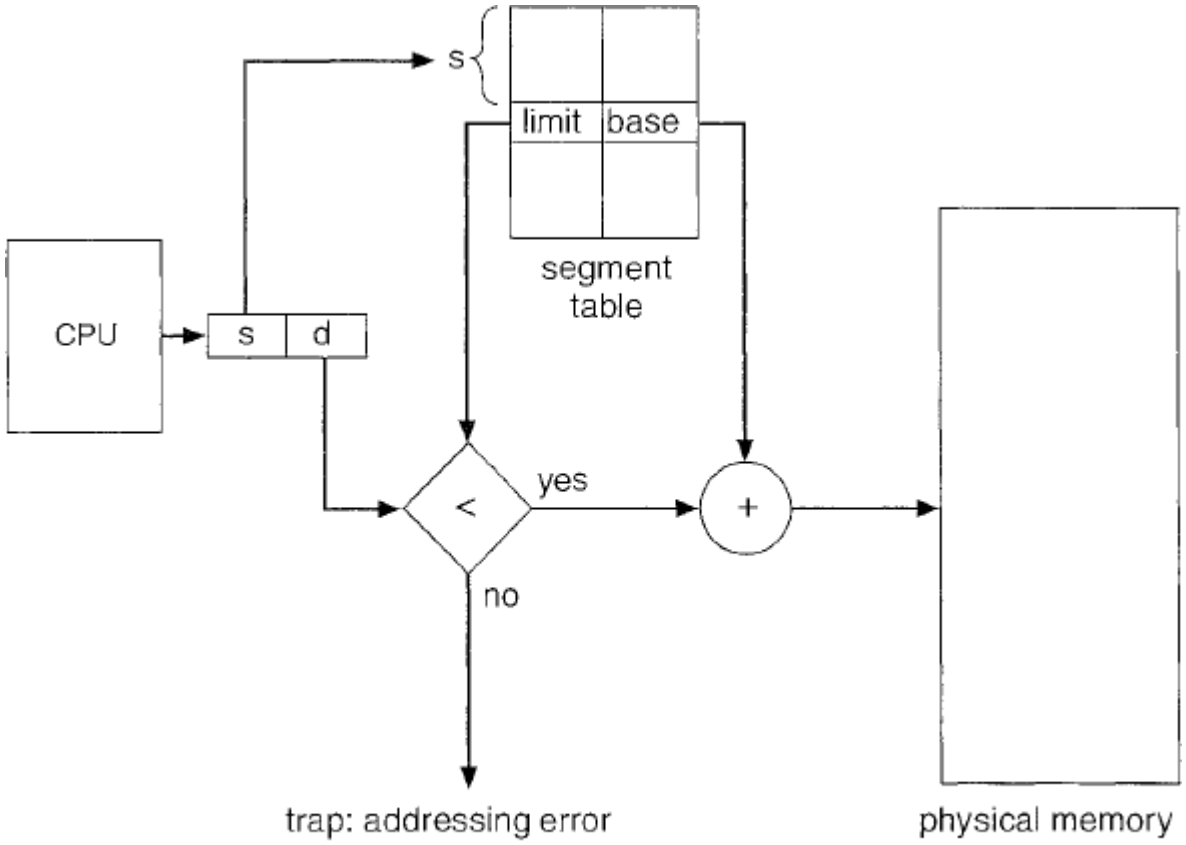


Figure 8.19 Segmentation hardware.

- Above fig illustrates the use of segment table. A logical address consist of two parts a sgment number S and an offset into that segment d.
- The segment number is used as an index to the segment table.
- The offset d of the logical address must be between 0 and the segment limit
- If it is not trap is generated.

  Ex: segment 2 is 400 bytes long and begins at location 4300.

  A reference to byte 53 of segment 2 is mapped onto location 4300+53=4353.

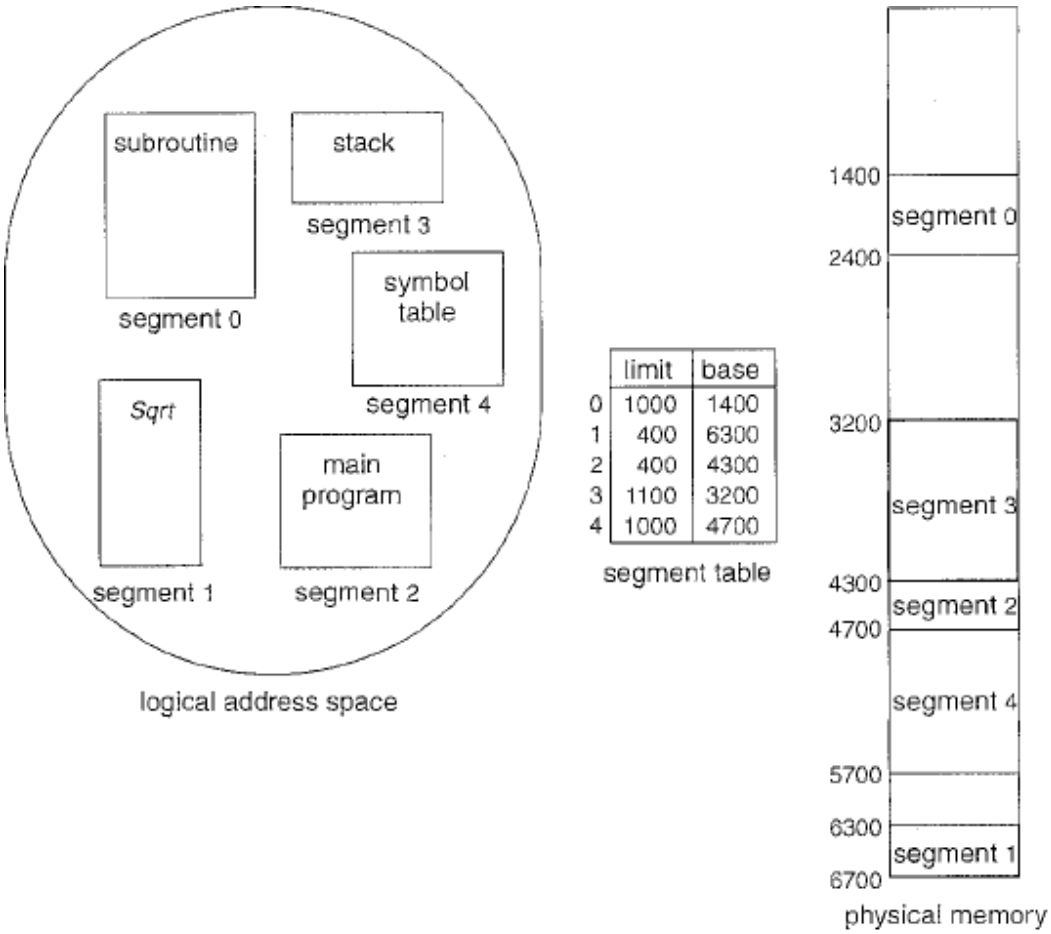


Figure 8.20 Example of segmentation.

Introduction

This unit gives the overview of file, access methods, file system mounting. File sharing, protection and implementing file systems are discussed in detail. Different allocation methods are highlighted. The concept of free space management and directory implementations are introduced. The concept of virtual memory and demand paging is discussed. The different page replacement algorithms with problems are highlighted. The concept of trashing is introduced

Objective

- virtual memory management
- Know different page replacement algorithm
- Understand different file access methods
- Understand implementing file system
- Understand allocation methods and free space management

Virtual memory-management

- The instructions being executed must be in physical memory. Memory management algorithms are required to satisfy this. It limits the size of a program to the size of physical memory
- In many cases entire program is not needed at the same time.
  Ex: Programs used to handle error conditions.
- Ability to execute a program that is only partially in memory has following advantages
  - Program is not constrained by physical memory
  - More programs can be run at the same time.

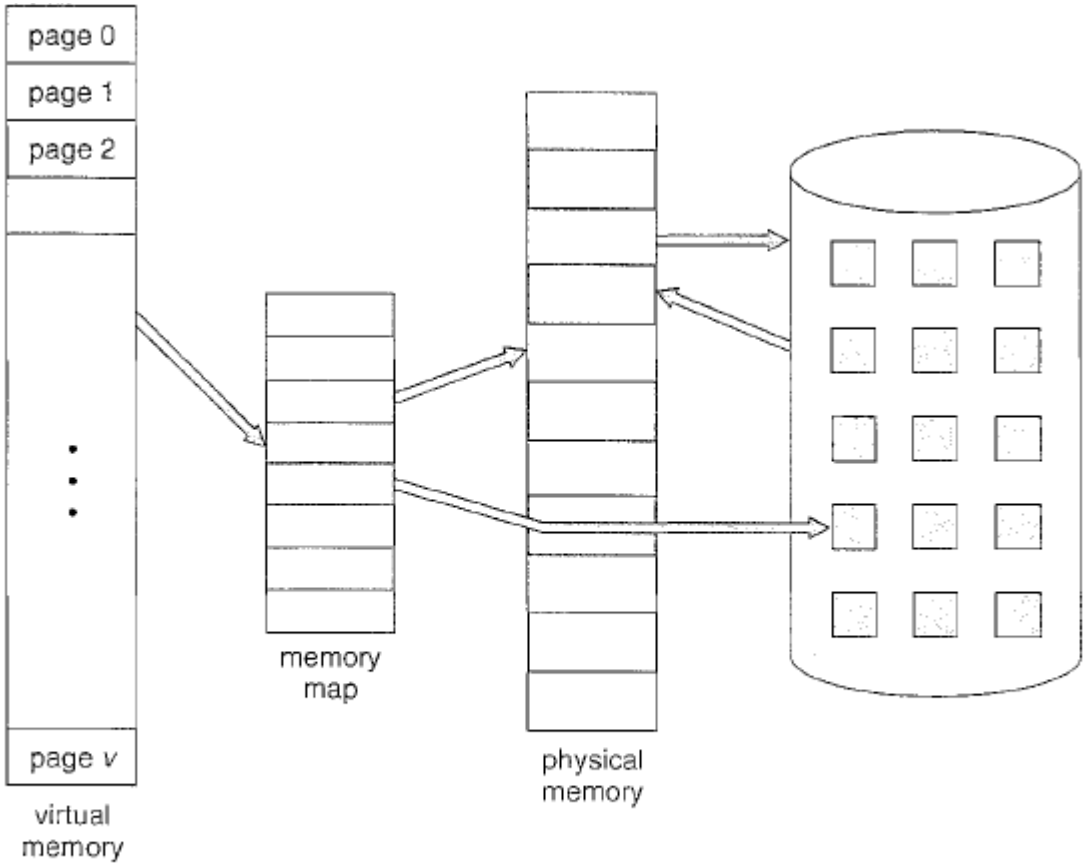


Figure 9.1 Diagram showing virtual memory that is larger than physical memory.

- Virtual memory separates logical memory from physical memory.
- Large virtual memory is provided to user when only a smaller physical memory is available.
- Virtual address space of a process refers to the logical view of how a process is stored in memory.

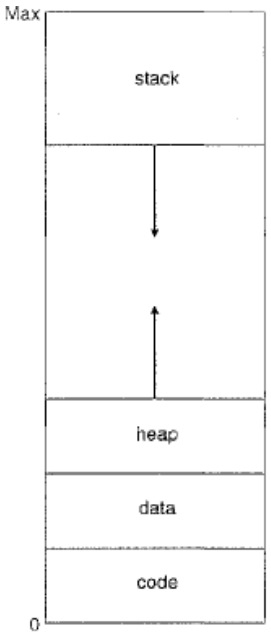


Figure 9.2 Virtual address space.

- We allow for heap to grow upwards in memory. Similarly, we allow for stack to grow downward in memory.
- Virtual memory also allows files and memory to be shared by two or more processes through page sharing.
- System libraries can be shared by several process through mapping of the shared object into a virtual address space.

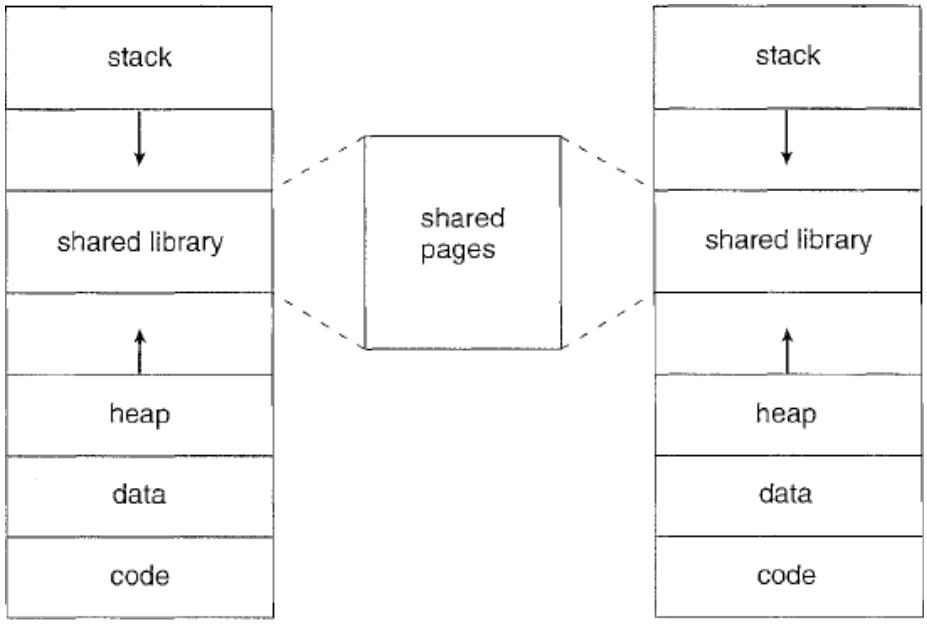


Figure 9.3 Shared library using virtual memory.

Demand Paging

- In demand paging virtual memory system pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.
- A demand-paging system is similar to a paging system with swapping. Rather than swapping the entire process, pages are swapped at the time of need.
- We thus use *pager,* rather than *swapper,* in connection with demand paging.

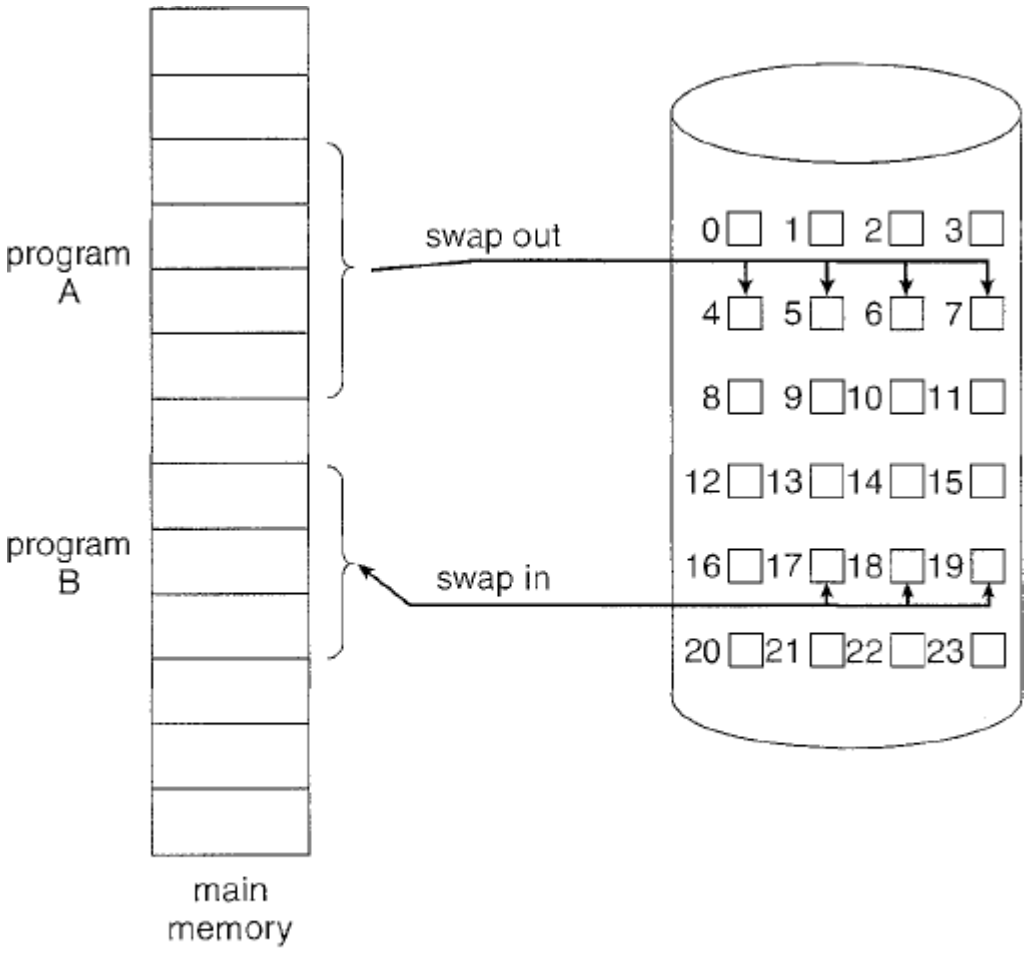


Figure 9.4 Transfer of a paged memory to contiguous disk space.

- Instead of swapping in a whole process, the pager brings only those pages into memory. decreasing the swap time and the amount of physical memory needed.
- Pages in memory and pages on the disk can be distinguish by "valid-invalid" bit scheme
- When the bit is set to "valid", the page is in memory. If the bit is set to be "invalid" page in on the disk.

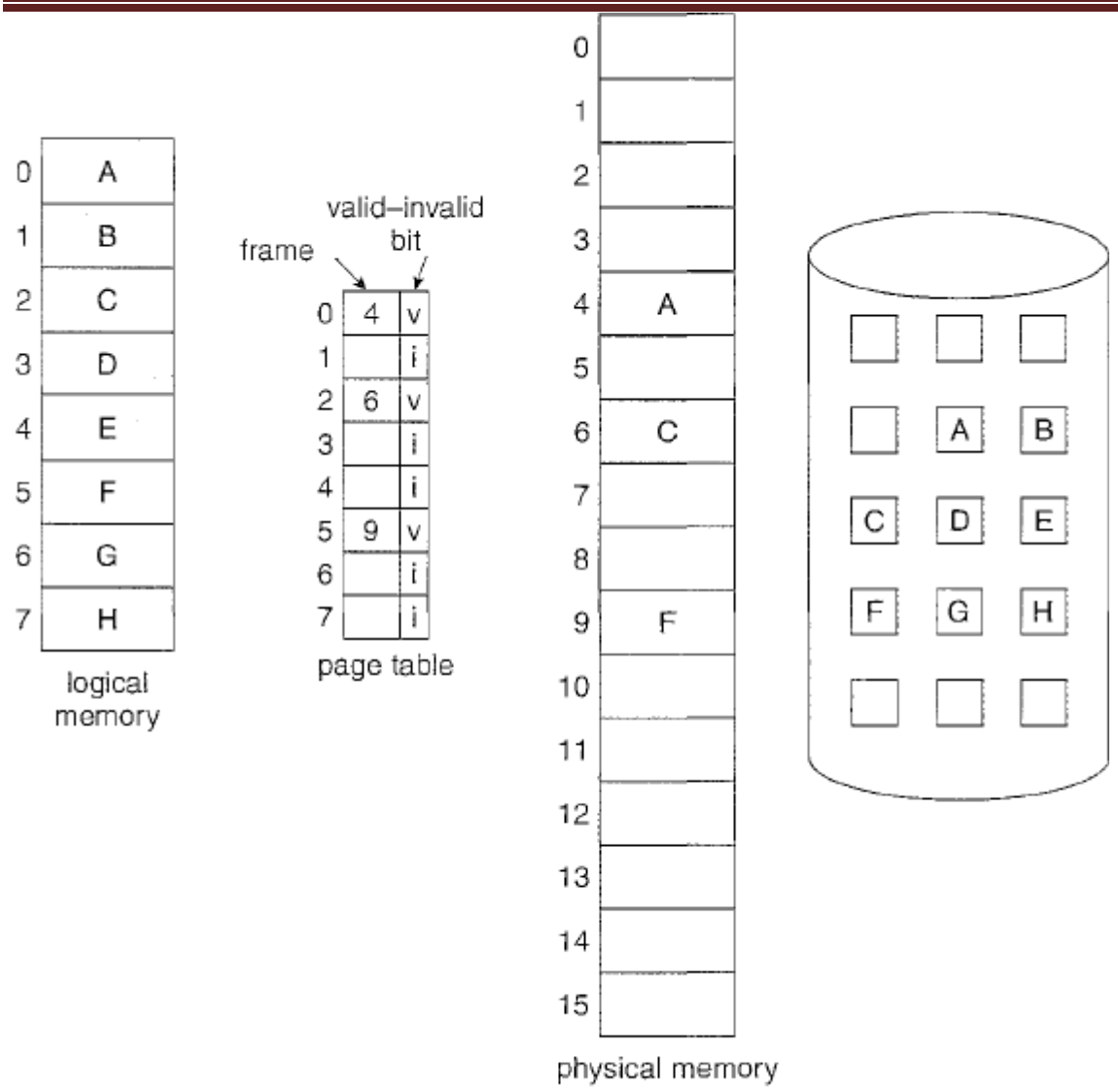


Figure 9.5 Page table when some pages are not in main memory.

- When the process tries to access a page that was not brought into memory it causes **page-fault trap**
- Procedure for handling page fault is
    1. Internal table is checked for respective page to determine whether the reference was a valid or an invalid memory access.
    2. If the reference was invalid, process is terminated. If it was valid, it is page in.
    3. Free frame is searched
    4. Disk operation is scheduled to read the desired page into the newly allocated frame.

5. When the disk read is complete, internal table of the process is modified. Page table is also modified to indicate that page is now in memory.
6. Instruction that was interrupted by the trap is restarted. The process can now access the page as though it had always been in memory.

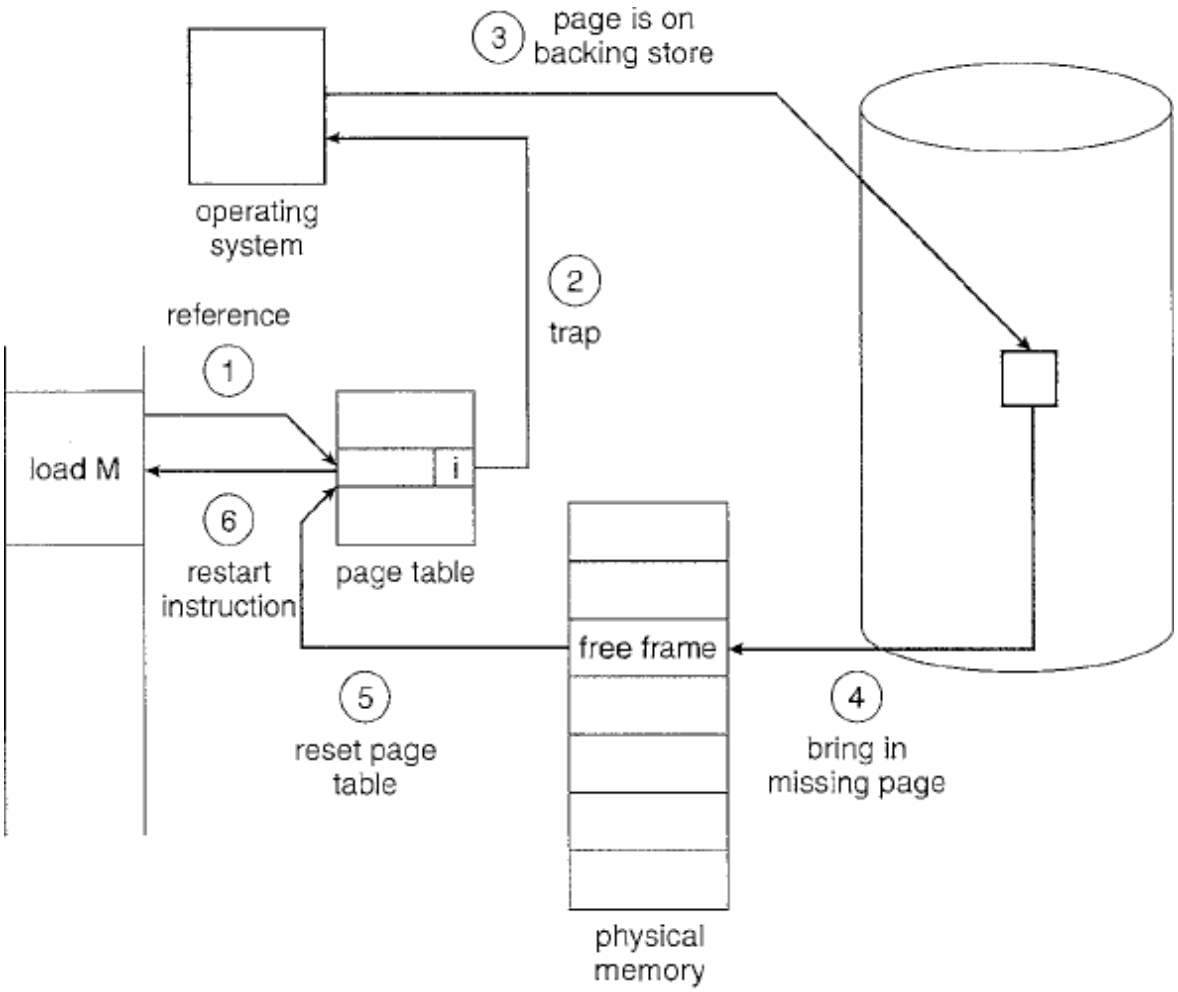


Figure 9.6 Steps in handling a page fault.

- ***"Pure demand paging"*** never brings a page into memory until it is required.

Performance of Demand Paging

- Computation of **effective access time** for a demand-paged memory.
- Let p be the probability of a page fault ($0 \leq p \leq 1$). And ma is memory access time.
- We would expect p to be close to zero that is, we would expect to have only a few page faults.
    - The effective access time is then be effective access time = $(1 - p) \times ma + p \times$ page fault time.

- To compute the effective access time, we must know how much time isneeded to service a page fault.

  Three major components of page fault

  1. Service the page-fault interrupt.
  2. Read in the page.
  3. Restart the process.

- The first and third tasks may take from 1 to 100 microseconds each.
- Page-switch time is close to 8 milliseconds.
- we take an average page-fault service time of 8 milliseconds and a memory-access time of 200 nanoseconds, then the effective access time in nano seconds is

effective access time = (1 - *p)* x (200) + *p* (8 milliseconds)

= (1 - *p)* x 200 + *p x* 80,00000

= 200 + 7,999,800 x *p.*

- Effective access time is directly proportional to page-fault rate.
- It is important to keep the page-fault rate low in a demand-paging system. Otherwise the effective access time increases, slowing process execution dramatically.

Copy-on-Write

- Copy-on-write works by allowing the parent and child processes initially to share the same pages. These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created.
- Copy-on-write is illustrated in below Figures, which show the contents of the physical memory before and after process 1 modifies page C.

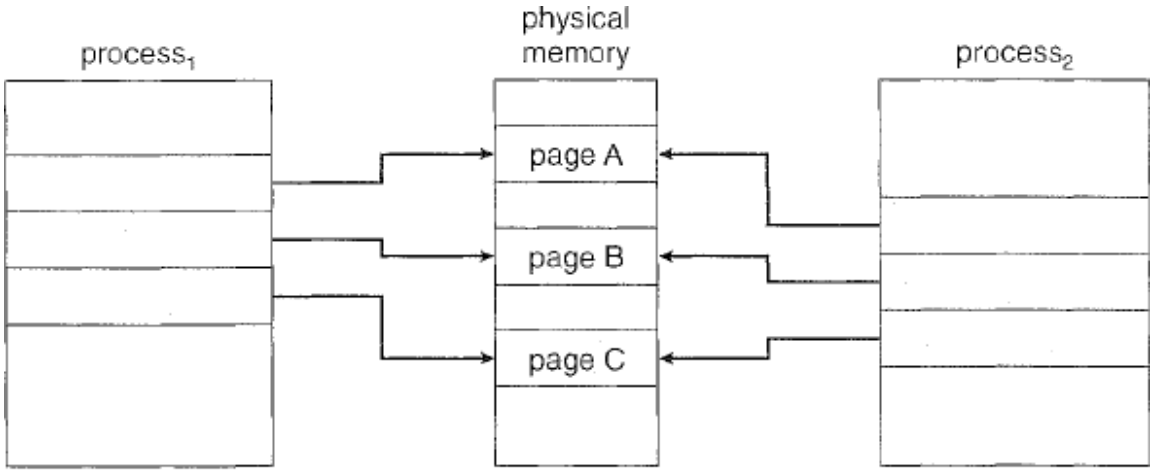


Figure 9.7 Before process 1 modifies page C.

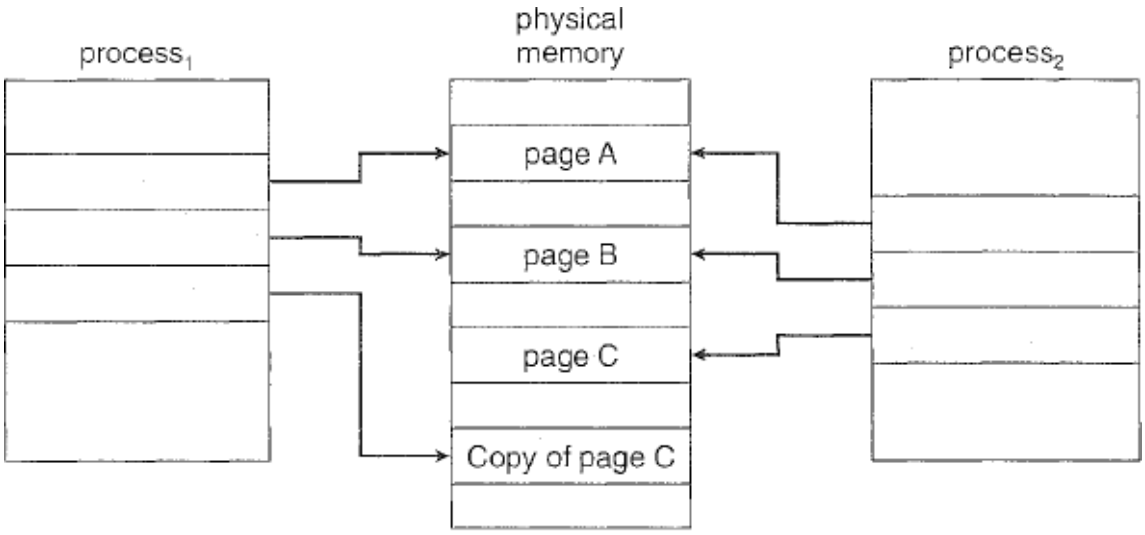


Figure 9.8 After process 1 modifies page C.

- Many operating systems provide a **pool** of free pages to satisfy copy-on-write requests. Files are allocated using ***Zero-fill0on demand*** technique.
- Zero-fill-on-demand pages have been zeroed-out before being allocated, thus erasing the previous contents.

Page Replacement

- Demand paging shares the I/O by not loading the pages that are never used.
- Demand paging also improves the degree of multiprogramming by allowing more process to run at the sometime.

- Page replacement policy deals with the solution of pages in memory to be replaced by a new page that must be brought in. When a user process is executing a page fault occurs.
- The hardware traps to the operating system, which checks the internal table to see that this is a page fault and not an illegal memory access.
- The operating system determines where the derived page is residing on the disk, and this finds that there are no free frames on the list of free frames.
- When all the frames are in main memory, it is necessary to bring a new page to satisfy the page fault, replacement policy is concerned with selecting a page currently in memory to be replaced.
- The page i,e to be removed should be the page i,e least likely to be referenced in future.

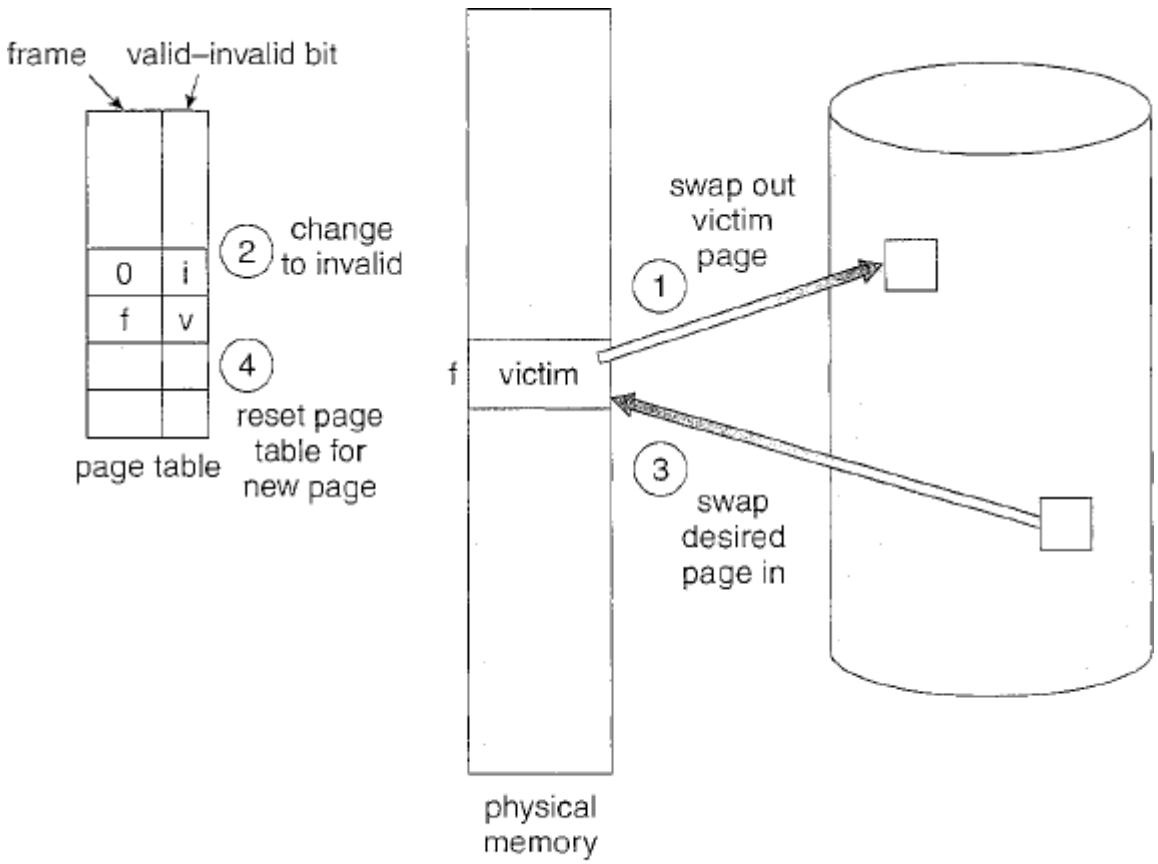


Figure 9.10 Page replacement.

Working of Page Replacement Algorithm

1. Find the location of derived page on the disk.
2. Find a free frame

a. If there is a free frame, use it.
b. Otherwise, use a replacement algorithm to select avictim.
c. Write the victim page to the disk; change the page and frame tables accordingly.

3. Read the desired page into the free frame; change the page and frame tables.
4. Restart the user process.

FIFO Page Replacement

- The simplest page-replacement algorithm is a **first-in, first-out** (FIFO) algorithm.
- In this case when a page must be replaced, the oldest page is chosen.
- Example reference string, is 7, 0,1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1for a memory with three frames.
- The first three references (7,0,1) cause page faults and are brought into empty frames.
- The next reference (2) replaces page 7, because page 7 was brought in first.
- Since 0 is the next reference and 0 is already in memory, we have no fault for this reference.
- The first reference to 3 results in replacement of page 0.This process continues as shown in below fig resulting in 14 page faults.

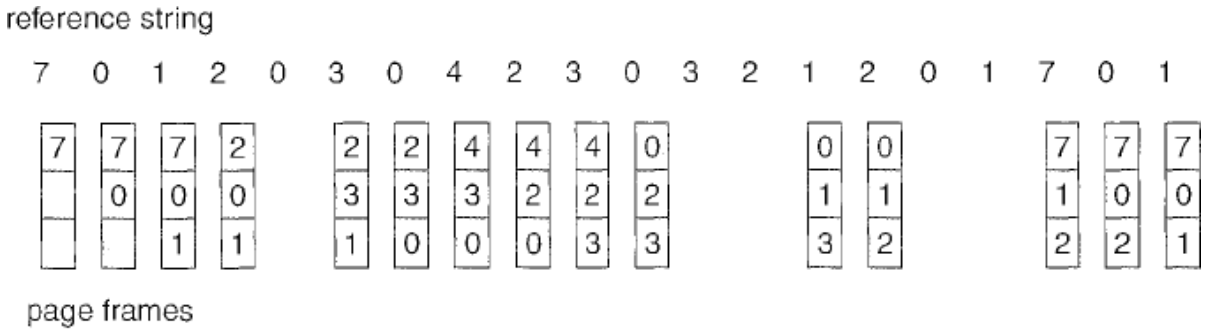


Figure 9.12 FIFO page-replacement algorithm.

- Page fault rate is very high in FIFO algorithm.

Belady's anomaly:

For some page-replacement algorithms, the page-fault rate may *increase* as the number of allocated frames increases.

Ex:- Number of faults for four frames is greater than number of faults for three frames. This unexpected result is called as Belady's Anomaly.

Optimal Page Replacement

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms.
- It will never suffer from Belady's anomaly.
- It Replace the page that will not be used for the longest period of time.
- Fig
- For the reference string shown in the fig optimal page replacement gives nine page faults
- The first three references cause faults that fill the three empty frames. The reference to page2 replaces page 7, because 7 will not be used until reference 18, whereas page0 will be used at 4, and page 1 at 14.
- Optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

LRU Page Replacement

- It replaces the page that *has not been used* for the longest period of time (Figure 9.14).
- LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.

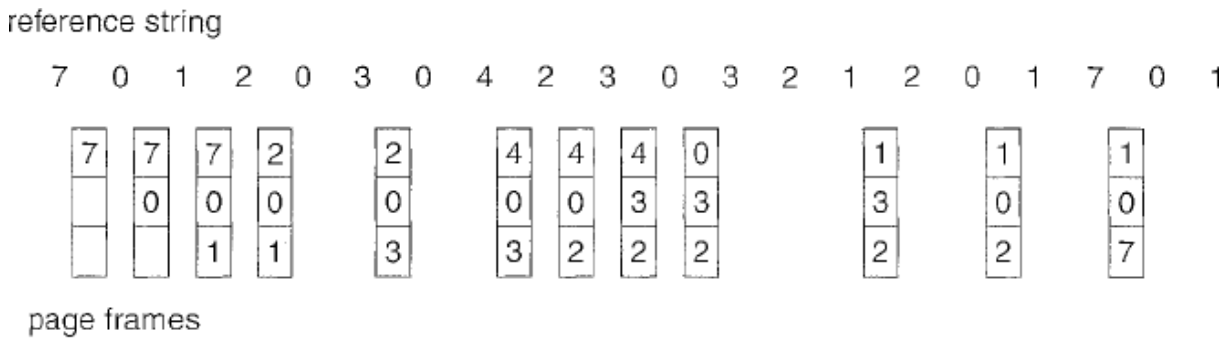


Figure 9.15 LRU page-replacement algorithm.

- LRU results in 12 page faults
- LRU Page-replacement may require hardware assistance for implementation.
- Two types of implementations are
  2. Counters
  3. Stack.
- Like optimal replacement, LRU does not suffer from Belady's Anomaly. Both belong to a class of algorithms called ***Stack algorithms.***

LRU-Approximation Page Replacement

- Few computer systems provide sufficient hardware support for true LRU page replacement.
- Many systems provide some help, in the form of a reference bit.
- The reference bit for a page is set by the hardware whenever that page is referenced.
- Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware.

Additional-Reference-Bits Algorithm

- We can gain additional ordering information by recording the reference bits at regular intervals.
- Operating system stores most recent 8 reference bits for each page in an 8-bit byte in the page table entry.
- At periodic intervals, the Operating system right shifteach of the reference bytes by one bit and discards low-order bit.

- These 8-bit shift registers contains history of page use for the last eight time periods.
- At any given time, the page with the smallest value for the reference byte is the LRU page.

Second-Chance Algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm.
- When a page has been selected, however, we inspect its reference bit. If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time.
- Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances).
- Circular queue is used to implement this

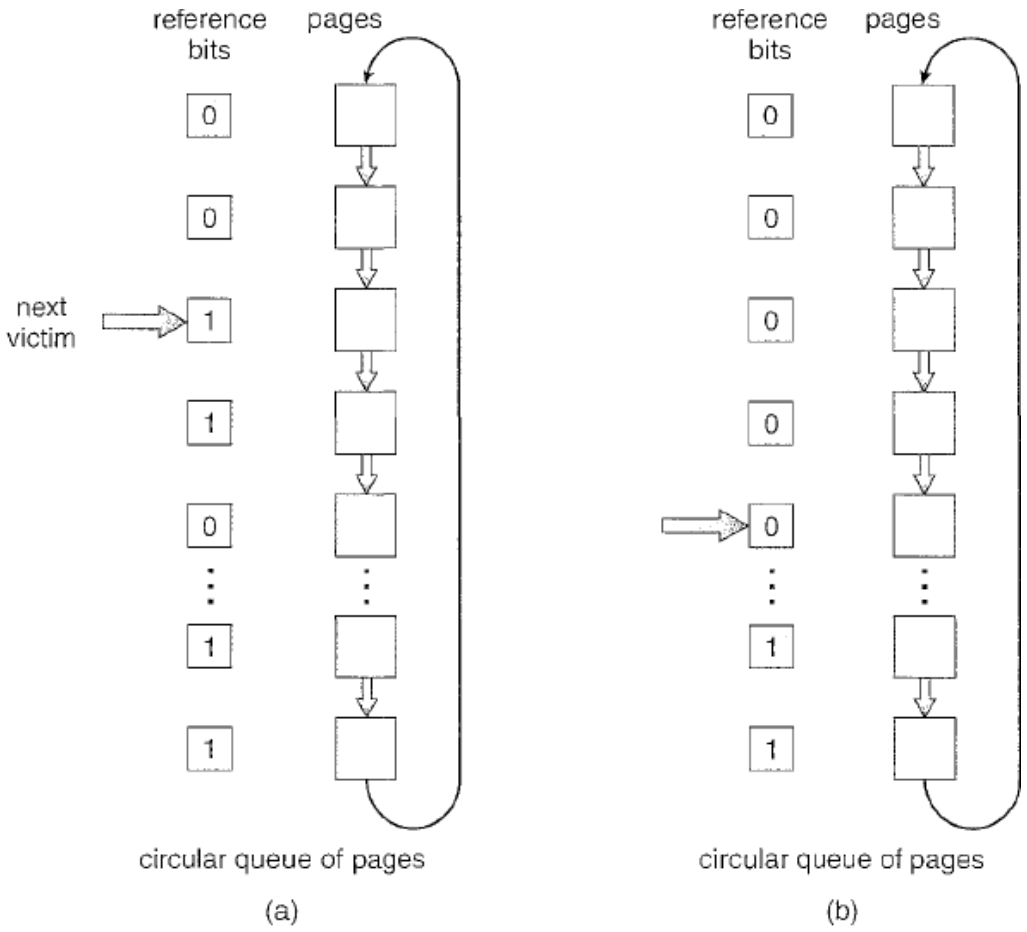


Figure 9.17 Second-chance (clock) page-replacement algorithm.

Enhanced Second-Chance Algorithm

- We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair.
- With these two bits, we have the following four possible classes:
  2) (0, 0) neither recently used nor modified—best page to replace
  3) (0, 1) not recently used but modified—not quite as good, because the page will need to be written out before replacement
  4) (1., 0) recently used but clean—probably will be used again soon
  5) (1,1) recently used and modified—probably will be used again soon, and the page will be need to be written out to disk before it can be replaced.
- Each page is in one of these four classes.

- This algorithm searches the page table in a circular fashion, looking for the first page it can find in the lowest numbered class. i.e., it first makes a pass looking for a(0,0) and then if it can't find one it makes another pass looking for (0,1) etc.

Counting-Based Page Replacement

- Counter is used to develop following schemes

b) **Least frequently used (LFU) page-replacement algorithm** It requires that the page with the smallest count be replaced.

c) **Most frequently used** (MFU) It is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

- The implementation of these algorithms is expensive.

Page-Buffering Algorithms

- There are number of page-buffering algorithms that can be used in conjunction with the before-mentioned algorithms, to improve overall performance.
- Systems keep a pool of free frames. When a page fault occurs, a victim frame is chosen as before.
- The desired page is read into a free frame from the pool before the victim is written out. This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written put, its frame is added to the free-frame pool.

Applications and Page Replacement

- Some applications (database programs) Understand their data accessing and caching needs better than the general purpose operating system, and should be given their own memory management.
- Sometimes such programs are given a raw disk partition to work with, containing raw data blocks and no file system structure. It is then up to the application to use this disk partition as extended memory or for whatever other reasons it see fit.

Allocation of Frames

- The allocation policy in a virtual memory controls the operating system decision regarding the amount of real memory to be allocated to each active process.
- In a paging system if more real pages are allocated, it reduces the page fault frequency and improved turnaround throughput.

- If too few pages are allocated to a process its page fault frequency and turnaround times may deteriorate to unacceptable levels.
- The minimum number of frames per process is defined by the architecture, and the maximum number of frames. This scheme is called equal allocation.
- With multiple processes competing for frames, we can classify page replacement into two broad categories

a) Local Replacement: requires that each process selects frames from only its own sets of allocated frame.

b) Global Replacement: allows a process to select frame from the set of all frames.

Minimum Number of Frames

- We cannot, allocate more than the total number of available frames. We must also allocate at least a minimum number of frames.
- One reason for allocating at least a minimum number of frames involves performance. As the number of frames allocated to each process decreases, the page-fault rate increases, slowing process execution.
- The minimum number of frames is defined by the computer architecture.

Allocation Algorithms

Equal Allocation

- If there are m frames available and n processes to share them, each process gets m/n frames and leftovers are kept in a free-frame buffer pool.

Proportional Allocation

- Allocate the frame proportionally to the size of the process, relative to the total size of all processes.
- Variations on proportions allocation could consider priority of process rather than just their size.

Global versus Local Allocation

- With local replacement the number of pages allocated to process is fixed, and page replacement occurs only amongst the pages allocated to this process.

- With global replacement, any page may be a potential victim, whether it currently belongs to the process seeking a free frame or not.
- Local page replacement allows processes to better control their own page fault rates.
- Global page replacement is more efficient, and more commonly used approach.

Thrashing

- If the process does not have the enough number of frames it needs to support pages in active use, it will quickly page-fault at this point, it must replace some page. Since all pages are in active use, it must replace a page that will be needed again right away. So it quickly faults again and again.
- This high paging activity is called ***"Thrashing". "A process is thrashing if it is spending more time paging than executing"***

Cause of Thrashing

- Early process scheduling schemes would control the level of multiprogramming allowed based on CPU utilization, adding in more process when cpu utilization was low.
- If a process needs more frames it starts faulting and taking frames away from other process. These process need those pages, so they also fault, taking frames from other processes. Because of this ready queue empties. CPU utilization decreases.
- CPU scheduler sees this and increases degree of multiprogramming. New process takes frames from running process causing more page faults. CPU utilization drops even further. Because of this effective memory access time increases.

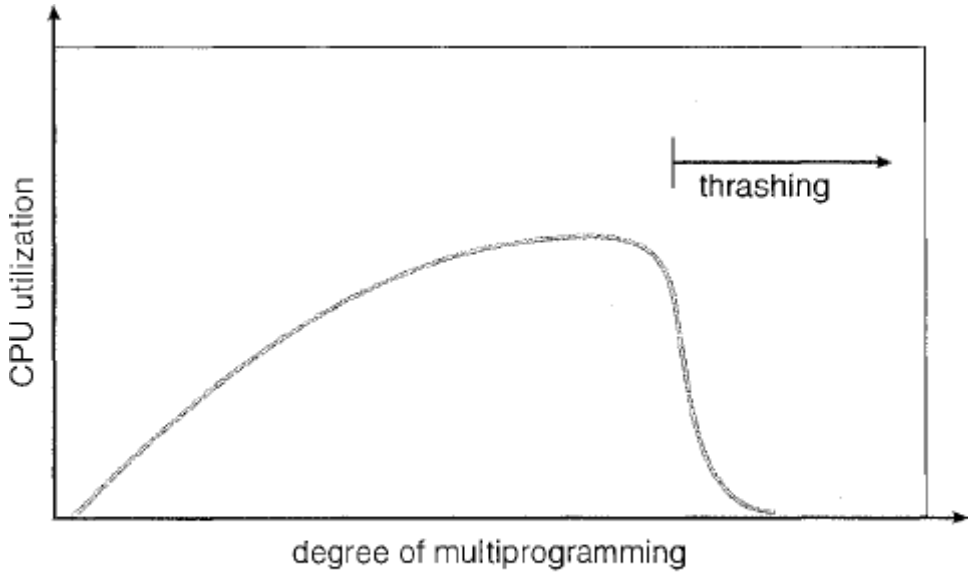


Figure 9.18 Thrashing.

- To prevent thrashing, we must provide a process with as many frames as they really need "right now". There are several techniques to achieve this.
- The working-set strategy defines the Locality model of process execution.
- The locality model states that, as a process executes, it moves from locality to locality. A locality is a set of pages that are actively used together (Ex: When one function exits and another is called)

Working-Set Model

working-set model is based on the assumption of locality. This model uses a parameter, **"Δ"**to define the working-set window. The idea is to examine the most recent Δ page references. The set of pages in the most recent Δ page references is the working set.

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 Δ time units after its last reference.

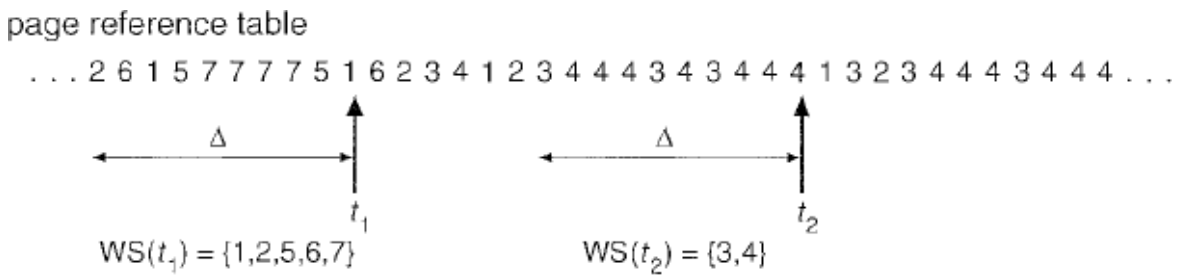


Figure 9.20 Working-set model.

For example, given the sequence of memory references shown in above Figure, if $\Delta = 10$ memory references, then the working set at time *t1*is {1, *2*, 4,6, *7}*. By time *t2*, the working set has changed to {3, 4}.

The accuracy of the working set depends on the selection of Δ . If A is too small, it will not encompass the entire locality; if Δ is too large, it may overlap several localities.

MODULE 5

File System, Implementation of File System

FILE **STRUCTURE**

File System, Implementation of File System:

File system

File concept

Access methods

Directory and Disk structure

File system mounting

File sharing

Implementing File system

File system structure

File system implementation

Directory implementation

Allocation methods

Free space management.

Secondary Storage Structure

Protection

Mass storage structure

Disk structure

Disk attachment

Disk scheduling

Disk management

Protection: Goals of protection

Principles of protection

Domain of protection

Access matrix.

File Concept

- Computers can store information on various storage media, operating system provides a uniform logical view of information storage. This logical storage unit is called as a file.
- Files are mapped by operating system onto physical devices. These storage devices are nonvolatile, so contents are persistent through power failures and system reboots.
- File is a named collection of related information that is recorded on secondary storage
- **File structure** : File has certain structure according to its type
  - ***Text File*** : Sequence of characters organized into lines
  - ***Source File*** : Sequence of subroutines and functions.
  - ***Object File*** : Sequence of bytes organized into blocks understandable by the system linker.
  - ***Executable File*** : Series of code sections that the loader can bring into memory and execute.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ***Name:*** The symbolic file name is the only information kept in human readable form,.
- ***Identifier*** : Identifies the file within the file system.
- ***Type:*** Needed for systems which support multiple file types.
- ***Location:*** It is a pointer to a device and to the location of the file on that device.
- ***Size:*** The current size of the file (in bytes, words, or blocks)
- ***Protection:*** Access-control information determines who can do reading, writing, executing.
- ***Time, date, and user identification :***It includes details about creation time, modification time and last use time of a file.

File Operations

Six basic file operations are :

Creating a file : Two steps are necessary 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 :

- System calls is used to perform the write operation.
- Name of the file and information are two required data to complete the write operation.

Reading a file :

- System call is used to read a file.
- Name of the file is used to search the file. System needs to keep a read pointer to the location in the file where the next read is to take place.

Delete a file:

- System will search the directory, which file to be deleted. If directory entry is found, it releases all file space. That free space can be reused by other files.

Truncating a file:

- The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, truncate function allows all attributes to remain unchanged except for all file length.

Repositioning within a file :

- The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.
- This operation is also known as **seek.**

File Types

- File type is included as a part of the filename
- File name is split into two parts- a ***name*** and ***extension.***
- Following table gives the file type with usual extension and function.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats

library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

Sequential Access

- Sequential access is the simplest access method. Information in the file is sequentially accessed, i.e., one record after the other.
- Editors, compilers usually access files in this method.
- Read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- Write operation appends to the end of the file and advances to the end of the newly written material.

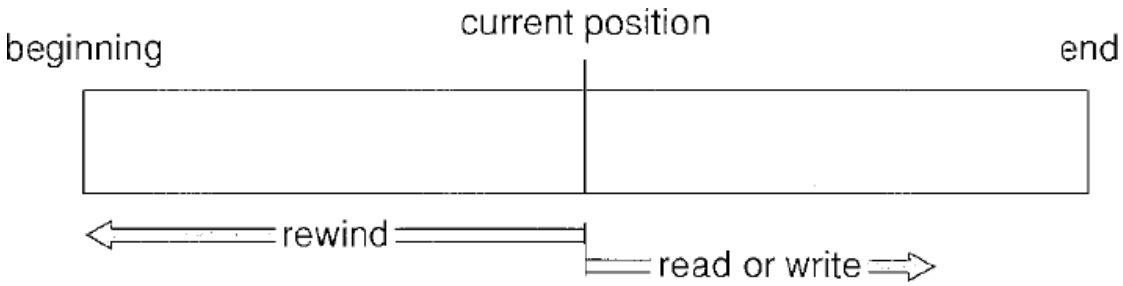


Figure 10.3 Sequential-access file.

- Such a file can be reset to beginning.

Direct Access

- A file is made up of fixed length **logical records** that allow programs to read and write records rapidly in no particular order.
- It is based on disk-model of a file.
- There are no restriction on the order of reading or writing for a direct-access file. We may read block 14, then read block 43 and then write block 7.
- Direct access files are use full for immediate access to large amounts of information

  Ex: Data bases.
- In this file operations must include ***block number*** as a parameter.
- This block number is normally ***relative block number***

Other Access Methods

- These methods involve the construction of an index for the file.
- Index contains pointers to the various blocks. To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.
- This structure allows to search a large file.
- For very large files index for the index file can be created.

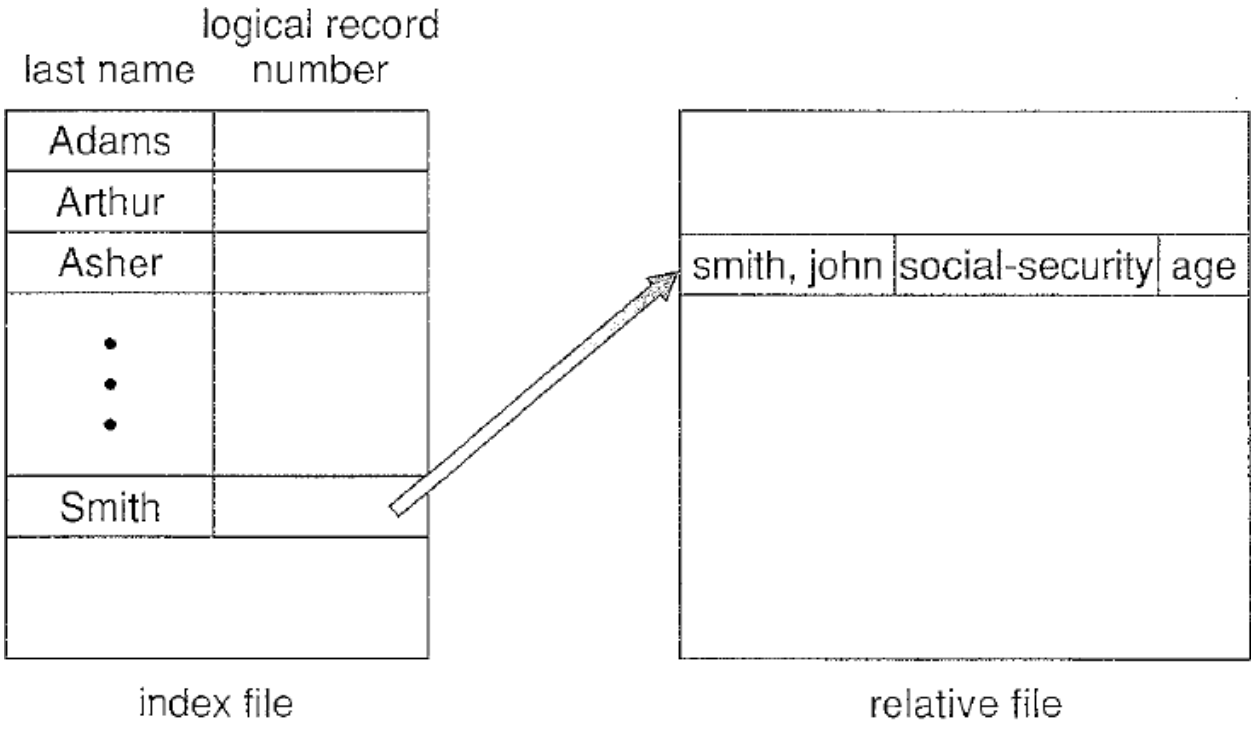


Figure 10.5 Example of index and relative files.

Directory Structure

- Directory can be viewed as a symbol table that translates file names into their directory entries.
- Operations performed on a directory are
  - **Search for a file :** We should be able to search a directory structure to find the entry for a particular file
  - **Create a file :** New files need to be created and added to the directory.
  - **Delete a file :** When a file is no longer needed, we want to be able to remove it from the directory.
  - **List a Directory :** We need to be able to list the files in a directory.
  - **Rename a file :** We must be able to change the name when the contents or use of the file changes.
  - **Traverse the file system :** We must be able to access every directory and every file within a directory structure.

Different Types of Directory Structures are

Single-Level Directory

- It is the simplest directory structure
- All files are contained in the same directory, which is easy to support and understand
- Limitation of single-level directory structure is
  - Not suitable for a large number of files and more than one user.
  - Because of single directory, files require unique file names.
  - Difficult to remember the names of all the files as the number of files increases.

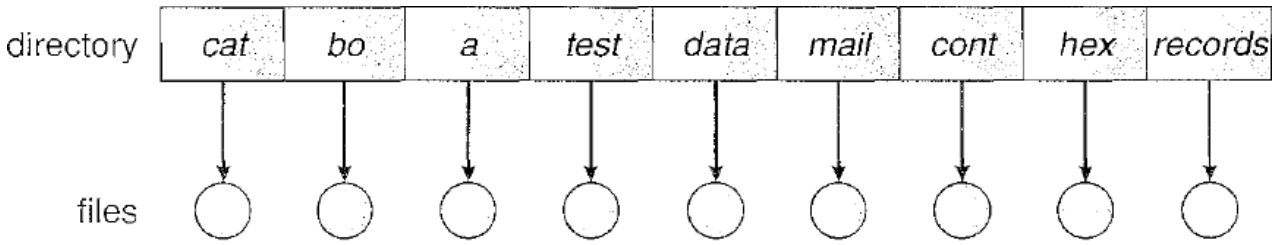


Figure 10.8 Single-level directory.

Two-Level Directory

- In two level directory, each user has his own directory called as User file directory (UFD)
- When a user refers to a particular file, only his own UFD is searched.
- Different users may have files with the same name. But file names within each UFD must be unique.

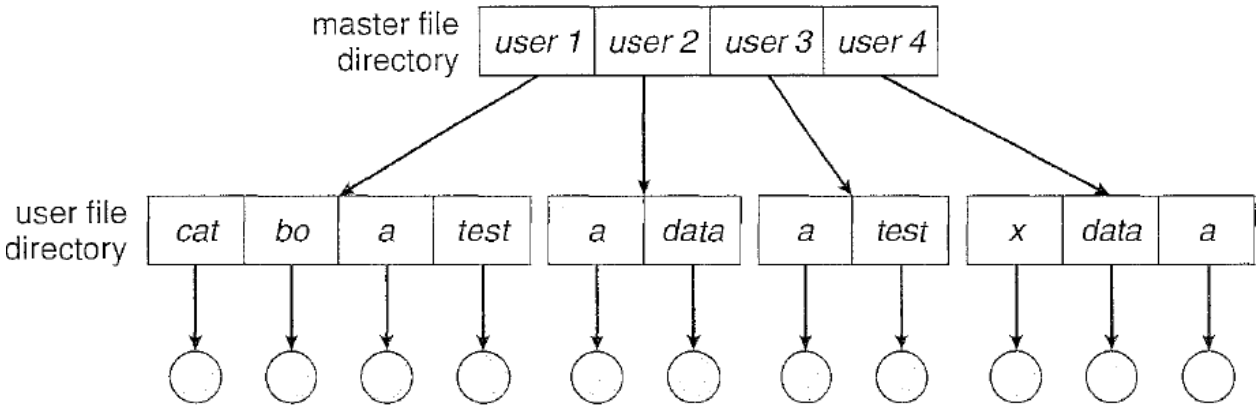


Figure 10.9 Two-level directory structure.

- To create a file for a user, operating system searches only that users directory to check whether another file of that name exists.
- Disadvantage of the two level directory is-it isolates one user from another-which is a major problem when the users want to co-operate on some task and to access one another's files.

Tree-Structured Directories

- It allows users to create their own subdirectories and to organize their files accordingly. A subdirectory contains a set of files or subdirectories.
- All the directories have the same internal format. One bit in each directory entry defines the entry as a file or as a subdirectory.
- In normal use, each process has a current directory. The **current directory** should contain most of the files that are of current interest to the user.
- When reference is made to a file, the current directory is searched. Path name is used to search for any operation on file with another directory.
- Two types of path name are
  - ***Absolute Path Name :*** Begins at the root and follows a path down to the specified file, giving the directory names on the path.
  - ***Relative Path Name :*** It defines a path from the current directory.
- When a request is made to delete a directory, all that directory files and subdirectories are also be deleted.

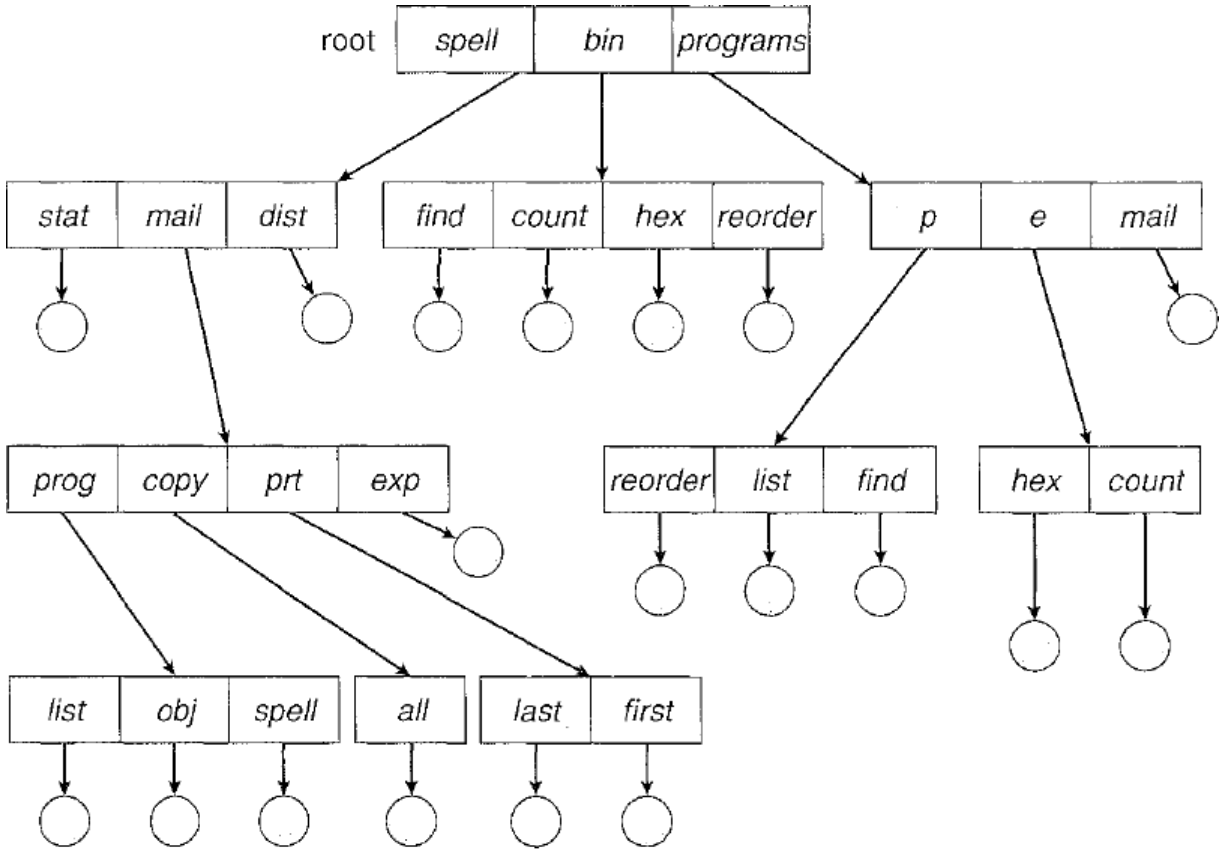


Figure 10.10 Tree-structured directory structure.

- In a tree-structured directory, users can be allowed to access, other users files.

Acyclic-Graph Directories

- Acyclic-graph (Graph with no cycles) Allows directories to share subdirectory and files.
- The same subdirectory may be in two different directories.
- With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.

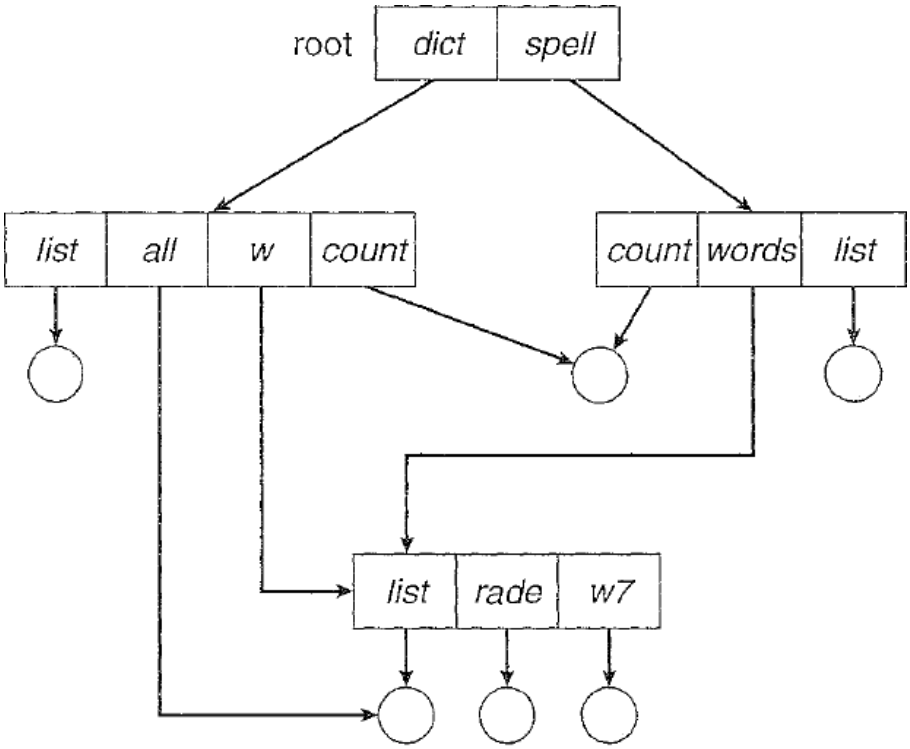


Figure 10.11 Acyclic-graph directory structure.

- Shared files and subdirectories can be implemented by using links.
- Link is implemented as a absolute or relative path name.
- An acyclic graph directory structure is more flexible than a simple tree structure but sometimes it is more complex.

General Graph Directory

- A problem with using an acyclic-graph structure is ensuring that there are no cycles.
- Whenever we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure.

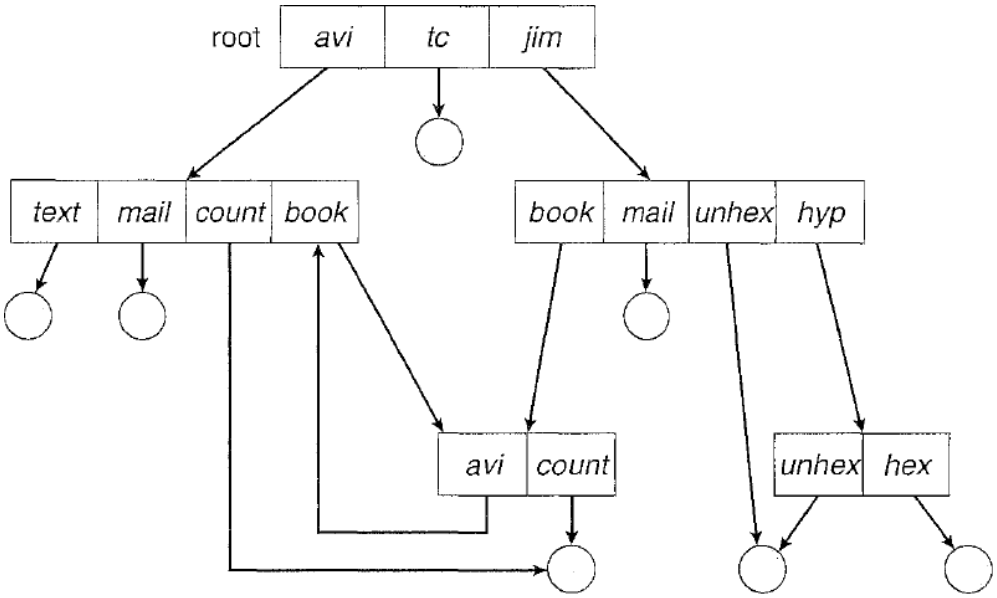


Figure 10.12 General graph directory.

File-System Mounting

- File system must be mounted before it can be available to process on the system.
- Procedure for mounting a file system is
  - Mount point is an empty directory at which the mounted file system will be attached
  - Name of the device and location within the file structure at which to attach the file system is required.
  - Operating system verifies that the device contains a valid file system
  - Device driver is used by Operating system for these verifications
  - Finally operating system mounts the file system at a specified mount point.

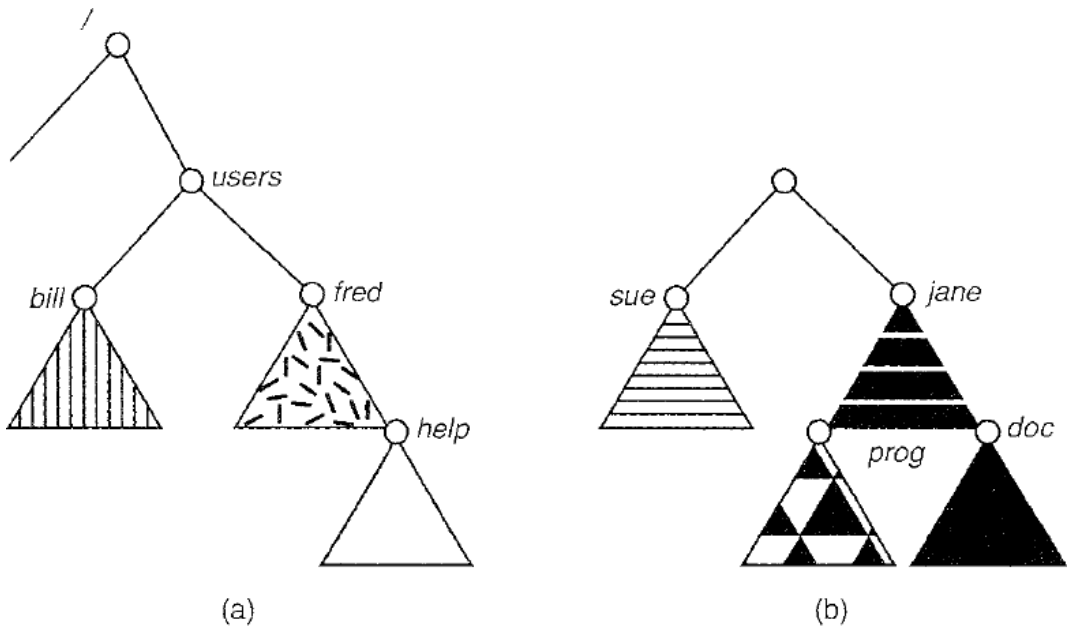


Figure 10.13 File system. (a) Existing system. (b) Unmounted volume.

- In a fig (a), an existing file system is shown and in fig (b) an un-mounted partition residing an /device/dsk is shown. At this point, only the files on the existing file system can be accessed.
- Fig : Mount Point shows the effects of the mounting of the partition residing on /device/dsk over user. If the partition is un-mounted, the file system is restored to the situation before mount operation.

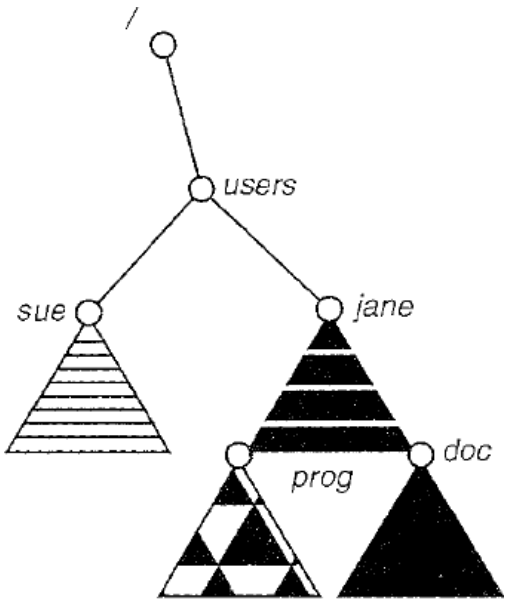


Figure 10.14 Mount point.

File Sharing

Multiple Users

- Given a directory structure that allows files to be shared by users, the operating system must mediate the file sharing.
- The system either can allow a user to access the files of other users by default or it may require that a user specifically grant access to the files.
- To implement sharing and protection, the system maintain more file and directory attributes than on a single user system, most systems support the concept of file owner and group.
- When a user requests an operation on a file, the user ID can be compared to the owner attribute to determine if the requesting user is the owner of the file. Likewise the group ID's can be compared. The result indicates which permissions are applicable.

Remote File Systems :

- Remote sharing of the file system is implemented by using network
- Network allows the sharing of resources. File Transfer Protocol (FTP) is one o the methods used for remote sharing. Other methods re distributed file system and world wide web.
  - **Client-server Model :** System containing the files is the server and the system requesting access to the files is a client. Files are specified on a partition or subdirectory level. A server can serve multiple clients and a client can use multiple servers.
  - **Distributed Information systems:** For managing client server services, distributed information system is used to provide a unified access to the information needed for remote computing.
  - UNIX systems have a wide variety of distributed information methods.
  - The domain name system (DNS) provides host-name-to-network-address translations for the entire internet.

Protection

Information must be protected from a physical damage and improper access i.e., reliability and protection.

Types of Access

- Protection mechanisms provide controlled access by limiting the types offile access that can be made. Access is permitted or denied depending onseveral factors, one of which is the type of access requested. Several differenttypes of operations may be controlled: They are
  - **Read.** Read from the file.
  - **Write**. Write or rewrite the file.
  - **Execute**. Load the file into memory and execute it.
  - **Append**. Write new information at the end of the file.
  - **Delete**. Delete the file and tree its space for possible reuse.
  - **List.** List the name and attributes of the file.

Access Control

- Different users may need different types of access to a file or directory.
- When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.
- Many systems recognize three classifications of users in connection with each file for access control
  - **Owner**. The user who created the file is the owner.
  - **Group.** A set of users who are sharing the file and need similar access is a group, or work group.
  - **Universe.** All other users in the system constitute the universe.
- With the more limited protection classification, only three fields are needed to define protection. Each field is a collection of bits and each bit allows or prevents the access associated with it.

  Ex: INIX System defines three fields of 3 bits each –rwx

  r→ Controls read access

  w→Controls write access

x→Controls execution access

Implementing File System

File system structure

- To provide efficient and convenient access to the disks,Os imposes one or more file system to allow the data to be stored, located and retrieved easily.
- File system itself is composed of many different levels. Layered design is shown below figure.

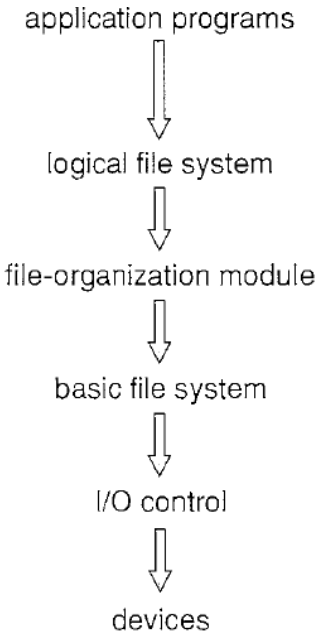


Figure 11.1 Layered file system.

- Each level in the design uses the features of lower levels to create new features for use by higher levels.
- The lowest level, the *I/O control*, consists of **device drivers** and interrupt handlers to transfer information between the main memory and the disk system.
- The **basic file system** issue generic commands to the appropriate device driver to read and write physical blocks on the disk.
- The file-organization module knows about files and their logical blocks, as well as physical blocks. It also includes free-space manager.
- Logical file system manages metadata information. Metadata includes all of the file-system structure except actual data.
- Most operating system supports more than one type of file system.

File-System implementation

Overview on-disk structure

- **Boot control block** (per volume) It can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty.
- A **volume control block** (per volume) contains volume (or partition)details, such as the number of blocks in the partition, size of the blocks, free block count and free-block pointers, and free FCB count and FCB pointers.
- A directory structure per file system is used to organize the files.

In-memory information

It is used for both file-system management and performance improvement via caching. The data are loaded at mount time and discarded at dismount. The structures described below:

- An in-memory mount table contains information about each mounted volume.
- An in-memory directory-structure cache holds the directory information of recently accessed directories.
- The **system-wide open-file table** contains a copy of the FCB of each open file.
- The **per-process open-file table** contains a pointer to the appropriate entry in the system-wide open-file table.
- To create a new file, an application program calls the logical file system. The logical file system knows the format of the directory structures. To create a new file, it allocates a new FCB. The system then reads the appropriate directory into memory, updates it with the new file name and FCB, and writes it back to the disk. Atypical FCB is shown in below Figure

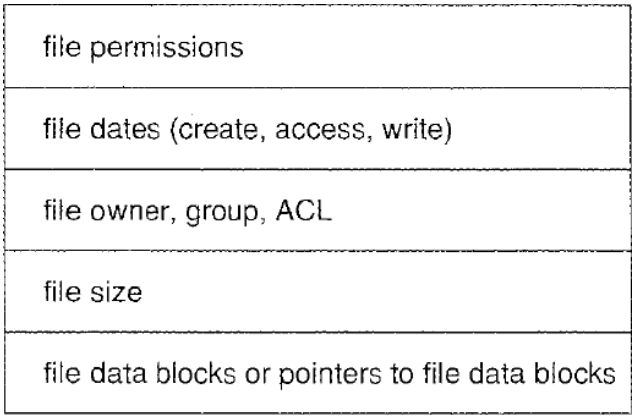


Figure 11.2 A typical file-control block.

- File must be opened before using it for I/O. The open() call passes a file name to the file system.
- The open() system call searches the system-wide open-file table to find the file name given by the user.
- If it is open, a per-process open-file table entry is created pointing to the existing system-wide open-file table.
- When a file is opened, the directory structure is searched for the given file name
- The open() call returns a pointer to the appropriate entry in the per-process file-system table. All file operations are performed via this pointer.
- When a process closes the file, the per-process table entry is removed, and system-wide entry's open count is decremented

Operating structures of file-system implementation are given in the below fig

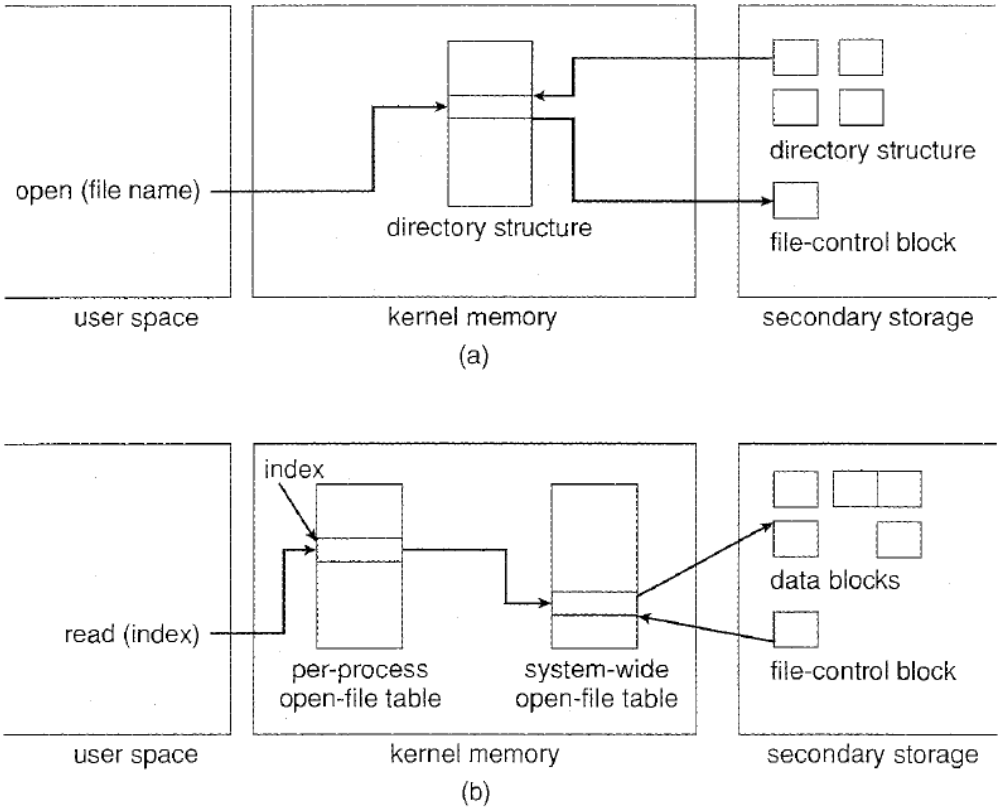


Figure 11.3 In-memory file-system structures. (a) File open. (b) File read.

Partitions and Mounting

- A disk can be sliced into multiple partitions.
- Each partition can be either "raw," containing no file system, or "cooked;' containing a file system.
- Boot information can be stored in a separate partition.
- The root **partition,** which contains the operating-system kernel and some times other system files, is mounted at boot time.
- Other volumes can be automatically mounted at boot or manually mounted later, depending on the operating system.
- Operating system notes in its in-memory mount table structure that a file system is mounted, along with type of the file system.

Virtual File Systems

- Modern operating systems must concurrently support multiple types of file systems.
- Most operating systems, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementation.

- Data structures and procedures are used to isolate the basic system call functionality from the implementation details. The file- system implementation consists of three major layers, as depicted schematically in the below Figure

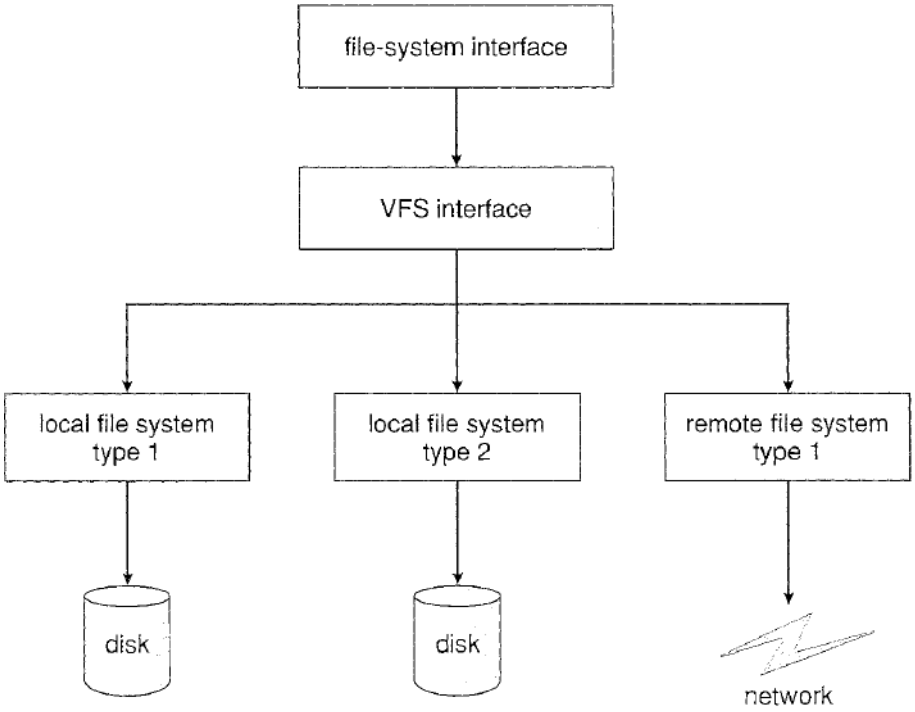


Figure 11.4 Schematic view of a virtual file system.

- The virtual file system (VFS) layer, serves two important functions:
  1. It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.
  2. The VFS provides a mechanism for uniquely representing a file through out a network. The VFS is based on a file-representation structure, called an node, that contains a numerical designator for a network-wide unique file.
- The VFS activates file-system-specific operations to handle local requests according to their file-system types and even calls the NFS protocol procedures for remote requests. File handles are constructed from the relevant vnode sand are passed as arguments to these procedures. The layer implementing the file system type or the remote-file-system protocol is the third layer of the architecture.

Directory implementation

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

Linear List

- The simplest method of implementing a directory is to use a linear list of filenames with pointers to the data blocks.
- To create a new file., we must first search the directory to be sure that no existing file has the same name.
- To delete a file, we search the directory for the named file, then release the space allocated to it.
- A linked list can also be used to decrease the time required to delete a file.
- The real disadvantage of a linear list of directory entries is that finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow.

Hash Table

- Another data structure used for a file directory is a **hash table.** With this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- Insertion and deletion are also fairly straight forward, although some provision must be made for **collisions**—situations in which two file names hash to the same location.
- The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.
- Alternatively, a chained-overflow hash table can be used.

Allocation Methods

The direct-access nature of disks allows us flexibility in the implementation of files, in almost every case, many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed.

Contiguous Allocation

A single set of blocks is allocated to a file at the time of file creation. This is a pre- allocation strategy that uses portion of variable size. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. The figure shows the contiguous allocation method.

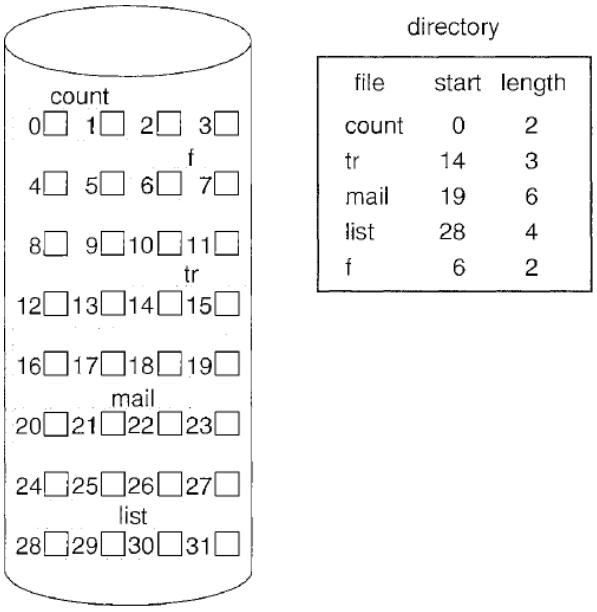


Figure 11.5 Contiguous allocation of disk space.

If the file is n blocks long and starts at location b, then it occupies blocks b, b+1,b+2… .....b+n

The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation. Contiguous allocation algorithm suffers from external fragmentation. Depending on the

amount of disk storage the external fragmentation can be a major or minor problem. Compaction is used to solve the problem of external fragmentation. The following figure shows the contiguous allocation of space after compaction. The original disk was then freed completely creating one large contiguous space. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2…. .....b+n

The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation. Contiguous allocation algorithm suffers from external fragmentation.

Characteristics:

- Supports variable size portion.
- Pre-allocation is required.
- Requires only single entry for a file.
- Allocation frequency is only once.

Advantages:

- Supports variable size problem.
- Easy to retrieve single block.
- Accessing a file is easy. x It provides good performance.

Disadvantage:

- Pre-allocation is required.
- It suffers from external fragmentation.

Linked Allocation:

- It solves the problem of contiguous allocation. This allocation is on the basis of an individual block. Each block contains a pointer to the next block in the chain.
- The disk block can be scattered any where on the disk.
- The directory contains a pointer to the first and the last blocks of the file.
- The following figure shows the linked allocation. To create a new file, simply create a new entry in the directory.

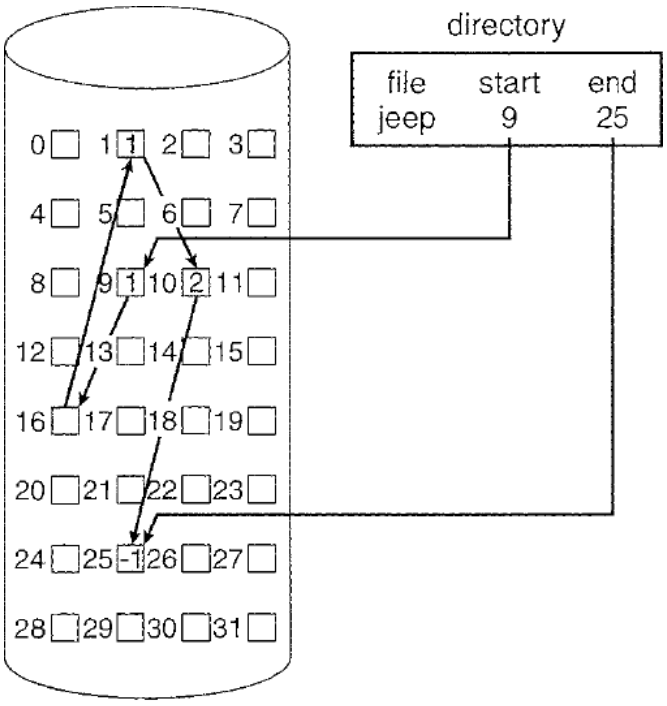


Figure 11.6 Linked allocation of disk space.

There is no external fragmentation since only one block is needed at a time.

The size of a file need not be declared when it is created. A file can continue to grow as long as free blocks are available.

Advantages:

No external fragmentation.

Compaction is never required.

Pre-allocation is not required.

Disadvantage:

Files are accessed sequentially.

Space required for pointers.

Reliability is not good.

Cannot support direct access.

Indexed Allocation:

The file allocation table contains a separate one level index for each file. The index has one entry for each portion allocated to the file. The i^{th} entry in the index block points to the i^{th} block of the file. The following figure shows indexed allocation.

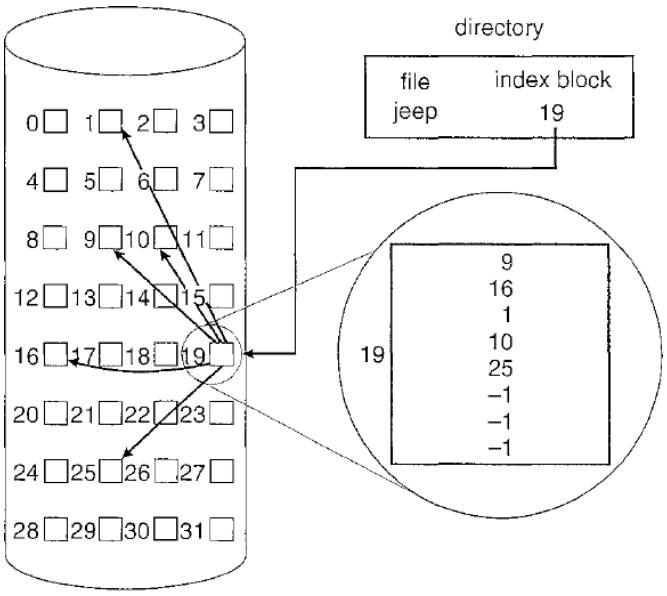


Figure 11.8 Indexed allocation of disk space.

The indexes are not stored as a part of file allocation table rather than the index is kept as a separate block and the entry in the file allocation table points to that block. Allocation can be made on either fixed size blocks or variable size blocks. When the file is created all pointers in the index block are set to nil. When an entry is made a block is obtained from free space manager. Allocation by fixed size blocks eliminates external fragmentation where as allocation by variable size blocks improves locality. Indexed allocation supports both direct access and sequential access to the file.

Advantages:

Supports both sequential and direct access.

No external fragmentation. Faster then other two methods.

Supports fixed size and variable sized blocks.

Disadvantage:

Suffers from wasted space.

Pointer overhead is generally greater

Free space management

- To keep track of free disk space, the system maintains a free space list. The free space list records all the free disks blocks those not allocated to some file or directory. We need a disk allocation table in addition to a file allocation table. Four techniques are in common use-
  - Bit vector
  - Linked list
  - Grouping
  - Counting

Bit Vector

- This method uses a vector contain 1 bit for each block on the disk
- Each entry of a "0" corresponds to a free block and each "1" corresponds to a block in use
- Ex: consider a disk where blocks 2,3,4,4,8 are free and rest of the blocks are allocated. The free space map would be →

1	1	0	0	0	0	1	1	0
0	1	2	3	4	4	6	7	8

- Main advantage of this method is that it is relatively easy to find one or a contiguous group of free blocks. Second advantage is that it is as small as possible and can be kept in main memory.

Linked List

- In linked list, all free space disk blocks are linked, keeping a pointer to the first free block in a special location on a disk and caching it in memory.
- This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first position.
- This method is suitable for all file allocation methods.

Grouping

- It stores the address of *n* free blocks in the first free block
- The first *n-1* of these blocks are actually free. The last block contains the addresses of another *n* free blocks.
- Addresses of a large number of free blocks can be found quickly.

Counting

- It keeps the address of the first free blocks and the number *n* of free contiguous blocks that follow the first block.
- Each entry in the free space list then consists of a disk address and a count.

Mass Storage Structure

Magnetic Disks

- Magnetic disks provide the bulk of secondary storage for modern computer systems.
- Each disk platter has a flat circular shape like a CD. The two surfaces of a platter are Covered with a magnetic material.

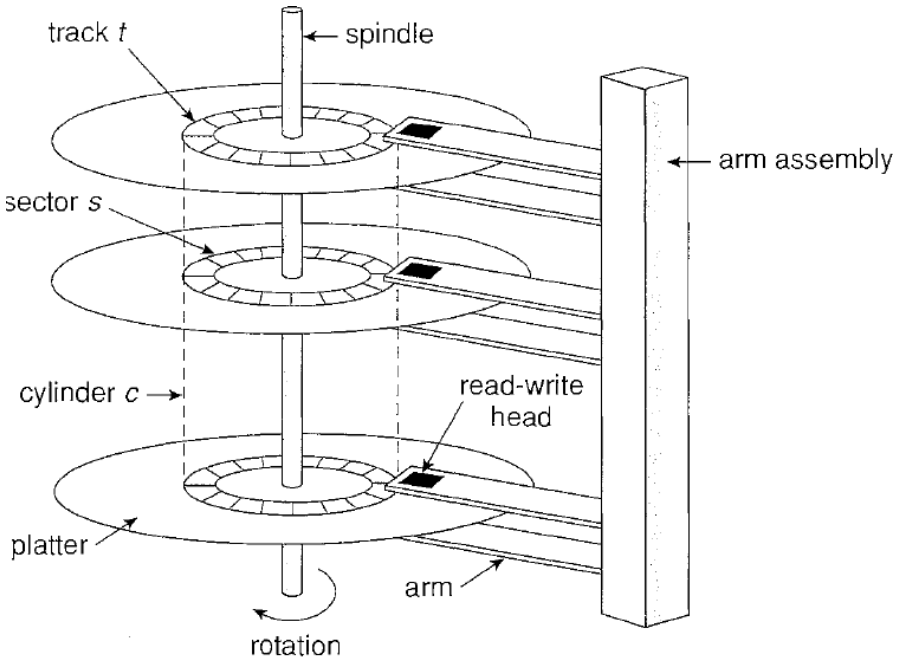


Figure 12.1 Moving-head disk mechanism.

- We store information by recording it magnetically on the platters.
- A Read-write head files just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit.
- Surface of a platter is logically divided into circular track which are subdivided into sectors.
- When the disk is in use, a drive motor spins it at high speed. Most drivers rotate 60 to 200 times per second.
- A disk can be removable, allowing different disks to be mounted as needed
  Ex: floppy disks.

Magnetic Tapes

- Magnetic tape was used as all early secondary-storage medium. Its access time is slow when compared to main memory and magnetic disk.
- Random access to magnetic tape is slower than disk so it is not very useful for secondary storage.
- Tapes are used for backup and to store infrequently accessed data.

Disk Structure

- Disk drives are addressed as large one-dimensional arrays of logical blocks.
- Size of logical blocks is mapped onto the sectors of the disk sequentially.
- These logical blocks are mapped onto the sectors of the disk sequentially.

Disk Attachment

Computers access disk storage in two ways

1. Via I/O ports(host attached storage)
2. Via a remote host in a distributed file system(network attached storage)

Host Attached Storage

- Host-attached storage is storage accessed through local I/O ports.
- High-end workstations and servers use more sophisticated I/O architectures like SCSI and Fiber channel (FC).
- SCSI supports 16 devices on the bus.
- Fiber-channel is a high-speed serial architecture that can operate over optical fiber.
- A wide variety of storage devices are suitable for use as host-attached storage.
- Ex: RAID arrays, CD, DVD….etc

Network-Attached Storage

- A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network.
- Clients access network–attached storage via a remote-procedure-call interface
  Ex: NFS for UNIX and
  CIFS for WINDOWS
- The remote procedure calls are carried via TCP or ODP over all IP network.
- ISCSI is the latest network-attached storage protocol.

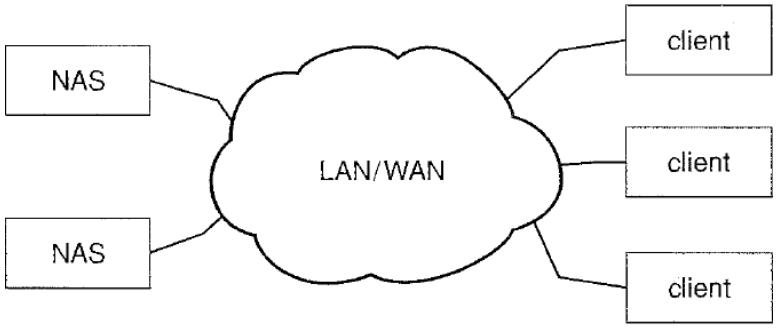


Figure 12.2 Network-attached storage.

Storage-Area Network

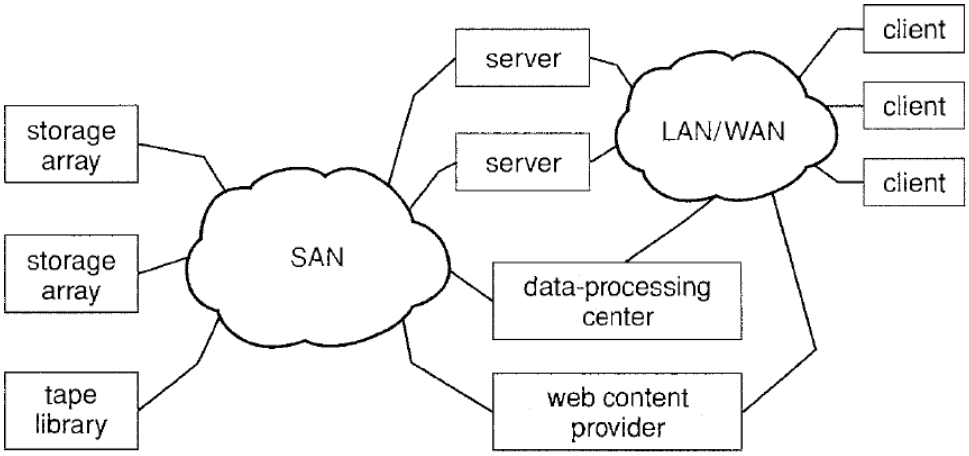


Figure 12.3 Storage-area network.

- A storage-area network(SAN) is a private network connecting servers and storage units as shown in the above figure
- Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts
- A SAN switch allows or prohibits access between the hosts and the storage.
- SAN's have more ports and less expensive ports than storage arrays.
- Fiber channel is used to interconnect multiple storage area networks.

Disk Scheduling

Disk access time has two major components

Seek Time: It is the time for the disk arm to move the heads to the cylinder containing the desired sector.

Rotational Latency: It is the additional time for the disk to rotate the desired sector to the disk head.

Disk Bandwidth: It is the total number of bytes transferred, divided by the total time between the first request for service and the completion of last transfer

Whenever a process needs I/O from the disk, it issues a system call to the operating system of the desired disk drive available, the request can be serviced immediately. If the driver is busy, request will be placed in the queue. When one request is completed, OS chooses another pending request to service next. Several disk scheduling are used for this purpose.

FCFS Scheduling

Queue=98, 83, 37,122,14,124,65,67

Head starts at 53

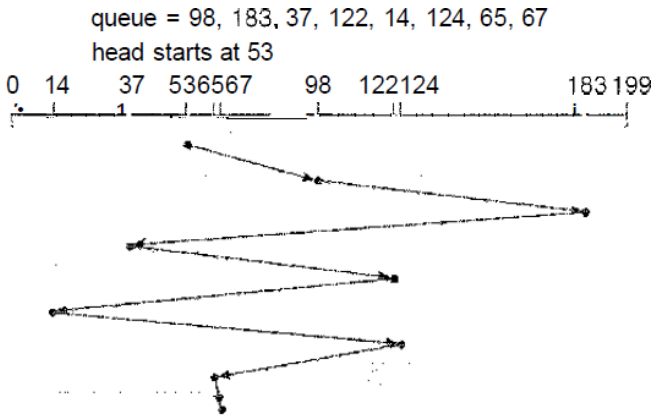


Figure 12.4 FCFS disk scheduling.

- Simplest form of disk scheduling
- Generally doesn't provide fastest service
- Ex: A disk queue with requests for I/O to blocks on cylinders98, 183, 37,122,14,124,65,67 disk heard is initially at cylinder 53

  It will first move from 53 to 98, then to 183,37,122,14,124,65 and 67 for a total head movement of 640 cylinders.

SSTF Scheduling (Short-seek-Time-First)

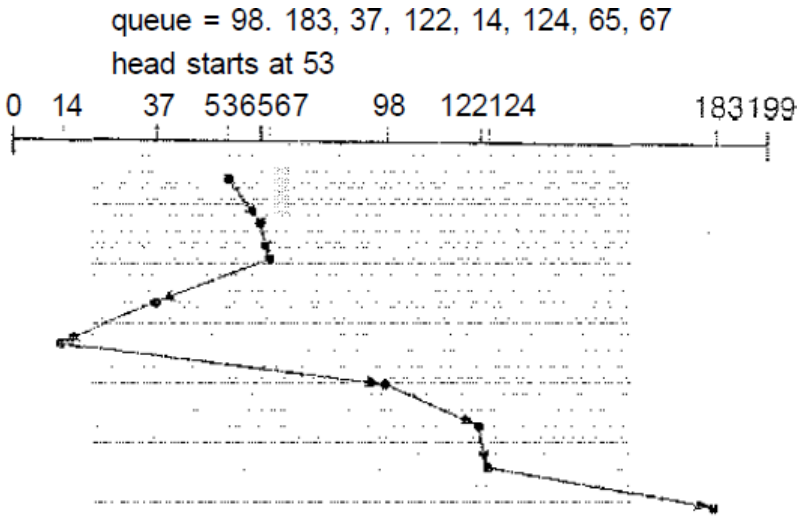


Figure 12.5 SSTF disk scheduling.

- SSTF assumes that it is better to service all the requests close to the current head position before moving the head for away to service other requests.
- SSTF choose the pending request closest to the current head position.

  For the queue 9, 7, 183, 37, 122, 14, 124, 65, 67 with head position = 53

  Closest request to the initial head position is 65. Once we are at cylinder 65 next request served is 67 next is 37

  This scheduling results in a total head movement of only 236 cylinders
- SSTF may cause starvation of some process.

SCAN Scheduling (Elevator algorithm)

- In SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it get to other end of the disk. At the other end, the direction of head movement is reversed and servicing continues.
- Head continuously scans back and forth across the disk.
- Consider requests 98, 183, 37, 122, 14, 124, 65 and 67 head position = 53
- For this algorithm we need to know the direction of head movement
- If the disk arm is moving towards 0, the head will service 37 and then 14.
- At cylinder 0, the arm will reverse and move towards the other end servicing the requests at 65, 67, 98, 122, 124 and 183

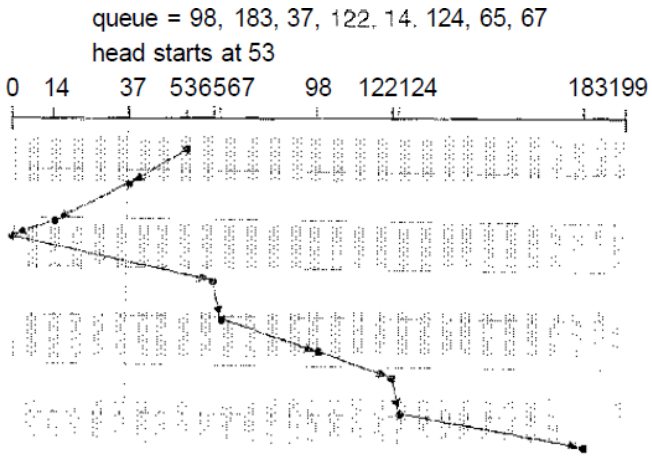


Figure 12.6 SCAN disk scheduling.

C-Scan Scheduling

- Circular SCAN (C-SCAN) Scheduling is a variant of SCAN designed to provide a more uniform wait time.
- C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

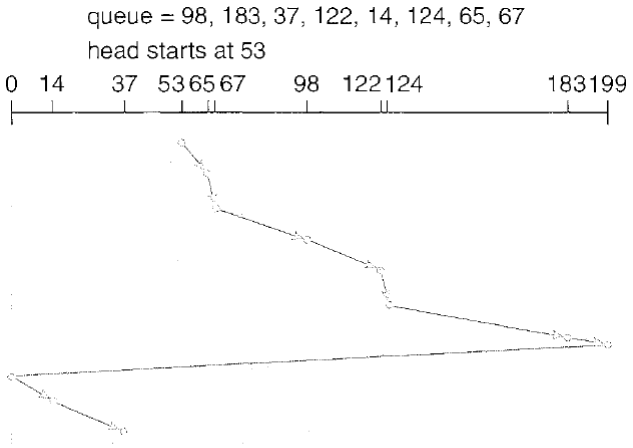


Figure 12.7 C-SCAN disk scheduling.

Look Scheduling

- In this disk arm does not move across the full width of the disk.
- The arm goes only as far as the final request in each direction .Then it reverses direction immediately, without going all the way to the end of the disk. Version of SCAN and C-SCAN that follows this pattern are called LOOK and C-LOOK Scheduling.

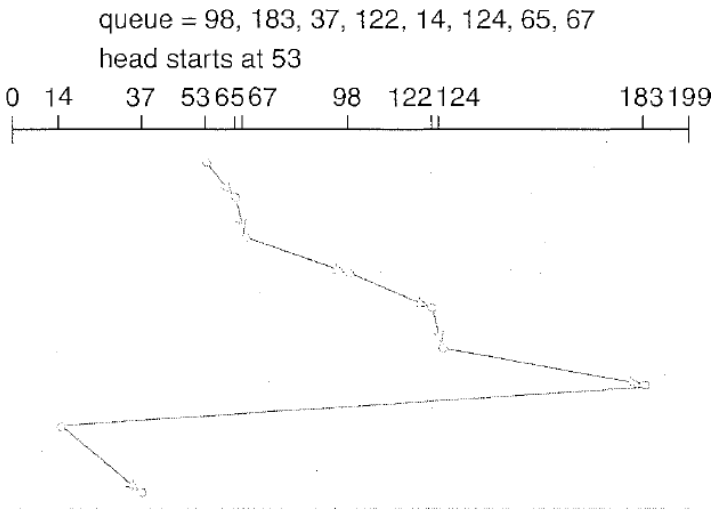


Figure 12.8 C-LOOK disk scheduling.

Selection Of Disk Scheduling Algorithm

- SSTF is a common scheduling algorithm because it increases performance over FCFS.
- SCAN and C-SCAN used in the heavily loaded disk because these avoids starvation problem.
- File-allocation method must be considered while selecting scheduling algorithm.
- The location of directories and index blocks is also plays an important role in selecting scheduling algorithm.

Disk Management

OS is responsible for several disk management activities like

a) Disk formatting

b) Boot block

c) Bad blocks

Disk Formatting

- A new magnetic disc is a blank slate, before a disk can store data, it must be divided into sectors that the disk controllers can read and write. This process is called ”***Low level formatting “or “Physical formatting”.***
- Low–level formatting fills the disk with a special data structure for each sector.It consists of a header, data–area and a trailer.

- To use a disk to hold files,the OS needs to record its own data structures on the disk.Two steps involved in this process are
    - Partition: The disks into one or more groups of cylinders .The OS can treat each partition as it were a separate disk.
    - Logical formatting: In this step, OS stores the initial –file-system data structure onto the disk.

Boot Block

- Initial ***bootstrap*** program is required for a computer to start running. It initializes the system and then starts the OS.
- Boot strap program finds the OS kernel on disk, loads that kernel into memory and jumps to an initial address to begin the OS execution.
- It is stored in ***Read Only Memory*** (ROM).
- Most systems store a ***tiny bootstrap program*** in the boot ROM whose job is to bring in a full bootstrap program from disk.
- Full bootstrap program is stored in "***the boot blocks"*** at a fixed location on the disk
- A disk that has boot partition is called a ***Boot disk*** or ***System disk.***

Bad Blocks

- Because disks have moving parts and small tolerances they are prone to failures.Failure may effect complete disk or it may effect one or two sectors. Most disks even come from factory with "*bad blocks"*
- On simple disks bad blocks are handled manually. If blocks go bad during normal operation. A special program like ***chkdsk*** must be run manually to search for the bad blocks and to lock them. Data that resided on the bad blocks usually are lost.
- In sophisticated disks like **SCSI**, controller maintains a list of bad blocks on the disk . The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as ***" Sector Sparing "*** *or* ***" Forwarding ".***

Swap - Space Management

Swap - Space Use

- Amount of Swap Space needed on a system can vary depending on the amount of physical memory, amount of virtual memory, way in which virtual memory is used etc.
- It is safer to over estimate than to underestimate the amount of Swap Space required, because If system runs out of space it may abort processes. Overestimate writer disk space but it will not cause any harm.

Swap - Space Location

- Swap Space can reside in one of two places. It can be carved out of the normal file system or it can be in a separate disk partition.
- If the Swap Space is a large file within the file system, normal file - system routines can be used to create it, name it and allocate its space.
- Swap Space can be created in a separate raw partition. A separate Swap Sp0ace storage manager is used to allocate and de allocate the blocks from the raw partition.

Swap - Space Management: An Example

- In Solaris 1, when a process executer text - segment pages containing code are brought in form the file system, accused in main memory, and thrown away if select for page out.
- It is more efficient to reread a page from the file system than to write it to Swap Space and then reread it from there.
- Swap Space is only used as a backing store for pages of anonymous memory, which includes memory allocated for the stack, heap and uninitialized data of a process.

SYSTEM PROTECTION

Goals of Protection

Need of protection

- Prevention of mischievous, intentional violation of an access restriction by a user.
- Ensures that each program component in a system uses system resources

according to stated policies.

- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.
- A protection - oriented system provides means to distinguish between authorised and un authorized usage.
- Role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use.

Principles of Protection

- Guiding principle for protection is the ***"Principle Of Least Privilege "***. It dictates that programme, users and even system be given just enough privileges to perform their tasks.
- Principle of least privilege implements programs, system calls in such a way that failure of a component does the minimum damage.
- It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed.

Domain of Protection

- A computer system is a collection of processes and objects (Hardware and software objects)
- The operations that are possible may depend on the object. A process should be allowed to access only those resources for which it has authorization.
- At anytime, a process should be able to access only those resources that it currently requires to complete are task. This is referred as ***"Need-to-Know"*** principle. It limits the amount of damage caused by faulty process.

Domain Structure

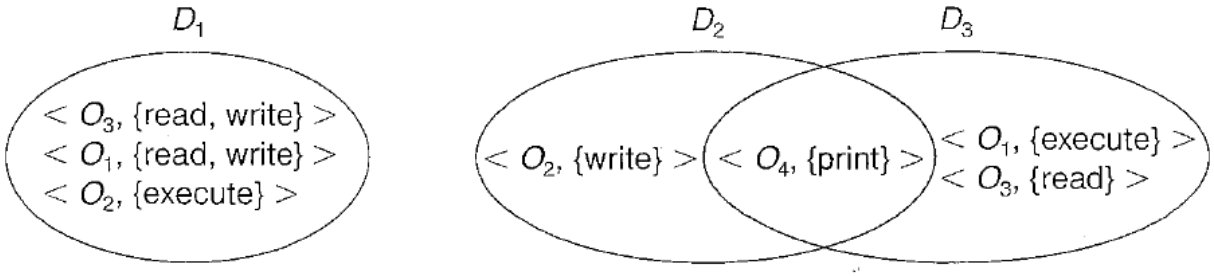


Figure 14.1 System with three protection domains.

- A process operates within a ***Protection Domain*** which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- Ability to execute are operation on an object is all ***" Access Right "***
- A domain is a collection of access rights. It is denoted by ordered pair-
- <object-name, right-set>
- <O_3, {read, write}>
- Domains may share access rights.
- Association between a process and a domain may be static or dynamic.
- Dynamic association supports domain switching i.e.., it enables the process to switch from one domain to another.

A domain can be realized in a variety of ways:

- Each user may be a domain. In this case the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when one user logs out and another user logs in.
- Each process may be a domain. Domain switching occurs when one process and then waits for a response.
- Each producer may be a domain. Domain switching occurs when a procedure call is made.

Domain example in UNIX

- In UNIX Operating system, domain is related with the user. The kernel associates two user ID with a process, independent of the process ID.

They are-

Real user ID & effective user ID or setuid.

- The real user ID identifies the user who is responsible for the running process. The effective user ID is used to assign ownership of newly created files, to check file access permission and to check permission to send signals to processes via the kill system call.
- An owner identification and a domain bit are associated with each file. A setuid program is an executable file that has the setuid bit set in its permission mode field.
- Domain switch is accomplished via file system. Each file has a setuid flag. When setuid flag is set, current user ID set to file owner of program being executed, and reset on exit.

Domain example in multics

- Protection of domains are organized hierarchically into a ring structure.
- Rings are numbered from 0 to ring N-1. Each ring is a single domain.

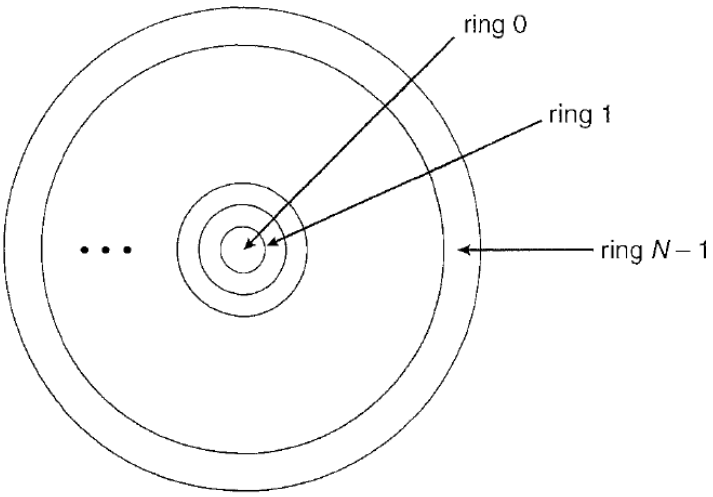


Figure 14.2 MULTICS ring structure.

- Let us consider any two domain rings, i .e, D_i&D_j. If value of j is less than i (j<i), then domain D_i is subset of domain D_j. The process executing in domain D_j has more privileges than does a process executing in domain D_i. Ring 0 has full privileges.
- Ring 1 is a subset of Ring 0. Domain switches is accomplished via access gates.

- Disadvantage of ring structure is that it does not allow us to enforce the need-to-know principle.
- Information in each domain is ordinarily stored in files.
- Ring authority is
    - Inner rings have higher priority
    - Ring 0 corresponds to supervisor mode.
    - Ring 1 to s have decreasing protection and used to implement the operating system.
    - Ring s+1 to N-1 have decreasing protection and used to implement applications.

Access Matrix

- Access matrix is used to implement the protection. Domain is represented by rows of access matrix. Object is represented by columns of access matrix. Each entry in the matrix consists of a set of access rights.
- The entry access (i,j) defines the set of operations that a process executing in Domain D_i can invoke on object O_j.
- Below fig. shows the access matrix.

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

Figure 14.3 Access matrix.

-The access matrix consists of four domains, four objects, three files and one printer. The summary of access matrix is as follows:

- Process in domain D1 can read files F1 and files F3.
- Process in domain D2 can only use printer.

- Process in domain D3 can read file F2 and execute file F3.
- Process in domain D4 can read and write file F1 and file F3.
- Access matrix scheme provides us with the mechanism for specifying a variety of policies. Mechanism consists of implementing the access matrix.
- Access matrix implements policy decisions concerning protection. Policy decisions involve which rights should be include in the $(i, j)^{th}$ entry. We must also decide the domain in which each process executes.
- When a user creates a new object O_j, the column O_j is added to the access matrix. Blank entries indicate no access rights. A process is switched from one domain to another domain by executing switch operation on the object.
- Each entry in the access matrix may be modified individually. Domain switch is only possible if and only if the access right switch ∈ access (i, j).
- Below fig shows the access matrix with domains as objects. Process can change domain as follows-

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

Figure 14.4 Access matrix of Figure 14.3 with domains as objects.

- Process in domain D2 can switch to domain d3 and domain D4.
- Process in domain D4 can switch to domain D1.
- Process in domain D1 can switch to domain D2.
    - Access matrix are inefficient for storage of access rights in computer system because they tend to be large and sparse.
    - Column oriented list is called ***Access Control List(ACL).***Unix uses access control list for file protection. Row oriented list is called a ***capability list***. List kept with the subject

Implementing Access matrix

It is implemented in several ways. Methods for implementing access matrix are-

1. Global table.
2. Access lists for objects.
3. Capability.
4. A lock key mechanism.

1. Global Table

- One of the simplest method for implementation of access matrix. Global table consists of domain, object and right set. The order of syntax is < domain, object, right-set >
- If operation P is executed on an object O_j within domain D_j the global table is searched for a triple-

  <Dj, Oj, Rk> with P∈ Rk
- If the above triple is found, then operation is allowed to continue.
- If suppose triple is not found then an exception error condition occurs.

Limitation of Global table

- Global table is large.
- Global table can not be kept in memory and additional Input/ Output required.

2. Access list for objects

- Matrix is decomposed by columns, yielding access control list. For each object, lists users and their permitted access rights.
- Access list are frequently used in file systems. In systems that employ access lists, a separate list is maintained for each object.
- Only the owner has the authority to modify and define the access list. Deleting the related entry in the access list is possible by owner for granting to the particular subject or domain.

- In UNIX operating system, access lists are reduced to three entries per file, one each for the owner, group and all other user.

Capability

- Each row is associated with its domain.
- A capability list for a domain is a list of objects together with the operations allowed on those objects.
- An object is often represented by its physical name or address, called a Capability.
- To execute operation H, specifying the capability(or pointer) for object OS as a parameter.
- Capabilities are distinguished from other data in two ways-
    - Each object has a tag to denote its type as either a capability or as accessible data.
    - The address space associated with a program can be split into two parts. One part is accessible to the program and contains the programs normal data and instructions.
    - The other part containing the capability list is accessible only by the operating system.

A Lock –Key Mechanism

- The lock key scheme is a compromise between axis list and capability list.
- Each object has a list of unique bit patterns called locks and each domain has a list of unique bit patterns called keys.
- A process executing in a domain can access an object only if the domain has a key that matches one of the locks of the object.
- Users are not allowed to examine or to modify the list of keys directly.

Comparison

- Global table is simple but table can be quite large and cannot take advantage of special groupings of objects or domains.
- Access lists corresponds directly to the needs of users. But determining the set of access rights of a particular domain is difficult.
- Capability lists do not correspond directly to the needs of users. They are useful for localizing information for a given process.
- Lock-Key mechanism is a compromise between access lists and capability lists. The mechanism can be effective and flexible depending on the length of the keys.

Access Control

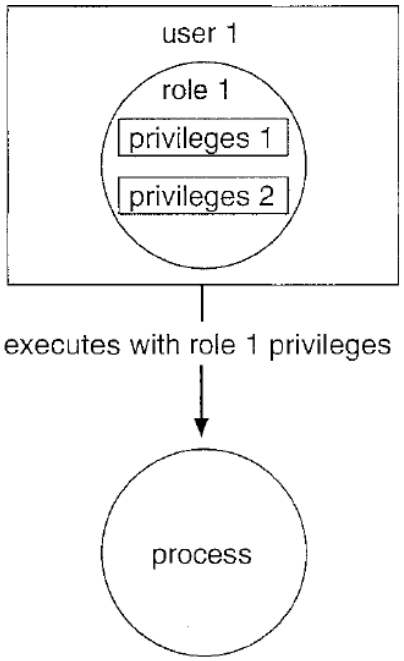


Figure 14.8 Role-based access control in Solaris 10.

- ***Role-based Access control***(RBSC) FACILITY revolves around privileges.
- A privilege is the right to execute a system call or to use an option within that system call. Privileges can be assigned to process or roles.
- Users are assigned roles or can take roles based on passwords to the roles. In this way a user can take a role that enables a privilege allowing the user to run a program to accomplish a specific task as shown in the figure.

Revocation Of Access Rights

- Revocation of access rights to objects in shared environment is possible.
  Following parameters are considered for revocation of access rights
  - Immediate and delayed
  - Selective and general
  - Partial and total
  - Temporary and permanent
- Revocation is easy for access list and complex for capabilities list. The access rights to be revoked and they are deleted from the list.