UNIT 1 - Chapter 1

INTRODUCTION TO SYSTEM SOFTWARE & OPERATING SYSTEM

Unit Structure

1.0 OBJECTIVE

After completion of this unit, you will be able to answer:

- Define System Software.
- Define Operating System.
- Different types of System Software and Operating Systems.
- Role of Virtual Machines in the use of Operating System.
- Define System calls and its types.
- Operating System concepts like Buffering and spooling.

1.1 INTRODUCTION

In our day-to-day life we somehow come across with different types of software that assist us in solving our tasks and help us to increase efficiency in our work. We find software in various electronic gadgetslike a Desktop, Laptop, Cellular Phone and what not. From an operating system software that greets us when we switch on the computer system to the web browser software that is used to explore the electronic content through the internet or the games that we play on our computer to the step count application software on our smartphone,are all instances of software. In this technological world, we even come across various software development trends that help our business to expand, we are surrounded by all this software which help us to make our lives simpler.By definition, a Software is anassembly of data, programs, procedures, instructions, and documentation that perform various predefined tasks on a computer system. They enable users to interact with the computer by processing different type of information.

Any software works only when it has an assistance of some computer hardware technology. Both the entities need each other and neither one of them can be influentially used on its own. The incorporation of the hardware and the software gives control and flexibility to modern-day computing systems. For example, without the help of your web browser software, you will not be able to surf the Internet. Similarly, in the absence of an operating system, no application can run on your computer.

Today there are ample ofsuperior technologies and software accessible to us that define the way we lead our lives and house our frequently changing needs. There are non-ending number of types of software categorised on the basis of technology, functionality, usage etc.

Different types of Software

Software is primarily classified into are two major types, namely SystemSoftware and Application Software.

1. System Software

A system software helps the user and the hardware device to function and interact with each other. Basically, it is a type of software which is used to manage the department of a computer hardware to provide the very basic functionalities that are required by the user. In simple words, we can say that system software works like an intermediatory ora middle layer between the user and the hardware. A system software provides a necessary platform or an environment for the other software to work in. Due to this reason a system software plays an important role in handling the overall computer system. A system software is not just limited to a desktop or a Laptop computer system. It has a broad existence in various digital and electronic devices wherever there is a usage of a computer processor. When a user turns on the computer, it is the system software that gets initialized and gets loaded in the memory of the system. The system software runs in the background and is not used by the end-users. This is the reason why system software is also known as 'low-level software'.

Following are the most common examples of a system software:

- **Operating System (OS):** It is one of the popularly used System Software throughout the digital arena. It is a collection of software that handles resources and provides general services for the other applications that run over them. Although each Operating System is different based on look and feel as well as functionalities, most of them provide a Graphical User Interface through which a user can manage the files and folders and perform other tasks. Every device, whether a desktop, laptop or mobile phone requires an operating system to provide the basic functionality to it. An Operating System essentially determines how a user interacts with the system;therefore, many users prefer to use one specific OS for their device. There are various types of operating system such as singleuser, multiuser, embedded, real-time,distributed,mobile, etc. It is important to consider the hardware specifications before choosing an operating system. Some examples of Operating systems software are Microsoft Windows, Linux, Mac OS, Android, iOS, Ubuntu, Unix, etc.

- **Device Drivers:** It is a type of software that controls the hardwaredevice which is attached to the system. Hardware devices that need a driver to connect to a system include displays, sound cards, printers, mouse, and hard disks. Further, there are two types of device drivers: User Device Driver and Kernel Device Drivers. Some examples of device drivers areVGA Drivers, VGA Drivers, Virtual Device Drivers, BIOS Driver, Display Drivers, Motherboard Drivers, Printer Drivers, ROM Drivers, Sound card Driver, USB Drivers, USB Drivers, etc.

- **Firmware:** It is the permanent software that is embedded into a read-only memory. It is a set of instructions permanently stored on a hardware device. It provides essential information regarding how the device interacts with other hardware. Firmware can be considered as 'semi-permanent' as it remains permanent unless it is updated using a firmware updater. Some examples of firmware are:BIOS, Computer Peripherals, Consumer Applications, Embedded Systems, UEFI, etc.
- **Programming Language Translators:** These are mediator programs on which software programs rely to translate high-level language code to simpler machine-level code. Besides simplifying the code, the translators have the capability to Assign data storage, enlist source code as well as program details, Offer diagnostic reports, Rectify system errors during the runtime. Examples of Programming Language Translators are Interpreter, Compiler and Assemblers.
- **Utility:** This software is designed to aid in analyzing, optimizing, configuring and maintaining a computer system. It supports the computer infrastructure. This software focuses on how an OS functions and then accordingly it decides its trajectory to smoothen the functioning of the system. Software like antiviruses, disk cleanup & management tools, compression tools, defragmenters, etc are all utility tools.

2. Application Software

Also known as end-user programs or productivity programs are software that helps the user in completing various tasks. In contrast to system software, these software are specific in their functionality or tasks used by the end-user. These software areplaced above the system software. Application Software or simply apps can also be referred to as non-essential software as their requirement is highly subjective and their absence does not affect the functioning of the system. For example, such as a text editor, online train or a flight booking application, online banking web-based application, a billing application,high end graphics designing application, accounting software, any type of calculatorapplication or even standalone as well as online games software, the various applications that we use in our cellular phones are also the examples of Application Software. There are certain software frameworks which are solelymade to develop other software applications in the efficient manner.

There are various types of application software:

- **Word Processors:** These applications are widely used to create the documentation. It also helps in storage, formatting, and printing of these documents. Some examples of word processors areAbiword, apple iWork-Pages, Corel WordPerfect, Google Docs, MS Word etc.
- **Database Software:** This software is used to create and manage a database. It is also known as the Database Management System or DBMS. They help with the

organization of data. Some examples of DBMS areClipper, dBase, FileMaker, FoxPro, MS Access, MySQL etc.

- **Multimedia Software:** It is the software that can play, create, or record images, audio or video files. They are used for video editing, animation, graphics, and image editing, some examples of Multimedia Software areAdobe Photoshop, Inkscape, Media Monkey, Picasa, VLC Media Player, Windows Media Player, Windows Movie Maker etc.
- **Education and Reference Software:** These types of software are specifically designed to facilitate learning on a particular subject. There are various kinds of tutorial software that fall under this category. They are also termed as academic software. Some examples areDelta Drawing, GCompris, Jumpstart titles, KidPix, MindPlay, Tux Paint etc.
- **Graphics Software:** As the name suggests, Graphics Software has been devised to work with graphics as it helps the user to edit or make changes in visual data or images. It comprises of picture editors and illustration software. Some examples areAdobe Photoshop, Autodesk Maya, Blender, Carrara, CorelDRAW, GIMP, Modo, PaintShop Pro etc.
- **Web Browsers:** These applications are used to browse the internet. They help the user in locating and retrieving data across the web. Some examples of web browsers areGoogle Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, Opera, Safari, UC Browser etc.

Other than these, all the software that serves a specific purpose fall under the category of Application Software.

1.2 Overview of all System Software

In the previous section of this chapter there is a brief introduction to the System Software and its various types. System software has a very important and necessary existence in the functioning of any computer system. It plays a vital role as a conciliator between application software and computer hardware. This role of a conciliator is played by various types of system software of which some are most important and have a detailed coverage in this chapter. Before moving on to our core concept of operating system software, we would have an insight into some of the system software.

1.2.1Compiler

Compiler is system software which sits in the category of programming language translators. Any application programmer writes the source code,which is also known as a human readable code, in a High Level Languageusing an editor. Acomputer processor only understands the binary language, i.e., of 1’s and 0’s i.e., into the digital form. So, there is requirement of such intermediary software which would help a computer

processor to understand into its own language. Compiler fulfils this objective. Compiler receives anentire source code written in a High Level Language, translates it and produces a target code which is in low level language machine understandable. The examples of High Level languages are C (also used as an assembly level language), C++, FORTRAN, COBOL, Java, C#, Python, Sophia etc.For each high level language, a separate compiler software program is written for performing the conversion task. Like a compiler there is interpreter system software which also translate a high level programming code into a low level code without any separate compilation requirement. As both compiler and interpreter do the same job, but the difference is thata compiler converts an entire source code into machine code before the execution or a program runs, whereas an interpreter translates only one statement of a programming source code into machine understandable codeat a time when the program is in a process of its execution phase. The fig 1.1 shows the conceptual view of the functioning of a Compiler.

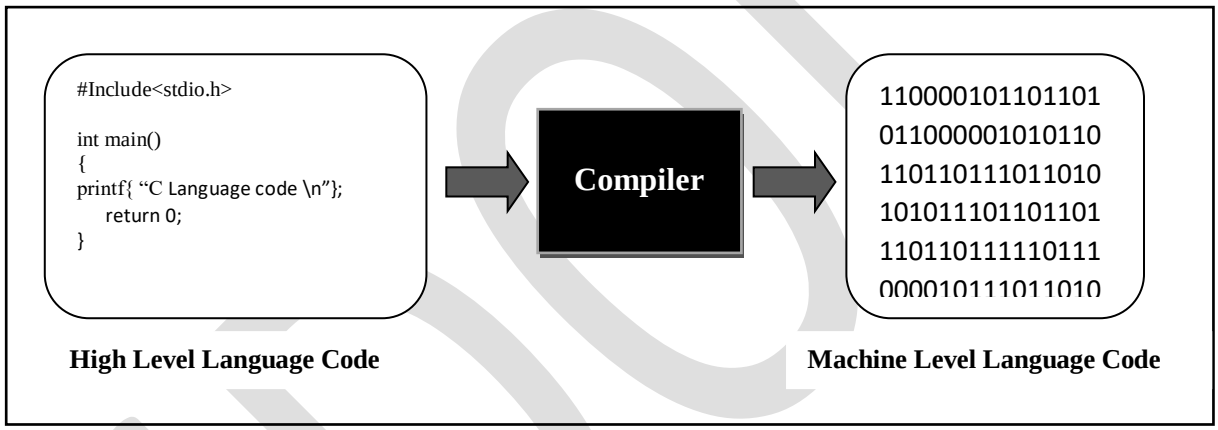


Fig 1.1 Compilation process

1.2.2 Assembler

An Assembler is a program which converts assembly language code into machine level language code. An assembly language is a low level programming language designed for a specific type of a processor. It can be developed from scratch or by the compilation of a high level language like C or C++. Assembly languages differ as per various processor architectures. The instructions written in the assembly language has a very string correspondence with the processor machine code instructions. An assembler accepts a program which consists of the instructions also called as mnemonic processor instructions and converts those into the equivalent numeric code which is finally recognised by a specific type of a processor. It also calculates constant expressions and resolves symbolic names for memory locations and other units. The assemblers are very similar to compilers in terms of producing executable code but in most cases a high level language code is directly converted into a machine level language code.

1.2.3 Linker

A Linker is a program that combines object files which are produced by the compiler or an assembler, and other parts of codes to invent an executable file. In the object file, the linker adds all the supporting libraries and files which are required to execute the file. It also adjusts and controls the memory space that will hold the code from each module. It also helps in uniting the two or more object code programs and forms a link among them. Linkers are of two types. Linkage editor and Dynamic Linker

1.2.4Loader

A Loader is a program that receives the input of object code from linker and loads it to the main memory and makes it ready for final execution by the computer. The main function of a loader is to allocate the memory area for a program. In addition to this it also loads supporting libraries in operating system. Loading mechanism provides three types of approaches:i) Absolute Loading ii) Relocatable Loading iii) Direct runtime loading.

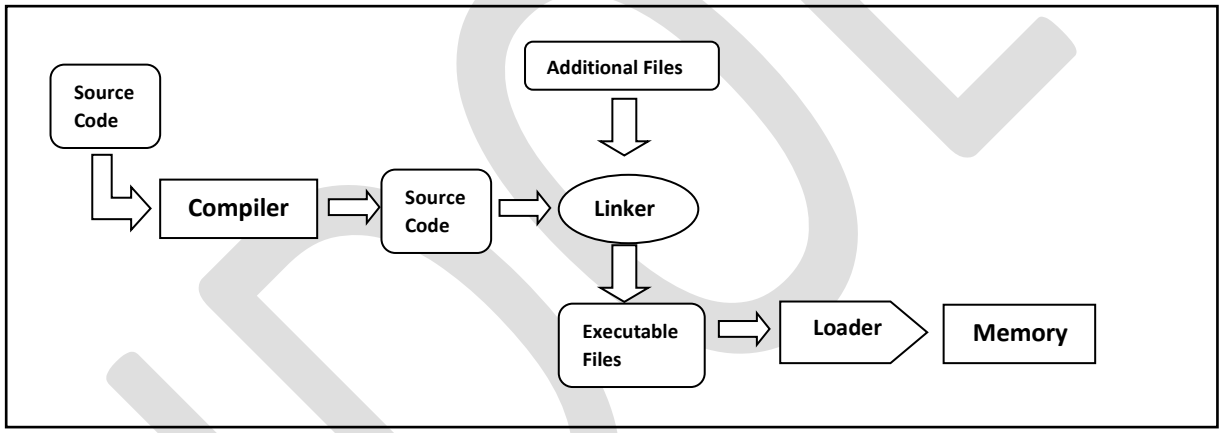


Fig. 1.2 Linker & Loader

1.3Operating Systems

1.3.1 Definition of Operating System

- An operating system is a type of system software that manages the computer machine hardware and acts as a mediator between the computer hardware and the computer user. It also works as a basis for the application software to get executed on the computer machine.
- An operating system is a large and a complex program (basically called as kernel) which runs all the time on computer and controls the execution of application programs and is responsible for the allocation of various services like scheduling, memory management, file system management, Input/output management and resources such as memory, processors, information, and devices.

1.3.2 Role of an Operating System in a computer

Today you can find the operating systems everywhere, from personal computer systems, enterprise servers, and cloud computing servers as well as to the cars, smart phones, home appliances, cars Operating System is one of the necessary software without which the functioning of computer hardware peripherals and the application software is almost impossible. It is works only because of the operating system that the application software which somehow makes use of the various computer devices, like a display monitor to provide with an output to the user or a printer to print some content on the paper. Before moving on to the detailed role of the operating system in this modern computing era, it is important to have a look on the basic organization of a computer system. A basic computer system technically includes the Central Processing Unit (CPU) which is also traditionally termed as heart of a computer, memory, I/O devices, and external storage. The responsibility of an operating system is to allocate these above resources to the application programs.

If we consider a computer system in the view of its user then the basic configuration is of various components like the hardware, the operating system, the application programs, and a user. The hardware of a computer system includes the CPU, memory, the various input and output devices like keyboard, mouse, display monitor, & a printer. The application programs like word processing editors, spreadsheets, compilers, & web browsers- express the ways in which these resources are used to solve a particular problem. Fig 1.3 gives a clear picture of the computer system and the level of operating system at which it is placed.

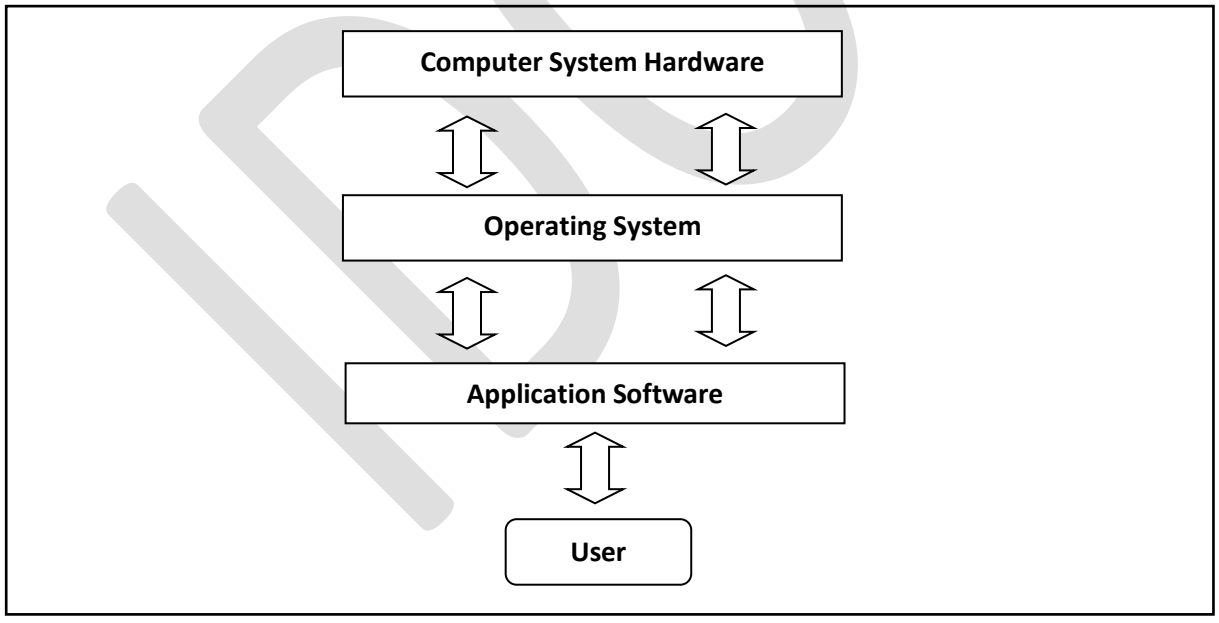


Fig 1.3 User view of various components of a Computer System

From the Fig 1.3 which shows a basic user view of a computer system, we come to know easily that every computer must have an operating system and manages the use of the hardware peripherals among the various user system and application software designed for the users. It also shows that an operating system provides necessary environment within which other programs work productively as well as efficiently. The operating system is a collection of some special programs, when executed on a computer system allows it to work in proper manner. It performs various tasks such as recognizing the input sent from the

keyboard, managing the files and directories on the hard disk, providing output to the display screen and handling various other peripheral devices.

Today, the technical scenario which is projected in the Fig 1.3 is changing rapidly. The computer system is replaced by a mobile device like a cellular phone or a tablet with an operating system specifically designed for the device. These devices are provided with the microprocessor, memory, storage,and touch display etc. which are again controlled by the operating system loaded into the memory. Apart from the configuration these devices are also connected to the networks through the cellular or the wireless technologies. Some computers come with no user view like the embedded system-based devices where the operating system of such devices are designed in such a way that there is very less or negligible user involvement. Modern mobile devices also come with voice recognition display where the user touch is prevented.

1.3.3 Functions provided by the Operating System.

Based on functionality the operating system provides three functions:

1. **Efficiency:** An Operating System lets the efficient use of computer system resources like memory, CPU, and I/O devices such as disks and printers. If the resource which is allocated is not used by an application, then it results into a poor efficiency. That means if the memory of a computer or any Input / Output device is allocated to one application and is remaining idle for longer time then the other application which require those resources, would wait for these resources to get free or may get a denial for its usage. This waiting application also cannot execute, hence the resources allocated to it also remain idle. Apart from this situation, an operating system also consumes resources like memory of its own continuous execution which finally results into a consumption overhead by reducing the access of resources to the applications which are running. To prevent this condition and gain good efficiency, the OS must minimise the waste of resources by the application programs and minimise the overhead.
2. **Convenience:** An Operating Systemhelps in maximum convenience to the user of the computer. This has many facets like ability to execute the application program, use the file system, getting a good service in terms of speedy response to various computational requests, Concurrent programming which is newly introduced, Web oriented features like networking support to setup the Web Servers, Evolutionary facilities so that the OS is comfortable with the adding new feature and technologies, user friendly interfaces, like the todays OS provide which are also known as graphical user interfaces (GUI) which are easy to command through icons on screen to represent the files and programs using either keyboard key strokes or mouse clicks. whereas in early days the operating systems were having a command line interfaces (CLI) which require a user to learn the use of commands and further type them and specify values of its parameters. Todays modern OS require very less user training and experience.

3. **Lesser interference:** An Operating System helps in reducing or preventing the interference in the tasks of the user. A computer user faces different kinds of interference in the computation activities such as execution of an application may get disrupted due to actions of other person or the OS services may also can get disrupted. The OS averts such interference by allocating the resources or by preventing the illegal access to the resources like files by using the act of authorization, whereby the user specifies which collaborators can access what files.

1.3.4 Operations of an Operating System

1.3.4.1 Tasks performed by the Operating Systems.

Once the OS is installed, it relies on a vast library of device drivers to tailor OS services to the specific hardware environment. Thus, every application may make a common call to a storage device, but the OS receives that call and uses the corresponding driver to translate the call into commandsneeded for the underlying hardware on that specific computer. Today, the operating system provides a comprehensive platform that identifies, configures and manages a range of hardware, including processors; memory devices and memory management; chipsets; storage; networking; port communication, such as Video Graphics Array (VGA), High-Definition Multimedia Interface (HDMI) and Universal Serial Bus (USB); and subsystem interfaces, such as Peripheral Component Interconnect Express (PCIe).

Whenever we switch on the computer system, it automatically loads a program storedon a reserved part of an I/O device, typically a disk, and starts executing theprogram. This program follows a software technique known as *bootstrapping* toload the software called the *boot procedure* in memory—the program initiallyloaded in memory loads some other programs in memory, which load otherprograms, and so on until the complete boot procedure is loaded. The bootprocedure makes a list of all hardware resources in the system, and hands overcontrol of the computer system to the OS.A system administrator specifies which persons are registered as users of thesystem. The OS permits only these persons to log in to use its resources andservices. A user authorizes his collaborators to access some programs and data.The OS notes this information and uses it to implement protection. The OS alsoperforms a set of functions to implement its notion of effective utilization. Thesefunctions include scheduling of programs and keeping track of resource statusand resource usage information.An operating System performs some common tasks:

- Construct a list of resources during the booting process.
- Maintain information for security when a new user registers.
- At login time it verifies the identity of a user.
- Initiates execution of programs at user commands.
- Maintain authorization information.
- Perform resource allocation.
- Maintain the current status of resources during its allocation or deallocation.
- Maintain the current status of programs and perform scheduling continuously during the OS operation.

- Provides the facilities to create, modification of programs and data files using an editor.
- Access to the compiler for translating the user program from high level language to machine language.
- Provide a loader program to move the compiled program code to the computer's memory for execution.
- Provide routines that handle the details of I/O programming.

1.3.4.2 Program Management

Modern CPUs have the capability to execute program instructions at a very highrate, so it is possible for an OS to interleave execution of several programs on a CPU and yet provide good user service. The key function in achieving interleaved execution of programs is *scheduling*, which decides which program should be given the CPU at any time. Figure 1.4 shows an abstract view of scheduling. The ***scheduler***, which is an OS routine that performs scheduling, maintains a list of programs waiting to execute on the CPU, and selects one program for execution. In operating systems that provide fair service to all programs, the scheduler also specifies how long the program can be allowed to use the CPU. The OS takes away the CPU from a program after it has executed for the specified period of time and gives it to another program. This action is called ***preemption***. A program that loses the CPU because of preemption is put back into the list of programs waiting to execute on the CPU. The scheduling policy employed by an OS can influence both efficient use of the CPU and user service. If a program is preempted after it has executed for only a short period of time, the overhead of scheduling actions would be high because of frequent preemption. However, each program would suffer only a short delay before it gets an opportunity to use the CPU, which would result in good user service. If preemption is performed after a program has executed for a longer period of time, scheduling overhead would be lesser but programs would suffer longer delays, so user service would be poorer.

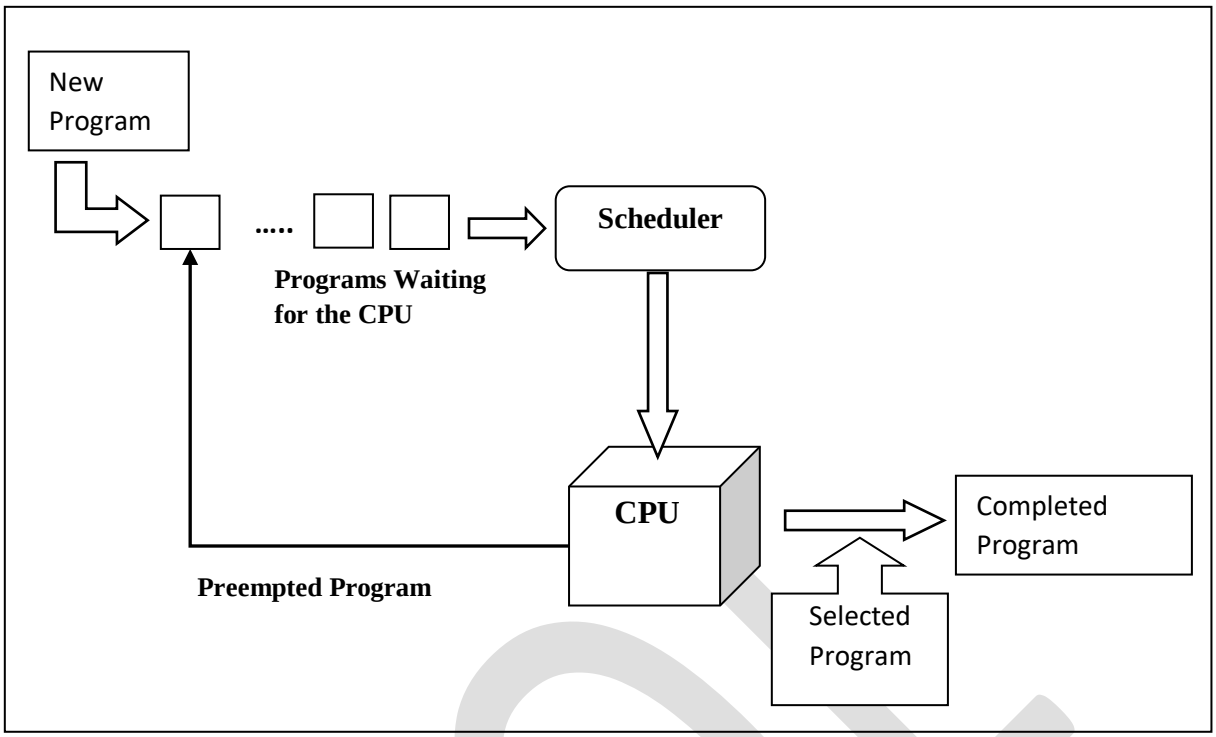


Fig 1.4 OS Scheduling process

1.3.4.3 Resource Management

Resource allocations and deallocations can be performed by using a resourcetable. Each entry in the table contains the name and address of a resource unitand its present status, indicating whether it is free or allocated to some program.Table 1.1 is such a table for management of I/O devices. It is constructed by theboot procedure by sensing the presence of I/O devices in the system and updatedby the operating system to reflect the allocations and deallocations made by it.Since any part of a disk can be accessed directly, it is possible to treat different parts of a disk as independent devices. Thus, the devices disk1 and disk2 in Table 1.3could be two parts of the same disk.Two resource allocation strategies are popular. In the *resource partitioning*approach, the OS decides *a priori* what resources should be allocated to eachuser program, for example, it may decide that a program should be allocated1MB of memory, 1000 disk blocks, and a monitor. It divides the resources in thesystem into many *resource partitions*, or simply *partitions*; each partition includes1 MB of memory, 1000 disk blocks, and a monitor. It allocates one resourcepartition to each user program when its execution is to be initiated. To facilitateresource allocation, the resource table contains entries for resource partitionsrather than for individual resources as in Table 1.3. Resource partitioning issimple to implement, hence it incurs less overhead; however, it lacks flexibility.Resources are wasted if a resource partition contains more resources than what aprogram needs. Also, the OS cannot execute a program if its requirements exceedthe resources available in a resource partition. This is true even if free resourcesexist in another partition.In the *pool-based* approach to resource management, the OS allocatesresources from a common pool of

resources. It consults the resource table whena program makes a request for a resource and allocates the resource if it is free.It incurs the overhead of allocating and deallocating resources when requested.However, it avoids both problems faced by the resource partitioning approach, an allocated resource is not wasted, and a resource requirement can be met if afree resource exists.

Table 1.1**Resource Table for I/O Devices**

Resource name	Class	Address	Allocation status
printer_1	Printer	101	Allocated to P1
printer_2	Printer	102	Free
printer_3	Printer	103	Free
disk_1	Disk	201	Allocated to P1
disk_1	Disk	202	Allocated to P2
cdw_1	CD Writer	301	Free

1.3.4.4 Virtual Resources

A *virtual resource* is a fictitious resource—it is an illusionsupported by an OS through use of a real resource. An OS may use the same realresource to support several virtual resources. This way, it can give the impressionof having a larger number of resources than it does. Each use of a virtualresource results in the use of an appropriate real resource. In that sense, a virtualresource is an abstract view of a resource taken by a program.Use of virtual resources started with the use of virtual devices. To preventmutual interference between programs, it was a good idea to allocate a deviceexclusively for use by one program. However, a computer system did not possessmany real devices, so virtual devices were used. An OS would create a virtualdevice when a user needed an I/O device, e.g., the disks called disk1 and disk2 in Table 1.1 could be two virtual disks based on the real disk, which are allocatedto programs P1 and P2, respectively. Virtual devices are used in contemporaryoperating systems as well. Print server is a common example of a virtual device.When a program wishes to print a file, the print server simply copies the file into theprint queue. The program requesting the print goes on with its operation as if theprinting had been performed. The print server continuously examines the printqueue and prints the files it finds in the queue. Most operating systems providea virtual resource called *virtual memory*, which is an illusion of a memory that islarger in size than the real memory of a computer. Its use enables a programmerto execute a program whose size may exceed the size of real memory.Some operating systems create *virtual machines* (VMs) so that each machinecan be allocated to a user. The advantage of this approach is twofold. Allocationof a virtual machine to each user eliminates mutual interference between users. Italso allows each user to select an OS of his choice to operate his virtual machine.In effect, this arrangement permits users to use different operating systems onthe same computer system simultaneously.

1.3.4.5 Security and Protection

As mentioned in Section 1.3.3, an OS must ensure that no person can illegallyuse programs and resources in the system or interfere with them in any manner.The *security* function counters threats of illegal use or interference that are posedby persons or programs outside the control of an operating system, whereasthe *protection* function counters similar threats posed by its users. Figure 1.5 illustrates how security and protection threats arise in an OS.In a classical stand-alone environment, a computer system functions in completeisolation. In such a system, the security and protection issues can be handledeasily. Recall that an OS maintains information that helps in implementing thesecurity and protection functions (see Table 1.2). The identity of a person wishingto use a computer system is verified through a password when the personlogs in. This action, which is called *authentication*, ensures that no person other than a registered user can use a computer system. Consequently, security threats do not arise in the system if the authentication procedure is fool proof. In thisenvironment, the forms of interference mentioned earlier in Section 1.3.3 are allprotection threats. The OS thwarts disruption of program executions and OSservices with the help of hardware features such as *memory protection*. It thwartsinterference with files by allowing a user to access a file only if he owns it or hasbeen authorized by the file's owner to access it.When a computer system is connected to the Internet, and a user downloadsa program from the Internet, there is a danger that the downloaded programmay interfere with other programs or resources in the system. This is a securitythreat because the interference is caused by some person outside the system,called an *intruder*, who either wrote the downloaded program, or modified it,so that it would interfere with other programs. Such security threats are posedeither through a *Trojan horse*, which is a program that has a known legitimatefunction and a well-disguised malicious function, or a *virus*, which is a pieceof code with a malicious function that attaches itself to other programs in thesystem and spreads to other systems when such programs are copied. Anotherclass of security threat is posed by programs called *worms*, which replicate bythemselves through holes in security setups of operating systems. Worms canreplicate at unimaginably high rates and cause widespread havoc. The Code Red worm of 2001 spread to a quarter of a million computer systems in 9 hours.Operating systems address security threats through a variety of means—byusing sophisticated authentication techniques, by plugging security holes whenthey are discovered, by ensuring that programs cannot be modified while they arecopied over the Internet, and by using Internet *firewalls* to filter out unwantedInternet traffic through a computer system. Users are expected to contribute tosecurity by using passwords that are impossible to guess and by exercising cautionwhile downloading programs from the Internet.

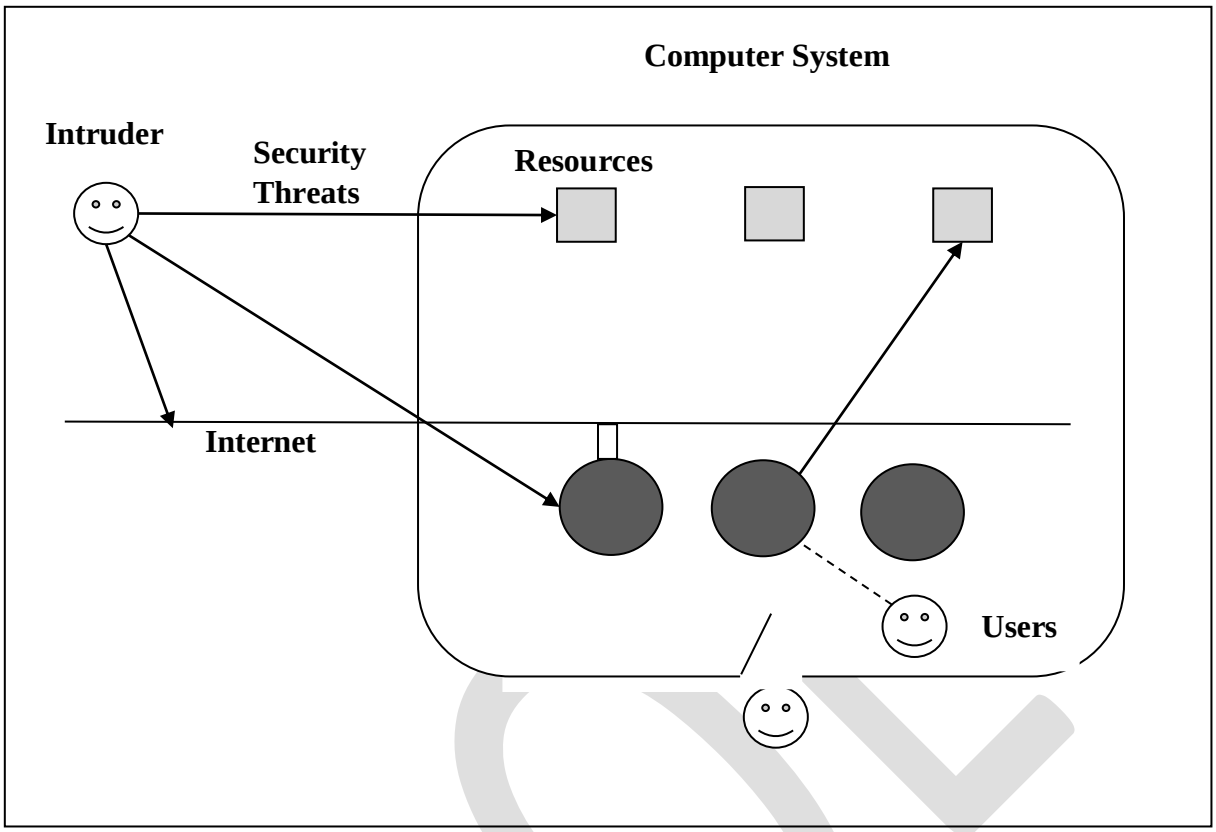


Fig 1.5 Security & Protection Threats

1.3.5 Operating System Services and components

1.3.5.1 OS Services

An operating system provides services to programs and tothe users of those programs. It provided by one environmentfor the execution of programs. The services provided by oneoperating system is difficult than another operating system.Operating system makes the programming task easier.

The common service provided by the operating system islisted below.

1. Program execution
2. I/O operation
3. File system manipulation
4. Communications
5. Error detection

1. **Program execution**: Operating system loads a program intomemory and executes theprogram. The program must beable to end its execution, either normally or abnormally.
2. **I/O Operation:** I/O means any file or any specific I/O device.Program may require any I/O device while running. So,operating system must provide the required I/O.

3. **File system manipulation:** Program needs to read a file orwrite a file. The operating system gives the permission to theprogram for operation on file.
4. **Communication:** Data transfer between two processes isrequired for some time. Both processes are on the onecomputer or on different computer but connected throughcomputer network. Communication may be implemented bytwo methods:
   a. Shared memory.
   b. Message passing.
5. **Error detection:** error may occur in CPU, in I/O devices orin the memory hardware. The operating system constantlyneeds to be aware of possible errors. It should take theappropriate action to ensure correct and consistentcomputing.

Operating system with multiple users provides followingservices.
1. Resource Allocation
2. Accounting
3. Protection

1. Resource Allocation:
   - If there are more than one user or jobs running at the sametime, then resources must beallocated to each of them.Operating system manages different types of resourcesrequirespecial allocation code, i.e., main memory, CPUcycles and file storage.
   - There are some resources which require only generalrequest and release code. For allocating CPU, CPUscheduling algorithms are used for better utilization of CPU.CPU scheduling algorithms are used for better utilization ofCPU. CPU scheduling routines consider the speed of theCPU, number of available registers and other requiredfactors.

2. Accounting:
   - Logs of each user must be kept. It is also necessary to keeprecord of which user how much and what kinds of computerresources. This log is used for accounting purposes.
   - The accounting data may be used for statistics or for thebilling. It also used to improve system efficiency.

3. Protection:
   - Protection involves ensuring that all access to systemresources is controlled.
   - Security starts with each user havingto authenticate to the system, usually bymeans of apassword. External I/O devices must be also protected frominvalid access attempts.
   - In protection, all the access to the resources is controlled. Inmultiprocessenvironment, it is possible that, one process tointerface with the other, or with the operating system, soprotection is required.

1.3.5.2 OS components

Modern operating systems share the goal of supporting thesystem components. The system components are:

1. Process Management
2. Main Memory Management
3. File Management
4. Secondary Storage Management
5. I/O System Management
6. Networking
7. Protection System
8. Command Interpreter System

1.3.6 Types of operating system

1.3.6.1 Batch Processing Systems

In 1960s, the computer systems were not interactive or were physically massive machines which used to run through console. The common input devices were punch card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact with the computer systems. Rather the user prepared the job, which consisted of the program, the data, and some control information about the nature of the job (control cards) and submitted it to the computer operator. The job was usually in the form of punch cards. A computer operator would load the cards into the card reader to set up the execution of a job. The operating system in these early computers was simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in the memory (Fig 1.6). User jobs could not interfere with each other's execution directly because they did not coexist in a computer's memory. However, since the card reader was the only input device available to the users, commands, user programs, and data were all derived from the card reader, so if a program in a job tried to read more data than provided in the job, it would read a few cards of the following job. This action wasted CPU time; batch processing was introduced to prevent this wastage.

A batch is a sequence of user jobs formed for processing by the operating system. A computer operator formed a batch by arranging a few user jobs in a sequence and inserting special marker cards to indicate the start and end of the batch. When the operator gave a command to initiate processing of a batch, the batching kernel set up the processing of the first job of the batch. At the end of the job, it initiated the execution of the next job, and so on, until the end of the batch. Thus, the operator had to intervene only at the start and end of a batch.

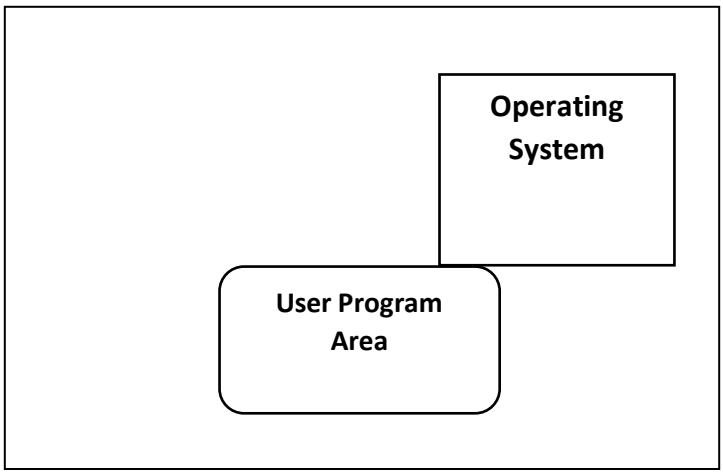


Fig1.6 Memory Layout for a simple batch system

In this execution environment, the CPU is often idle, because the speeds of the mechanical I/O devices are intrinsically slower than are those of electronic devices. Even a slow CPU works in the microsecond range, with thousands of instructions executed per second. A fast card reader, on the other hand, might read 1200 cards per minute (20 cards per second). Thus, the difference in speed between the CPU and its I/O devices may be three orders of magnitude or more. Over time, of course, improvements in the technology and the introduction of disks resulted in faster I/O devices. However, CPU speeds increased to an even greater extent, the problem was not only unresolved, but worsened. The introduction of disk technology allowed the operating system to keep all jobs on a disk, rather than in a serial card reader. With direct access to the several jobs, the operating system could perform job scheduling, to use the resources and perform tasks efficiently.

Advantages of the Batch System

- Move much of the work of the operator to the computer.
- Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of the Batch System

- Turn around time can me more from the user standpoint.
- Difficult to debug the program**.**
- A job could enter an infinite loop.
- A job could corrupt the monitor, thus effecting the pending jobs.
- Due to the lack of protection scheme, one batch job can affect the pending jobs.

1.3.6.2 Multiprocessing System

In a uni-processor system, only one process executes at a time. Multiprocessing is the use of two or more CPUs (processors) within a single Computer

system. The term also refers to the ability of a system to support more than one processor within a single computer system. Now since there are multiple processors available, multiple processes can be executed at a time. These multi processors share the computer bus, sometimes the clock, memory, and peripheral devices also.With the help of multiprocessing, many processes can be executed simultaneously. Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.A computer's capability to process more than one task simultaneously is called *multiprocessing*. A multiprocessing operating system is capable of running many programs simultaneously, and most modern network operating systems (NOSs) support multiprocessing. These operating systems include Windows NT, 2000, XP, and Unix.
But with multiprocessing, each process can be assigned to a different processor for its execution. If it is a dual-core processor, two processes can be executed simultaneously and thus will be two times faster, similarly a quad core processor will be four times as fast as a single processor.

Advantages of multiprocessing operating system

- The main advantage of multiprocessor system is to get more work done in a shorter period of time. These types of systems are used when very high speed is required to process a large volume of data. Multiprocessing systems can save money in comparison to single processor systems because the processors can share peripherals and power supplies.
- It also provides increased reliability in the sense that if one processor fails, the work does not halt, it only slows down. e.g., if we have 10 processors and 1 fails, then the work does not halt, rather the remaining 9 processors can share the work of the 10th processor. Thus, the whole system runs only 10 percent slower, rather than failing altogether.

Disadvantages of Multiprocessor Systems

There are some disadvantages as well to multiprocessor systems. Some of these are:

Increased Expense

Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive. It is much cheaper to buy a simple single processor system than a multiprocessor system.

Complicated Operating System Required

There are multiple processors in a multiprocessor system that share peripherals, memory etc. So, it is much more complicated to schedule processes and impart resources to processesthan

in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

Large Main Memory Required

All the processors in the multiprocessor system share the memory. So, a much larger pool of memory is required as compared to single processor systems.

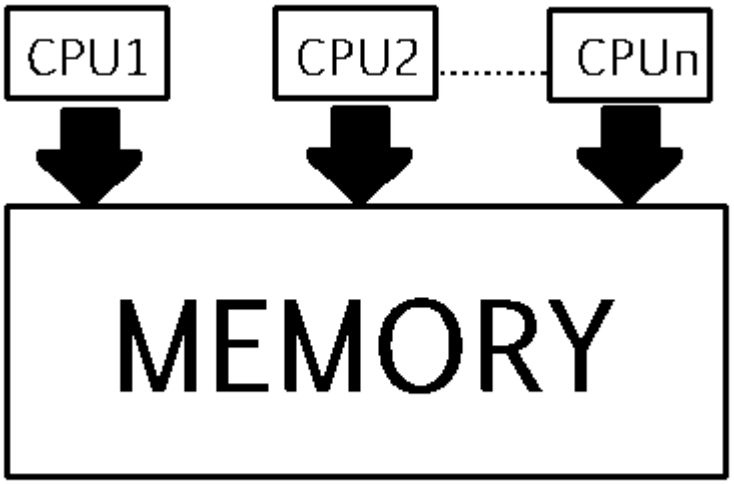


Fig 1.7 An overview of a Multiprocessor System

Multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. It is the ability of the system to leverage multiple processors' computing power.

1.3.6.3 Multitasking System

As the name itself suggests, multi-tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time. In the modern operating systems, we can play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all simultaneously, this is accomplished by means of multi-tasking.

Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

In a time-sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g. time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes

forthat quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.
Thus, the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.
Here also basically a context switch is occurring, but it is occurring so fast that the user is able to interact with each program separately while it is running. This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But only one process/ task is executing at a particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.
In a more general sense, multitasking refers to having multiple programs, processes, tasks, threads running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory).

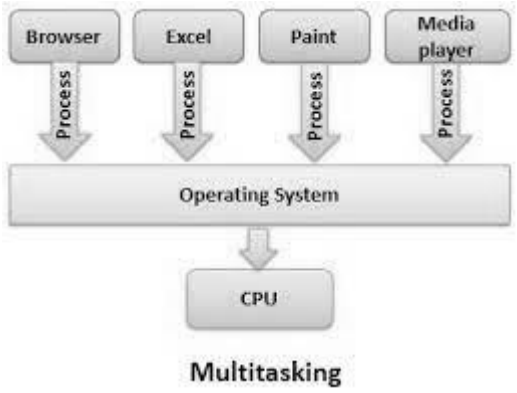


Fig 1.8 Multitasking Systems

1.3.6.4 Timesharing Systems

In an interactive computing environment, a user submits a computationalrequirement—a subrequest—to a process and examines its response on the monitorscreen. A time-sharing operating system is designed to provide a quickresponse to subrequests made by users. It achieves this goal by sharing the CPUtime among processes in such a way that each process to which a subrequest hasbeen made would get a turn on the CPU without much delay.The scheduling technique used by a time-sharing kernel is called *round-robinscheduling with time-slicing*. It works as follows (see Figure 1.9):

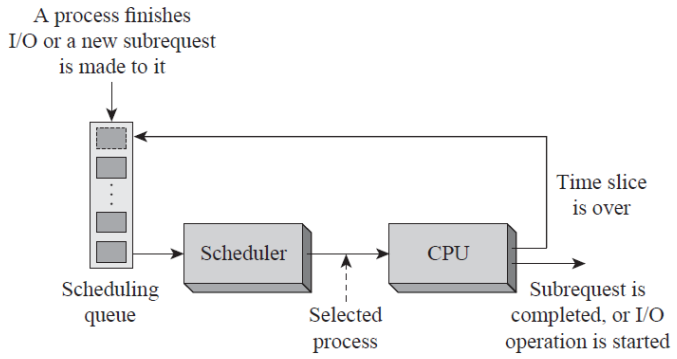


Fig 1.9 Round Robin scheduling with time slicing

The kernel maintains a *scheduling queue* of processes that wish to use the CPU; it always schedulesthe process at the head of the queue. When a scheduled process completes servicingof a subrequest, or starts an I/O operation, the kernel removes it fromthe queue and schedules another process. Such a process would be added at theend of the queue when it receives a new subrequest, or when its I/O operationcompletes. This arrangement ensures that all processes would suffer comparabledelays before getting to use the CPU. However, response times of processes woulddegrade if a process consumes too much CPU time in servicing its subrequest.

1.3.6.5 Distributed Systems

A distributed computer system consists of several individual computer systemsconnected through a network. Each computer system could be a PC, a multiprocessorsystem, or a *cluster*, which is itself a group ofcomputers that work together in an integrated manner. Thus,many resources of a kind, e.g., many memories, CPUs and I/O devices, exist inthe distributed system. A distributed operating system exploits the multiplicityof resources and the presence of a network. However, the possibility of network faults or faults in individual computersystems complicates functioning of the operating system and necessitatesuse of special techniques in its design. Users also need to use special techniquesto access resources over the network. *Resource sharing* has been the traditional motivation for distributed operatingsystems. A user of a PC or workstation can use resources such as printersover a local area network (LAN), and access specialized hardware or softwareresources of a geographically distant computer system over a wide area network(WAN).

A distributed operating system provides *reliability* through redundancy ofcomputer systems, resources, and communication paths—if a computer systemor a resource used in an application fails, the OS can switch the application toanother computer system or resource, and if a path to a resource fails, it can utilizeanother path to the resource. Reliability can be used to offer high *availability* ofresources and services,which is defined as the fraction of time a resource or serviceis operable. High availability of a data resource, e.g., a file, can be provided bykeeping copies of the file in various parts of the system.*Computation speedup* implies a reduction in the duration of an application,i.e., in its running time. It is achieved by dispersing processes of an applicationto different computers in the distributed system, so that they can execute at thesame time and finish earlier than if they were to be executed in a conventional OS.Users of a distributed operating system have user ids and passwords thatare valid throughout the system. This feature greatly facilitates *communication*between users in two ways. First, communication through user ids automaticallyinvokes the security mechanisms of the OS and thus ensures authenticity of communication.Second, users can be mobile within the distributed system and stillbe able to communicate with other users through the system.

Benefits of Distributed Operating Systems

1) Resources can be shared and utilised across boundaries of individualcomputer systems.

2) Its reliability confirms that the OS continues to function even when computersystems or resources in it fail.
3) Processes of an application can be executed in differentcomputer systems to speed up its completion.
4) Users can communicate among themselves irrespective oftheir locations in the system.

A distributed system is more than a mere collection of computers connected toa networkfunctioning of individual computers must be integrated.It is achieved through participation ofall computers in the control functions of the operating system. Accordingly, wedefine a distributed system as follows:

Asystem consisting of two or more nodes,where each node is a computer system with its own clock and memory, somenetworking hardware, and a capability of performing some of the controlfunctions of an OS.

1.3.6.6 Real Time Systems

In a class of applications called *real-time applications*, users need the computer toperform some actions in a timely manner to control the activities in an externalsystem, or to participate in them. The timeliness of actions is determined bythe time constraints of the external system. Accordingly, we define a real-timeapplication as follows:
Real-Time Application Aprogramthat responds to activities inan external system within a maximum time determined by the external system.If the application takes too long to respond to an activity, a failure canoccur in the external system. We use the term *response requirement* of a systemto indicate the largest value of response time for which the system can functionperfectly; a timely response is one whose response time is not larger than theresponse requirement of the system.Consider a system that logs data received from a satellite remote sensor.The satellite sends digitized samples to the earth station at the rate of 500 samplesper second. The application process is required to simply store these samples ina file. Since a new sample arrives every two thousandth of a second, i.e., every2 ms, the computer must respond to every "store the sample" request in less than2 ms, or the arrival of a new sample would wipe out the previous sample in thecomputer's memory. This system is a real-time application because a samplemustbe stored in less than 2 ms to prevent a failure. Its response requirement is 1.99 ms.The *deadline* of an action in a real-time application is the time by which the actionshould be performed. In the current example, if a new sample is received fromthe satellite at time t, the deadline for storing it on disk is t + 1.99 ms.Examples of real-time applications can be found in missile guidance, commandand control applications like process control and air traffic control, datasampling and data acquisition systems like display systems in automobiles,multimediasystems, and applications like reservation and banking systems that employlarge databases. The response requirements of these systems

vary from a fewmicroseconds or milliseconds for guidance and control systems to a few secondsfor reservation and banking systems.

Features of a Real-Time Operating System

The first threefeatures help an application in meeting the response requirement of a systemas follows: A real-time application can be coded such that the OS can executeits parts concurrently, i.e., as separate processes. When these parts are assignedpriorities and priority-based scheduling is used, we have a situation analogousto multiprogramming *within* the applicationif one part of the application initiatesan I/O operation, the OS would schedule another part of the application.Thus, CPU and I/O activities of the application can be overlapped with oneanother, which helps in reducing the duration of an application, i.e., its runningtime. *Deadline-aware scheduling* is a technique used in the kernel that schedulesprocesses in such a manner that they may meet their deadlines.Ability to specify *domain-specific events* and event handling actions enablesa real-time application to respond to special conditions in the external systempromptly. *Predictability* of policies and overhead of the OS enables an applicationdeveloper to calculate the worst-case running time of the application anddecide whether the response requirement of the external system can be met. Thepredictability requirement forces a hard real-time OS to shun features such as *virtualmemory* whose performance cannot be predicted precisely. The OS would also avoid shared use of resources by processes, because it can leadto delays that are hard to predict and unbounded, i.e., arbitrarily large.A real-time OS employs two techniques to ensure continuity of operation when faults occur*fault tolerance* and *graceful degradation*. A fault-tolerantcomputer system uses redundancy of resources to ensure that the system willkeep functioning even if a fault occurs, e.g., it may have two disks even thoughthe application needs only one disk. Graceful degradation is the abilityof a system to fall back to a reduced level of service when a fault occurs andto revert to normal operations when the fault is rectified. The programmer can assign high priorities to crucial functions so that they would be performed in atimely manner even when the system operates in a degraded mode.

1.4Virtual Machines

Different classes of users need different kinds of user service. Hence running a singleOS on a computer system can disappoint many users. Operating the computerunder different OSs during different periods is not a satisfactory solution becauseit would make accessible services offered under only one of the operating systemsat any time. This problem is solved by using a *virtual machine operating system*(VM OS) to control the computer system. The VM OS creates several *virtualmachines*. Each virtual machine is allocated to one user, who can use any OS of hisown choice on the virtual machine and run his programs under this OS.

This wayusers of the computer system can use different operating systems at the same time.We call each of these operating systems a *guest OS* and call the virtual machineOS the *host OS*. The computer used by the VM OS is called the *host machine*. Let us consider avirtual machine that has the same architecture as the host machine, i.e., it has a virtualCPUcapableof executing the same instructions, and similar memory and I/Odevices. It may, however, differ from the host machine in terms of some elementsof its configuration like memory size and I/O devices. Because of the identicalarchitectures of the virtual and host machines, no semantic gap exists betweenthem, so operation of a virtual machine does not introduce any performance losssoftware intervention is also not needed to run a guest OS on a virtual machine.The VM OS achieves concurrent operation of guest operating systemsthrough an action that resembles process scheduling—it selects a virtual machineand arranges to let the guest OS running on it execute its instructions on the CPU.The guest OS in operation enjoys complete control over the host machine'senvironment, including interrupt servicing. The absence of a software layerbetween the host machine and guest OS ensures efficient use of the host machine.A guest OS remains in control of the host machine until the VM OS decidesto switch to another virtual machine, which typically happens in response toan interrupt. The VM OS can employ the timer to implement time-slicing andround-robin scheduling of guest OSs.A somewhat complex arrangement is needed to handle interrupts that arisewhen a guest OS is in operation. Some of the interrupts would arise in its owndomain, e.g., an I/O interrupt from a device included in its own virtual machine,while others would arise in the domains of other guest OSs. The VM OS canarrange to get control when an interrupt occurs, find the guest OS whose domainthe interrupt belongs to, and "schedule" that guest OS to handle it. However, thisarrangement incurs high overhead because of two context switch operations—thefirst context switch passes control to the VM OS, and the second passes controlto the correct guest OS. Hence the VM OS may use an arrangement in which theguest OS in operation would be invoked directly by interrupts arising in its owndomain. It is implemented as follows:

While passing control to a guest operatingsystem, theVMOS replaces its own interrupt vectorsby thosedefined in the guest OS. This action ensures that an interrupt would switch theCPU to an interrupt servicing routine of the guest OS. If the guest OS finds thatthe interrupt did not occur in its own domain, it passes control to the VM OSby making a special system call "invoke VM OS." The VM OS now arranges topass the interrupt to the appropriate guest OS. When a large number of virtualmachines exists, interrupt processing can cause excessive shuffling between virtualmachines, hence theVMOS may not immediately activate the guest OS in whosedomain an interrupt occurred, and it may simply note occurrence of interrupts thatoccurred in the domain of a guest OS and provide this information to the guestOS the next time it is "scheduled."Distinction between kernel and user modes of the CPU causes some difficultiesin the use of a VM OS. The VM OS must protect itself from guest OSs,so it must run guest OSs with the CPU in the user mode. However, this wayboth a guest OS and user processes under it run in the user mode, which makesthe guest OS vulnerable to corruption by a user process. The Intel 80x86 familyof computers has a feature that provides a way out of this difficulty. The 80x86computers support four execution modes of the CPU. Hence the host OS canrun with the CPU in the kernel mode, a guest OS can execute processes runningunder it with the CPU in the user

mode but can itself run with the CPU in oneof the intermediate modes.*Virtualization* is the process of mapping the interfaces and resources of avirtual machine into the interfaces and resources of the host machine. Full virtualizationwould imply that the host machine and a virtual machine have identicalcapabilities, hence an OS can operate identically while running on a bare machineand on a virtual machine supported by a VM OS. However, full virtualizationmay weaken security.

Virtual machines are employed for diverse purposes:

• To use an existing server for a new application that requires use of a differentoperating system. This is called *workload consolidation*; it reduces the hardwareand operational cost of computing by reducing the number of serversneeded in an organization.
• To provide security and reliability for applications that use the same hostand the same OS. This benefit arises from the fact that virtual machines ofdifferent applications cannot access each other's resources.
• To test a modified OS (or a new version of application code) on a serverconcurrently with production runs of that OS.
• To provide disaster management capabilities by transferring a virtualmachine from a server that has to shut down because of an emergency toanother server available on the network.

1.5System Calls

System calls provide the means for a user program to ask the operatingsystem to perform tasks reserved for the operating system on the user program'sbehalf. A system call is invoked in a variety of ways, depending onthe functionality provided by the underlying processor. In all forms, it is themethod used by a process to request action by the operating system. Asystemcall usually takes the form of a trap to a specific location in the interrupt vector.This trap can be executed by a generic trap instruction, although some systemshave a specific system call instruction to invoke a system call.When a system call is executed, it is typically treated by the hardware asa software interrupt. Control passes through the interrupt vector to a serviceroutine in the operating system, and the mode bit is set to kernel mode. Thesystem-call service routine is a part of the operating system. The kernel examinesthe interrupting instruction to determine what system call has occurred;a parameter indicates what type of service the user program is requesting.Additional information needed for the request may be passed in registers, onthe stack, or in memory (with pointers to the memory locations passed in registers).The kernel verifies that the parameters are correct and legal, executesthe request, and returns control to the instruction following the system call.Wedescribe system calls more fully in Section 2.3.Once hardware protection is in place, it detects errors that violate modes.These errors are normally handled by the operating system. If a user programfails in some way—such as by trying either to execute an illegalinstruction or to access memory that is not in the user's address space—thenthe hardware traps to the operating system. The trap transfers control throughthe interrupt vector to the operating system, just as an interrupt does. Whena program error occurs, the operating system must terminate the programabnormally.

This situation is handled by the same code as a user-requestedabnormal termination. An appropriate error message is given, and the memoryof the programmay be dumped. Thememory dump is usuallywritten to a fileso that the user or programmer can examine it and perhaps correct it and restartthe program.A program needs to use computer resources like I/O devices during its execution.However, resources are shared among user programs, so it is necessary to preventmutual interference in their use. To facilitate it, the instructions that allocate oraccess critical resources are made privileged instructions in a computer's architecture.This way, these instructions cannot be executed unless the CPU is in thekernel mode, so user programs cannot access resources directly; they must makerequests to the kernel, and the kernel must access resources on their behalf. Thekernel provides a set of services for this purpose.In a programmer view, a program uses a computer's resources through statementsof a programming language. The compiler of a programming languageimplements the programmer view as follows: While compiling a program, a system call is implemented through the interrupt action describedearlier, hence we define it as follows:
A request that a program makes to the kernelthrough a software interrupt.

For example, if we need to write a program code to read data from one file, copy that data into another file. The first information that the program requires is the name of the two files, the input and output files.

In an interactive system, this type of program execution requires some system calls by OS.

- First call is to write a prompting message on the screen.
- Second, to read from the keyboard, the characters which define the two files.

1.5.1 Types of System Calls

Here are the five types of system calls used in OS:

- Process Control:
  This system call performs the task of the process creation, process termination etc.

- File Management:
  File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
- Device Management:
  Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
- Information Maintenance:
  It handles information and its transfer between the OS and the user program.
- Communications:
  These types of system calls are specially used for inter-process communications.

1.6 Buffering

A buffer is a memory area that stores data while they aretransferred between two devices or between a device arid anapplication. Buffering is done for three reasons.

1) One reason is to cope with a speed mismatch between theproducer and consumer of a data stream.
2) Asecond use of buffering is to adapt between devices thathave different data transfer sizes.
3) A third use of buffering is to support copy semantics forapplication I/O.

1.7 Spooling

Spool is an acronym for simultaneous peripheral operations online.Spooling refers to putting jobs in a buffer, a special area inmemory or on a disk where a device can access them whenit is ready.Spooling is useful because device access data that differentrates. The buffer provides a waiting station where data canrest while the slower device catches up. Computer can perform I/O in parallel with computation, itbecomes possible to have the computer read a deck ofcards to a tape, drum, or disk and to write out to a tapeprinter while it was computing. This process is calledspoolingthe most common spooling application is print spooling. Inprint spooling, documents are loaded into a buffer and thenthe printer pulls them off the buffer at its own rate.Spooling is also used for processing data at remote sites.The CPU sends the data via communications path to aremote printer. Spooling overlaps the I/O of one job with thecomputation of other jobs.One difficulty with simple batch systems is that the computerstill needs to read the decks of cards before it can begin toexecute the job. This means that the CPU is idle duringthese relatively slow operations.Spooling batch systems were the first and are the simplest ofthe multiprogramming systems.

Advantage of Spooling

1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operationfor one job with processor. operations for another job.

1.8 List of References

1. Operating System Concepts (9th Ed) by Silberschatz and Galvin, Wiley, 2000.
2. Operating Systems D.M.Dhamrdhere Tata McGraw Hill

1.9 Unit End Exercises

1. Define System Software. Explain Compiler, Assembler, Linker and Loader in detail.
2. Define Operating System. What are the various services provided by an operating system?
3. Explain Batch Operating System in detail.

4. Explain multiprocessing Operating System in detail.
5. Explain multitasking Operating System in detail.
6. Explain the time-sharing OS in detail.
7. Explain the Distributed OS in detail.
8. Explain Real Time OS in detail.
9. Define System Calls. What are different types of System calls?
10. What are Virtual machines?
11. Explain Buffering and Spooling in detail.

UNIT 2: Chapter 2

PROCESS AND THREAD MANAGEMENT

Unit Structure

2.0 OBJECTIVES

After going through this unit, you will be able to:

- define process, thread and multithreading
- state context switching and various process states
- Classify differentCPU Scheduling algorithms and multithreading models.

2.1CONCEPT OF PROCESS AND THREADS

2.1.1 Concept of Process

A process is a program in execution. A process is more than the program code. Process can be known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.

A process generally also includes:

- The process stack, which contains temporary data (such as function parameters, return addresses, and local variables)
- A data section, which contains global variables.
- A process may also include a heap, which is memory that is dynamically allocated during process run time.

A program by itself is not a process; a program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file), whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.

Figure 3.1 shows the structure of process in memory.

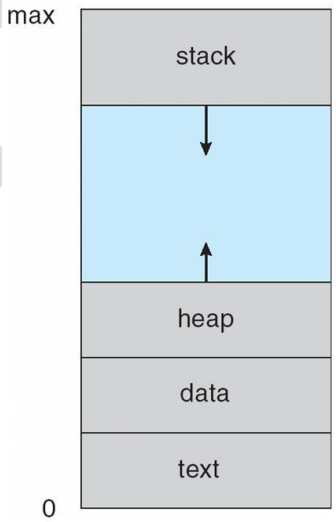


Figure 2.1 Process in memory

2.1.2 Process States

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. The process state transition diagram indicates the current activity of a process. The process may enter in one of the following states:

1. New

2. Ready

3. Running

4. Waiting

5. Terminated

The above mentioned states can be descried as follows:

1. *New:* A process that just been created.
2. *Ready:* Ready processes are waiting for the processor to get allocated operating system.
3. *Running:* The process that is currently being executed. A running process possesses all the resources needed for its execution, including the processor.
4. *Waiting:* A process that cannot execute until some event occurs such as the completion of an I/O operation. The running process may get suspended by invoking an I/O module.
5. *Terminated:* A process that has been released from the pool of executable processes by the operating system.

Figure 2.2 shows the state transition diagram of a process.

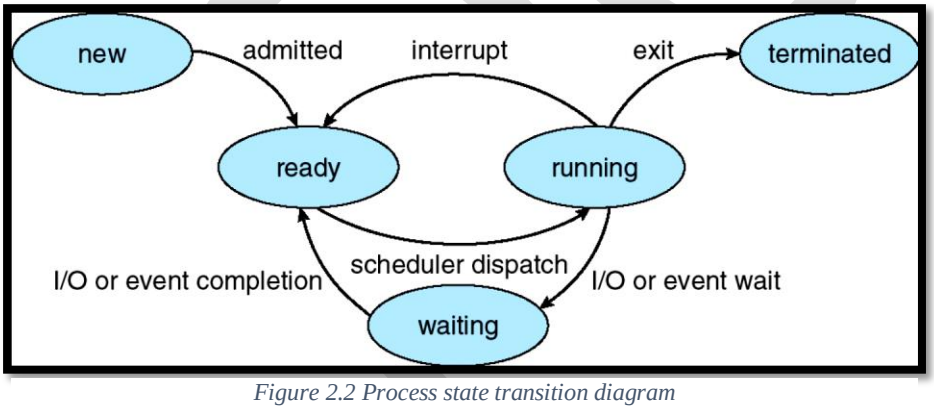


Figure 2.2 Process state transition diagram

2.1.3 Process Termination

The process gets terminated due to one of the following conditions:

1. Normal exit (voluntary).

2. Operating System Intervention….terminated due to deadlock.
3. Fatal error (involuntary)… Protection, Arithmetic, I/O failure, Misuse of access rights, writing to readonly file.
4. Killed by another process (involuntary).
5. Memory not available.

2.1.4 Types of processes

*User processes:*execute primarily applications and utilities also execute operating system programs

Kernel processes: execute only operating system programs.

Independent Processes: the processes which execute without any interference.

Cooperating process: affected by other processes executing in system, requires IPC(shared memory or message passing)

*Orphan Process:*Shell would terminate all the child processes with the SIGHUP process signal, rather than letting them continue to run as orphans.

Zombie Process: is a process that has completed execution (via the exit system call) but still has an entry in the process table: it is a process in the "Terminated state".

2.1.5 Process Control Block

Each process is represented in the operating system by a Process Control Block (PCB) also called as Task Control Block.

The operating system groups all information that needs about a particular process into a data structure called a PCB or Process Descriptor.

When a process is created, operating system creates a corresponding PCB and released whenever the process terminates.

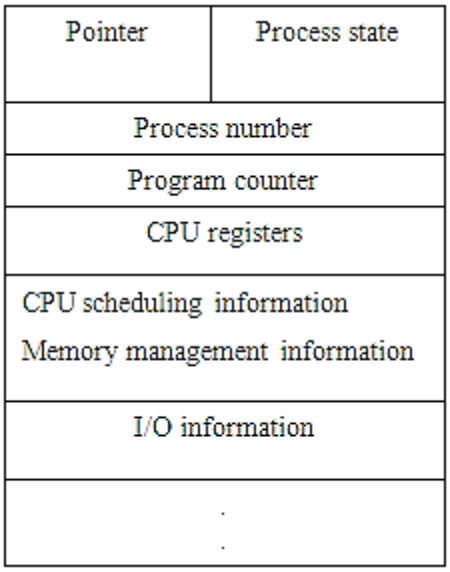


Figure 2.3 Process Control Block

The process control block describes the following sections:

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

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

CPU registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general purpose registers, plus any condition code information.

CPU scheduling information: This information includes a process priority, pointers to scheduling queues and any other scheduling parameters.

Memory management information: This information may include such information as the value of the base and limit registers, the page tables or the segment tables, depending on the memory system used by the OS.

Accounting information: This information includes the amount of CPU and real-time used, time limits, account numbers, job or process numbers and so on.

I/O status information: This information includes the list of I/O devices allocated to the process, a list of open files and so on.

2.2 PROCESS MANAGEMENT

2.2.1 Process Scheduling

The **process scheduler** selects an available process (possibly from a set of several available processes) for program execution on the CPU. The objective of multiprogramming is to have some process running at all times. To maximize the CPU utilization scheduling is done among various processes.

As processes enter the system, they are put into a **job queue.** Operating system manages following queues.

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute.
    - A ready queue header contains pointer to first & last PCBs in the linked list.
    - Each PCB has pointer, points to next process in queue.
- **Device queues** – set of processes waiting for an I/O device

Processes migrate among the various queues.

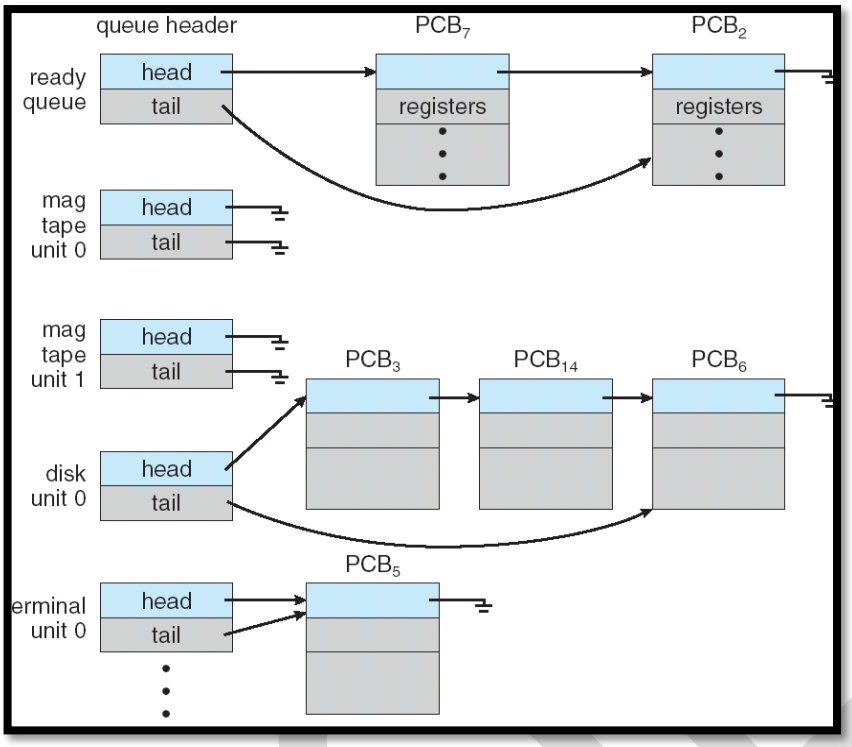


Figure 2.4 Ready queue and various I/O device queue

Figure 2.4 shows two types of queues are present: the ready queue and a set of device queues. Each rectangular box represents a queue. The circles represent the resources that serve the queues, and the arrows indicate the flowof processes in the system.

A new process is initially put in the ready queue. It waits there till it is selected for execution, oris dispatched. Once the process is allocated the CPU and is executing, one of several events could occur:

- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new sub process and wait for the sub process's termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in theready queue.

A queueing presentation of process scheduling is shown in figure 2.5. Each rectangular box represents a queue. The figure shows two types of queues, ready queue and a set of device queues. The circles represent the resources. A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched. Once the process is allocated the CPU and is executing, one of several events could occur:

- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new subprocess and wait for the subprocess's termination.
- The process could be removed forcibly from the CPU as a result of an interrupt, and be put back in the ready queue.

2.2.2 Schedulers

A process migrates between the various scheduling queues throughout its life-time purposes. The OS selects processes for scheduling from the queues.

The selection of a process is carried out by the appropriate scheduler.

Types of schedulers:

There are 3 types of schedulers:

1. **Long term scheduler:** Long term scheduler selects process from the disk & loads them into memory for execution. It controls the degree of multi-programming. It executes less frequently than other schedulers. Long term scheduler is needed to be invoked only when a process leaves the system. Most processes in the CPU are either I/O bound or CPU bound. An I/O bound process spends more time in doing I/O operation. A CPU bound process spends more time in doing computations than I/O operations. It is important that the long term scheduler should select a good mix of I/O bound & CPU bound processes.
2. **Short term scheduler:** The short term scheduler selects among the process that are ready to execute & allocates the CPU to one of them. The short-term scheduler must select a new process for the CPU quite frequently. Due to the short duration of time between executions, it must be very fast.
3. **Medium - term scheduler:** It is intermediate level scheduler. of scheduling known as medium - term scheduler. Sometimes it is advantageous to remove processes from memory & thus reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory & its execution can be continued from where it had left off. This is called as swapping. The process is swapped out & swapped in later by medium term scheduler. Swapping is necessary to improve the process miss or due to some change in memory requirements, the available memory limit is exceeded which requires some memory to be freed up. Figure 2.5 shows working of medium term scheduler.

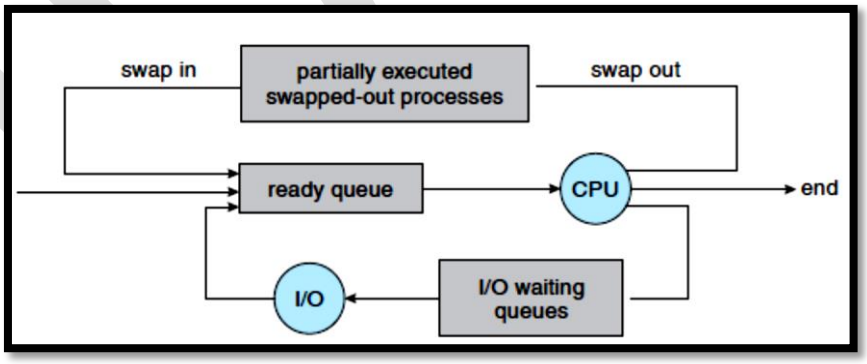


Figure 2.5 Medium term scheduler

Long term	Short Term	Medium Term
Job scheduler	CPU Scheduler	swapping
Speed less than short	Very fast	Between both
Controls degree of multiprogramming	Less control over degree of multiprogramming	Reduces degree of multiprogramming
Absent in timesharing system	Minimal in timesharing system	used

Selects processes from pool and load them into memory	Selects from among processes that are ready to execute	Process can be reintroduced into memory and its execution can be continued.
Process state(New, Exit)	Process state(Ready , Running,Blocked)	Ready-suspended, blocked-suspended
Select good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	--

Table 2.1 Difference between schedulers

2.2.3Context Switch

The machine register contain the hardware context of currently running process. When a context switch occurs, these registers are saved in PCB of current process.

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Context switch time dependent on hardware support. For example, if register contents do not have to be saved because of the availability of large number of registers, then context switch time will be low.

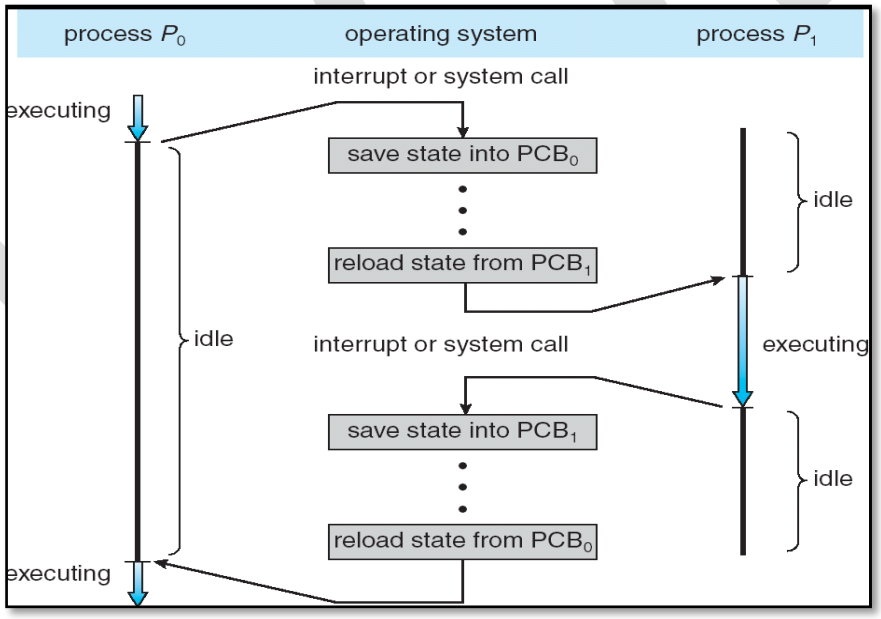


Figure 2.6 Context switch from process to process

2.2.4Interaction between processes and OS

Processes executing concurrently in the operating system may be either independent processes or cooperating processes. If the process is independent it cannot affect other processes. The process that shares data with other processes is a cooperating process. Cooperating process can affect other processes. Cooperating processes give following benefits:

- Information sharing
- Computation speedup

- Modularity
- Convenience

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information. There are two models of interprocess communication:

1. Shared memory
2. Message passing.

In the shared-memory model, processes can then exchange information by reading and writing data to the shared region. In the message passing model, communication takes place by means of messages exchanged. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided.

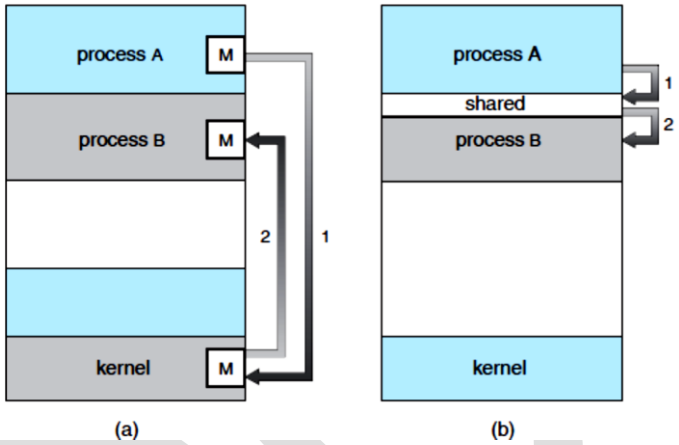


Figure 2.7 Communications models. (a) Message passing. (b) Shared memory.

2.3MULTITHREADING

2.3.1 Introduction to Threads

- Multithreading refers to the ability of an OS to support multiple, concurrent paths of execution within a single process.
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional (or heavyweight) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

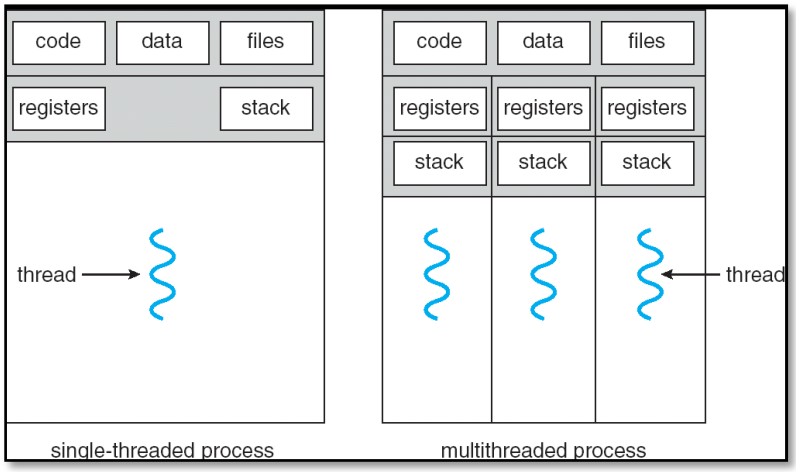


Figure 2.8 Multithreading concepts

2.3.2 Benefits of Multithreading

- *Responsiveness:* Multithreading an interactive application increases responsiveness to the user.
- *Resource Sharing:* Threads share memory and resources of the processes to which they belong.
- *Economy:* Process creation is expensive.
- *Utilization of Multiprocessor Architectures:* Benefits of multithreading increase in multiprocessor systems because different threads can be scheduled to different processors

2.3.3 Types of threads

1. User level threads

- Threads are implemented at user level by Thread Library: creates, scheduling and management of threads without kernel involvement.
- Hence fast to create & manage.
- Status of information table is maintained within Thread Library hence better scalability.

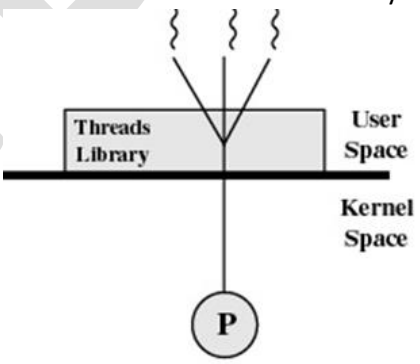


Figure 2.9 User level threads

2. Kernel level threads

- Kernel maintains process table and keeps track of all processes.
- Kernel threads are slow & insufficient because it requires a full TCB(Thread Control Block) for each thread to manage and schedule.

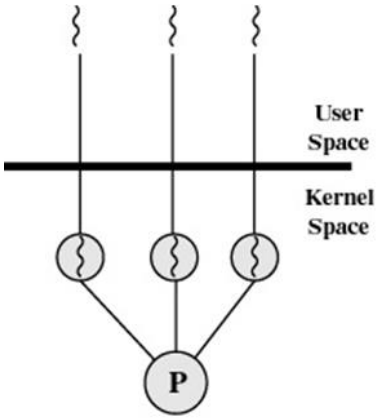


Figure 2.10 Kernel level threads

Table 2.2 shows the difference between user level threads and kernel level threads.

S.N.	User Level Threads	Kernel Level Thread
1	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User level thread is generic and can run on any operating system.	Kernel level thread is specific to the operating system.
4	Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Table 2.2 Difference between user level threads and kernel level threads

2.3.4 Multithreading Models

Some operating system provide a combined user levelthread and Kernel level thread facility. Solaris is a goodexample of this combined approach. In a combined system,multiple threads within the same application can run inparallel on multiple processors and a blocking system callneed not block the entire process.

Multithreading models are three types:

1. Many to many relationship.
2. Many to one relationship.
3. One to one relationship.

1. Many to Many Model

In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine. Figure 2.11 shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.

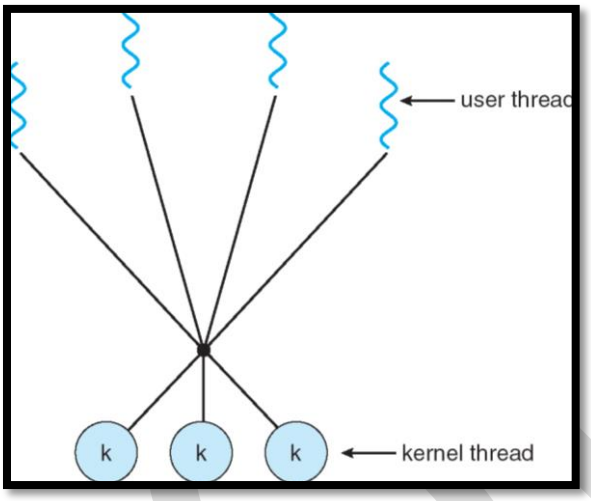


Figure 2.11 Many to many model

2. Many to One Model

Many to one model maps many user level threads to oneKernel level thread. Thread management is done in userspace. When thread makes a blocking system call, the entireprocess will be blocks. Only one thread can access theKernel at a time, so multiple threads are unable to run inparallel on multiprocessors.Figure 2.12 shows the many to one model. Ifthe user level thread libraries are implemented in theoperating system, that system does not support Kernelthreads use the many to one relationship modes.

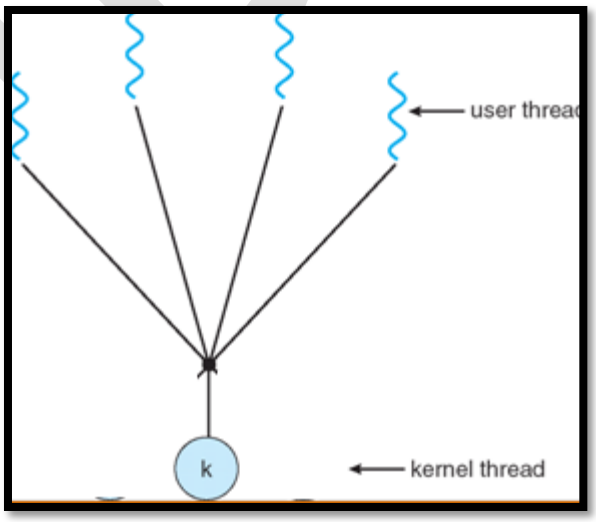


Figure 2.12 Many to one model

3. One to One Model

There is one to one relationship of user level thread to the kernel level thread. Figure 2.13 shows one to one relationshipmodel. This model provides more concurrency than themany to one model.It also another thread to run when a thread makes a blockingsystem call. It support multiple thread to execute in parallelon microprocessors. Disadvantage of this model is thatcreating user thread requires the corresponding Kernelthread. OS/2, Windows NT and windows 2000 use one toone relationship model.

2.3.5

2.4CPU SCHEDULING ALGORITHMS

2.4.1 Basic Concepts

The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization. Several processes are kept in memory at one time. Whenone process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. Everytime one process has to wait, another process can take over use of the CPU.Scheduling of this kind is a fundamental operating-system function.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
    1. Switches from running to waiting state.
    2. Switches from running to ready state.
    3. Switches from waiting to ready.
    4. Terminates.
- Scheduling under 1 and 4 is non preemptive.
- All other scheduling is preemptive.

Dispatcher

Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; thisinvolves:

- switching context
- switching to user mode
- jumping to the proper location in the user program to restart that program

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – Number of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Optimization Criteria

Max CPU utilization

Max throughput

Min turnaround time

Min waiting time

Min response time

2.4.2 Scheduling Algorithms:

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is tobe allocated first to the CPU. There are basically four types of CPU scheduling that exist.

1. **First Come, First Served Scheduling (FCFS) Algorithm:**This is the simplest CPU schedulingalgorithm. In this scheme, the process which requests the CPU first, that is allocated to the CPUfirst. The implementation of the FCFS algorithm is easily managed with a FIFO queue. When aprocess enters the ready queue its PCB is linked onto the rear of the queue. The average waitingtime under FCFS policy is quiet long.

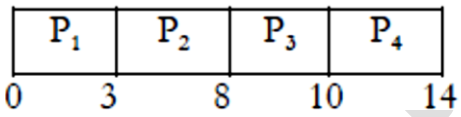
Consider the following example:

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

Using FCFS algorithm find the average waiting time and average turnaround time.

Solution:

Ganttchart:



The Turnaround time ofprocesses:

Process	**(Exit Time - Arrival Time) = Turnaround Time**
P1	(3-0) = 3
P2	(8-1) = 7
P3	(10-2)= 8
P4	(14-3)=11

Average Turnaround Time= (3+7+8+11) / 4 = 7.25

The waiting time of processes:

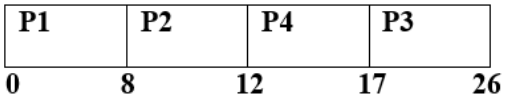
Process	(Turnaround Time –Burst Time) = Waiting Time
P1	(3-3)=0
P2	(7-5)=2
P3	(8-2)=6
P4	(11-4)=7

Average Waiting Time= (0+2+6+7) / 4 = 3.75

2. **Shortest Job First Scheduling (SJF) Algorithm:** This algorithm associates with each process if the CPU is available. This scheduling is also known as shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of the process rather than its total length.
Consider the following example:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart:



The Turnaround time of processes:

Process	(Exit Time - Arrival Time) = Turnaround Time
P1	(8-0) = 8
P2	(12-1) = 11
P3	(17-2)= 15
P4	(26-3)=23

Average Turnaround Time= (8+11+15+23) / 4 = 14.25

The waiting time of processes:

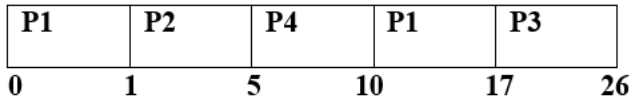
Process	(Turnaround Time –Burst Time) = Waiting Time
P1	(8-8)=0
P2	(11-4)=7
P3	(15-9)=6
P4	(23-5)=18

Average Waiting Time= (0+7+6+18) / 4 = 7.75

The SJF algorithm may be either preemptive or non preemptive algorithm. The preemptive SJFis also known as shortest remaining time first.
Consider the following example.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart:



The Turnaround time of processes:

Process	**(Exit Time - Arrival Time) = Turnaround Time**
P1	(17-0) = 17
P2	(5-1) = 4
P3	(26-2)= 24
P4	(10-3)= 7

Average Turnaround Time= (17+4+24+7) / 4 = 13

The waiting time of processes:

Process	**(Turnaround Time –Burst Time) = Waiting Time**
P1	(17-8)= 9
P2	(4-4)= 0
P3	(24-9)= 15
P4	(7-5)= 2

Average Waiting Time= (9+0+15+2) / 4 = 6.5

3. **Priority Scheduling:** A priority number (integer) is associated with each process. The CPU is allocated to the process with the highest priority (smallest integer means highest priority). Priority scheduling also has two types:

- Preemptive
- Non preemptive

SJF is a priority scheduling where priority is the predicted next CPU burst time. Equal priority processes arescheduled in FCFS manner.

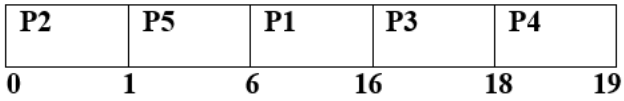
- Problem with priority →Starvation – means low priority processes may never execute
- Solution→ Aging - as time progresses increase the priority of the process (means Aging increases the priority of the processes so that to terminate in finite amount of time).

Consider the following example.

Process	Burst Time	Priority
P1	10	3
P2	1	1

P3	2	4
P4	1	5
P5	5	2

Gantt chart:



The average waiting time is 8.2 milliseconds.

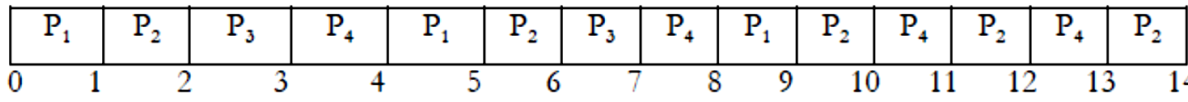
4. **Round Robin Scheduling Algorithm:** This type of algorithm is designed only for the time sharing system. It is similar to FCFS scheduling with preemption condition to switch between processes. A small unit of time called quantum time or time slice is used to switch between the processes. The average waiting time under the round robin policy is quiet long.

Consider the following example

Process	Burst Time
P1	3
P2	5
P3	2
P4	4

Time Quantum= 1

Gantt chart:



The waiting time for process

P1 = 0 + (4 – 1) + (8 – 5) = 0 + 3 + 3 = 6

P2 = 1 + (5 – 2) + (9 – 6) + (11 – 10) + (12 – 11) + (13 – 12) = 1 + 3 + 3 + 1 + 1 + 1 = 10

P3 = 2 + (6 – 3) = 2 + 3 = 5

P4 = 3 + (7 – 4) + (10 – 8) + (12 – 11) = 3 + 3 + 2 + 1 = 9

The average waiting time = (6 + 10 + 5 + 9)/4 = 7.5

Multilevel Queue Scheduling:

In multilevel queue scheduling, ready queue is partitioned into separate queues:foreground (interactive) and background (batch) processes. Each queue has its own scheduling algorithm. Foreground processes execute with round robin scheduling and background processes run with FCFS scheduling. Scheduling must be done between the queues. Fixed priority scheduling serves all from foreground then from background. Then it suffers from possibility of starvation. The solution for starvation can be as follows:

- Time slice – each queue gets a certain amount of CPU time which it can scheduleamongst its processes; i.e., 80% to foreground in RR
- 20% to background in FCFS

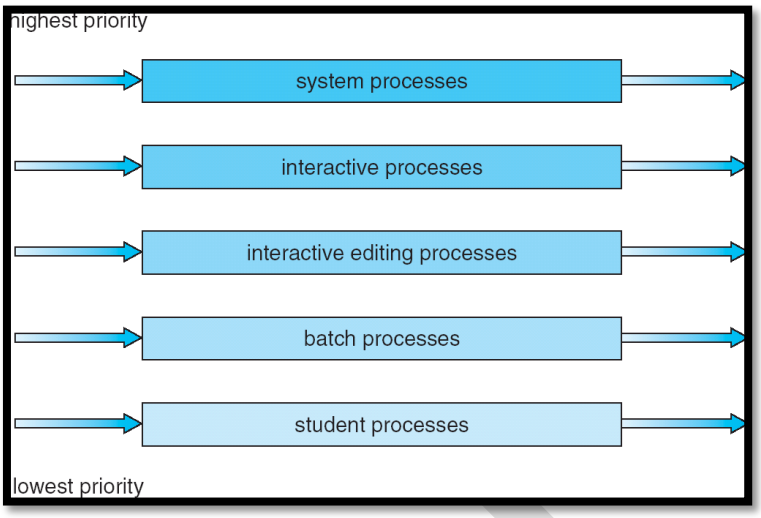


Figure 2.13 Multilevel Queue Scheduling

Multilevel Feedback Queue Scheduling:

In multilevel feedback queue scheduling a process can move between the various queues; aging can be implemented this way. Multilevel-feedback-queue scheduler defined by the following parameters:

- Numberof queues
- Schedulingalgorithms for each queue
- Methodused to determine when to upgrade a process
- Methodused to determine when to demote a process
- Methodused to determine which queue a process will enter when that process needsservice

Scheduling:

- A new job enters queue Q0 (RR) and is placed at the end. When it gains the CPU, the job receives 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to the end of queue Q1.
- A Q1 (RR) job receives 16 milliseconds. If it still does not complete, it is preempted and moved to queue Q2 (FCFS).

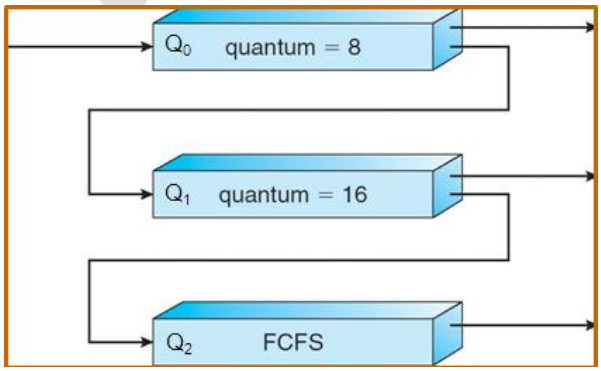


Figure 2.14 Multilevel Feedback Queue Scheduling

2.5 MULTIPROCESSOR SCHEDULING

In multiprocessor system multiple processors are working together. The processing load is distributed among these processors. CPU scheduling more complex when multiple CPUs are available, as most current general purpose processors are multiprocessors (i.e. multicoreprocessors). There is no single 'best' solution to multiple-processor scheduling. A multicore processor typically has two or more homogeneous processorcores, because the cores are all the same, any available processor can be allocated toany process in the system.

Approaches to Multiple-Processor Scheduling:

1. Asymmetric multiprocessing
   - All scheduling decisions, I/O processing, and other system activities handled bya single processor
   - Only one processor accesses the system data structures, alleviating the need fordata sharing

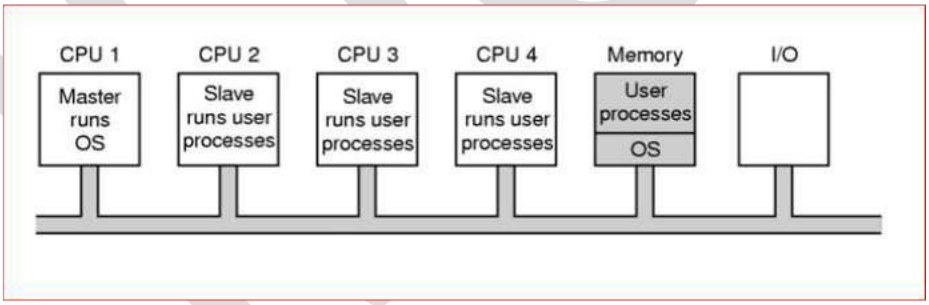


Figure 2.15 Asymmetric Multiprocessing

2. Symmetric multiprocessing (SMP)

- Each processor is self-scheduling
- All processes may be in a common ready queue, or each processor may have itsown private queue of ready processes
- Currently, most common approach to multiple-processor scheduling

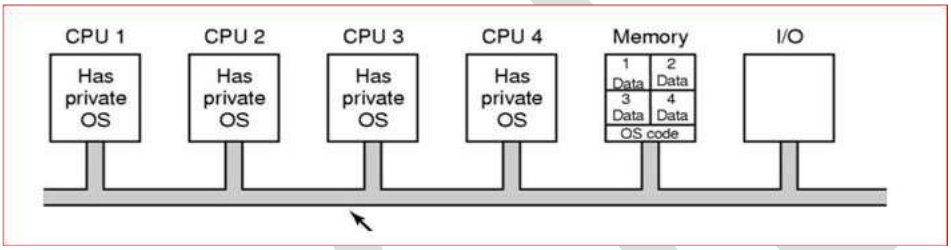


Figure 2.16 Symmetric Multiprocessing

Processor Affinity:

When a process runs on a specific processor there are certain effectson the cache memory. The data most recently accessed by the processpopulate the cache for the processor and as a result successivememory access by the process are often satisfied in the cache memory.Now if the process migrates to another processor, the contents of thecache memory must be invalidated for the first processor and the cachefor the second processor must be repopulated. Because of the highcost of invalidating and repopulating caches, most of theSMP(symmetric multiprocessing) systems try to avoid migration ofprocesses from one processor to another and try to keep a processrunning on the same processor. This is known as Processor Affinity.

There are two types of processor affinity:

1. Soft Affinity
2. Hard Affinity

Soft Affinity: When an operating system has a policy of attemptingto keep a process running on the same processor butnot guaranteeing it will do so, this situation is calledsoft affinity.

Hard Affinity: Some systems such as Linux also provide somesystem calls that support Hard Affinity which allows aprocess to migrate between processors.

Load Balancing: Load Balancing keepsthe workload evenly distributed across all processors in anSymmetric Multiprocessor System. Load balancing is necessary only on systemswhere each processor has its own private queue of processwhich are eligible to execute. Load balancing is unnecessarybecause once a processor becomes idle it immediately extractsa runnable process from the common run queue. OnSMP(symmetric multiprocessing), it is important to keep theworkload balanced among all processors to fully utilize thebenefits of having more than one processor else

one or moreprocessor will sit idle while other processors have highworkloads along with lists of processors awaiting the CPU.

Two approaches to load balancing:

- Push migration - a specific task periodically checks the load on each processorand rebalances the load if necessary.
- Pull migration - an idle processor can pull waiting tasks from a busy processor.

2.6 REAL TIME SCHEDULING

Correctness of the real time system depends not only on the logical result of the computation but also on the time at which the results are producedTasks or processes attempt to control or react to events that take place in the outside world. These events occur in "real time" and tasks must be able to keep up with them.

There are various classes of real time scheduling:

• Static table-driven approaches:

- performs a static analysis of feasible schedules of dispatching
- result is a schedule that determines, at run time, when a task must begin execution

• Static priority-driven preemptive approaches:

- a static analysis is performed but no schedule is drawn up
- analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used

• Dynamic planning-based approaches:

- feasibility is determined at run time rather than offline prior to the start of execution
- one result of the analysis is a schedule or plan that is used to decide when to dispatch this task

• Dynamic best effort approaches:

- no feasibility analysis is performed
- system tries to meet all deadlines and aborts any started process whose deadline is missed

Deadline Scheduling in real time systems:

- Real-time operating systems are designed with the objective of starting real-time tasks as rapidly as possible and emphasize rapid interrupt handling and task dispatching
- Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times
- Priorities provide a crude tool and do not capture the requirement of completion (or initiation) at the most valuable time

2.7 BIBLIOGRAPHY

1. Operating System Concepts (9th Ed) by Silberschatz and Galvin, Wiley, 2000.
2. Operating Systems (5th Ed) – Internals and Design Principles by William Stallings,Prentice Hall, 2000.
3. Modern Operating Systems by Andrew S Tanenbaum, Prentice Hall India, 1992.

2.8UNIT END EXERCISES

1. What is a process? Explain various process states.
2. What is thread? Explain various types of threads in detail.
3. Write short note on process control block.
4. What is scheduler? Differentiate between different types of schedulers.
5. For the processes listed below the table, draw Gantt chat and calculate averagewaiting time and average turnaround time using :-
   - FCFS (first come first serve)
   - SJF (Shortest Job First) in both condition pre-emptive and non-pre-emptive
   - Round – robin (Quantum = 2)

Process	Arrival Time	Burst Time
P1	0	9
P2	1	5
P3	2	7
P4	3	3

UNIT 3: Chapter 3

CONCURRENCY CONTROL

Unit Structure

3.15 Bibliography

3.16 Unit End Exercises

3.0 OBJECTIVES

After going through this unit, you will be able to:

- To elaborate Concurrency and Race condition, critical section problem.
- To implement software and hardware solutions of the critical section problem.
- To implement deadlock prevention, avoidance, detection and recovery algorithm.

3.1CONCEPT OF CONCURRENCY

- To introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data
- To present both software and hardware solutions of the critical-section problem
- To introduce the concept of an atomic transaction and describe mechanisms to ensure atomicity
- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer

Producer

```
while (true)
{
       /*  produce an item and put in nextProduced  */
    while (count == BUFFER_SIZE)
; // do nothing
  buffer [in] = nextProduced;
     in = (in + 1) % BUFFER_SIZE;
     count++;

}
```

Consumer

```
while (true)
{
       while (count == 0)
       ; // do nothing
       nextConsumed =  buffer[out];
       out = (out + 1) % BUFFER_SIZE;
       count--;
/*  consume the item in nextConsumed
}
```

3.2 RACE CONDITION

Race condition is the situation where several processes accessand manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.To prevent race conditions, concurrent processes must be synchronized.

The example of Race Condition:

count++ could be implemented as

```
register1 = count
register1 = register1 + 1
count = register1
```

count-- could be implemented as

```
register2 = count
register2 = register2 – 1
count = register2
```

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 value of countmay be either 4 or 6, where the correct result should be 5.
- Reverse of s4 & S5 may give counter=6.

3.2.1 The Critical-Section Problem

Suppose there are 'n' processesall competing tousesomeshared data, for example {P0, P1,…, Pn-1}
Eachprocesshasacodesegment,calledcriticalsection,inwhichtheshareddataisaccessed.(Changingcommonvariables, writefile, updatetableetc.)
Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, and then remainder section.

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

Figure 3.1 structure of critical section

3.2.2 Solution to Critical-Section Problem

A solution to the critical-section problem must satisfy the following threerequirements:

1. Mutual Exclusion-If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. 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.
3. 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 critical section and before that request is granted.

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

3.3REQUIREMENTS FOR MUTUAL EXCLUSION

---

1. Mutual exclusion must be enforced: Only one process at a time is allowed into its critical section, among all processes that have critical sections for the same resource or shared object.
2. A process that halts in its non-critical section must do so without interfering with other processes.
3. It must not be possible for a process requiring access to a critical section to be delayed indefinitely: no deadlock or starvation.
4. When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay.
5. No assumptions are made about relative process speeds or number of processors.
6. A process remains inside its critical section for a finite time only.

3.4SOFTEARE SOLUTION FOR MUTUAL EXCLUSION

Peterson's Solution for two processes

- Assume that the LOAD and STORE instructions are atomic; that is, cannot be interrupted.
- The two processes share two variables:

  int turn;

  Boolean flag[2]
- The variable turn indicates whose turn it is to enter the critical section.
- The flag array is used to indicate if a process is ready to enter the critical section.
- flag[i] = true implies that process Pi is ready!

Algorithm for Process Pi

```
do {
        flag[i] = TRUE;
        turn = j;
while (flag[j] && turn == j);
                critical section
flag[i] = FALSE;
remainder section
} while (TRUE);
```

The eventual value of turn determines which of the two processes is allowedto enter its critical section first.Peterson's solution proves that that the three critical requirement are met:

1. Mutual exclusion is preserved
   Pi enters in critical section only if:
   either flag[j] = false or turn = i
2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

3.5 HARDWARE SOLUTION FOR MUTUAL EXCLUSION

Software solutions for critical section does not work for modern computer systems. HenceSynchronization Hardware came into picture. Many systems provide hardware support for critical section code. Maximum solutions are based on idea of locking, they protectcritical regions via locks. The uniprocessors use solution to disable interrupts.

3.5.1 Solution to Critical-section Problem Using Locks

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

Solution using TestAndSet

This solution uses a shared boolean variable lock which is initialized to false.Processcanentercriticalsectiononlywhenlock=FALSE.

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

do {
          while ( TestAndSet (&lock ))
             ;   // do nothing
                 //    critical section
lock = FALSE;
                //     remainder section
 } while (TRUE);
```

Solution using Swap

Swap instruction automatically swaps two variables.Processcan enter critical section only if lock==false & key == true. lock==true and key ==false means lock is acquired.

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

It uses a shared Boolean variable lock initialized to FALSE; each process has a local Boolean variable key Solution:

```
        do {
               key = TRUE;
               while ( key == TRUE)
    Swap (&lock, &key );
                            //    critical section
      lock = FALSE;
                        //     remainder section
    } while (TRUE);
```

Bounded-waiting Mutual Exclusion with TestandSet()

```
do {
```

```
 waiting[i] = TRUE;
key = TRUE;
while (waiting[i] && key)
        key = TestAndSet(&lock);
waiting[i] = FALSE;
        // critical section
j = (i + 1) % n;
while ((j != i) && !waiting[j])
j = (j + 1) % n;
        if (j == i)
lock = FALSE;
else
        waiting[j] = FALSE;
// remainder section
} while (TRUE);
```

The advantages and disadvantages of locking solutions are as follows:

Advantages

1. It is simple and easy to verify.
2. Itis applicable to any number of processes.
3. Itcan be used to support multiple critical section.

Disadvantages

1. Busy waiting is possible.
2. Starvation is also possible.
3. There may be deadlock.

3.5.2 Interrupt Disabling:

In a uniprocessor machine, concurrent processes cannot be overlapped; they can only be interleaved. Furthermore, a process will continue to run until it invokes an operating system service or until it is interrupted. Therefore, to guarantee mutual exclusion, it is sufficient to prevent a process from being interrupted. This capability can be provided in the form of primitives defined by the system kernel for disabling and enabling interrupts.

```
while (true)
{
disable interrupts()
        //critical section
enable interrupts()
       // remainder
}
```

Because the critical section cannot be interrupted, mutual exclusion is guaranteed.

Disadvantages

- It works only in a single processor environment.
- Interrupts can be lost if not serviced promptly.

- A process waiting to enter its critical section could suffer from starvation.

3.6 SEMAPHORE

Semaphores:

For the solution to the critical section problem one synchronization tool is used which is known as semaphores. A semaphore 'S' is an integer variable which is accessed through two standard operations suchas wait and signal. These operations were originally termed 'P' (for wait means to test) and 'V' (for single means to increment). The classical definition of wait is

```
Wait (S)
{
        While (S <= 0)
        {
                Test;
        }
        S--;
}
```

The classical definition of the signal is

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

In case of wait the test condition is executed with interruption and the decrement is executed without interruption.

Properties of Semaphore:

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Canhavemanydifferent critical sectionswithdifferent semaphores.
5. Semaphore acquire many resources simultaneously.

Drawback of Semaphore:

1. They are essentially shared global variables.
2. Access to semaphores can come from anywhere in a program.
3. There is no control or guarantee of proper usage.
4. There is no linguistic connection between the semaphore and the data to which the semaphorecontrols access.
5. Theyservetwopurposes,mutualexclusionand scheduling constraints.

Two Types of Semaphores

1. Countingsemaphore –integer value can range over an unrestricted domain.
2. Binarysemaphore –integer value can range only between 0 and 1; can be simpler to implement.

Binary Semaphore:

- A binary semaphore is a semaphore with an integer value which can range between 0 and 1.
- Let 'S' be a counting semaphore.
- To implement the binary semaphore we need following the structure of data.
    - Binary Semaphores S1, S2;
    - int C;
- Initially S1 = 1, S2 = 0 and the value of C is set to the initial value of the counting semaphore 'S'.

Then the wait operation of the binary semaphore can be implemented as follows:

```
Wait (S1)
C--;
if (C < 0)
{
        Signal (S1);
        Wait (S2);
}
Signal (S1);
```

The signal operation of the binary semaphore can be implemented as follows:

```
Wait (S1);
C++;
if (C <=0)
Signal (S2);
else
Signal (S1);
```

3.7CLASSICAL IPC PROBLEMS AND SOLUTIONS

There are various types of Interprocess communication Problems (IPC) which are proposed for synchronization scheme such as:

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

3.7.1 Bounded Buffer Problem:

This problem was commonly used to illustrate the power of synchronization primitives. In this scheme the pool consists of 'N' buffer and each capable of holding one item. The 'mutex' semaphore provides mutual exclusion for access to the buffer pool and is initialized to the value one. The empty and full semaphores count the number of empty and full buffer respectively. The semaphore empty is initialized to 'N' and the semaphore full is initialized to zero. This problem is known as procedure and consumer problem. The code of the producer is producing full buffer and the code of consumer is producing empty buffer.

The structure of producer process is as follows:

```
do
 {
 produce an item in nextp
. . . . . .
Wait (empty);
Wait (mutex);
. . . . . .
add nextp to buffer
. . . . . .
Signal (mutex);
Signal (full);
} While (1);
```

The structure of consumer process is as follows:

```
do {
Wait (full); Wait (mutex);
. . . . . .
Remove an item from buffer to nextc
. . . . . .
Signal (mutex);
Signal (empty);
. . . . . .
Consume the item in nextc;
. . . . . .. . .. .
} While (1);
```

3.7.2 Reader Writer Problem

In this type of problem there are two types of process are used such as Reader process and Writer process. The reader process is responsible for only reading and the writer process is responsible for writing. In Reader –Writer problem, 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

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

Shared Data used by Reader-writer problem:

- Data set
- Semaphore mutex initialized to 1
- Semaphore wrt initialized to 1
- Integer readcount initialized to 0

The structure of a writer process:

```
        do {
            wait (wrt) ;
   //   writing is performed
```

```
            signal (wrt) ;
        } while (TRUE);
```

The structure of a reader process:

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

3.7.3 Dining Philosopher Problem:

Consider 5 philosophers to spend their lives in thinking & eating. A philosopher shares common circular table surrounded by 5 chairs each occupies by one philosopher. In the centre of the table there is a bowl of rice and the table is laid with 6 chopsticks as shown in below figure 3.2.

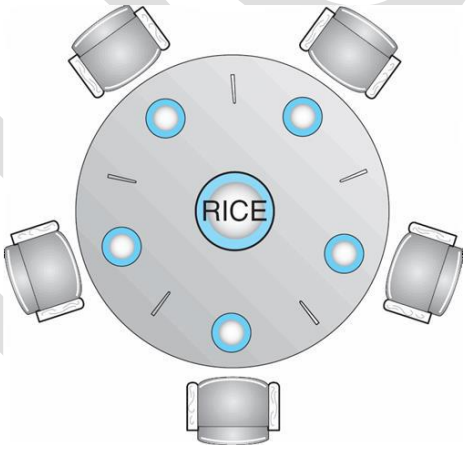


Figure 3.2 Dining Philosopher Problem

A philosopher can think without interacting with her colleagues. If a philosopher gets hungry, pickup two chopsticks that are closest to her. A philosopher may pickup one chopstick or two chopsticks at a time but she cannot pickup a chopstick that is already in hand of the neighbour. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she finished eating, she puts down both of her chopsticks and starts thinking again. This problem is considered as classic synchronization problem. According to this problem each chopstick is represented by a semaphore. A philosopher grabs the chopsticks by executing the wait operation on that semaphore. She releases the chopsticks by executing the signal operation on the appropriate semaphore.

Shared data in Dining Philosopher Problem:

- Bowl of rice (data set)
- Semaphore chopstick [5], initialized to 1.

The structure of philosopher i is as follows:

```
do {
    wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );

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

        // think
} while (TRUE);
```

3.8 MONITOR

The monitor is a programming-language construct that provides equivalent functionality to that of semaphores and that is easier to control. It is implemented in a number of programming languages, including Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3, and Java.

Characteristics of Monitor:

1. Local data variables are accessible only by the monitor
2. Process enters monitor by invoking one of its procedures
3. Only one process may be executing in the monitor at a time, any other processes that have invokedthe monitor are blocked, waiting for the monitor to become available.

In monitors, only one process may be active within the monitor at a time.Proceduredefinedwithina monitorcanaccessonlythosevariablesdeclaredlocallywithinmonitor. Figure 3.3 shows the basic structure of monitors.

```
monitor monitor-name
{
// shared variable declarations
procedure P1 (…) { …. }
…
procedure Pn(…) {……}

Initialization code ( ….) { … }
…
```

}
}

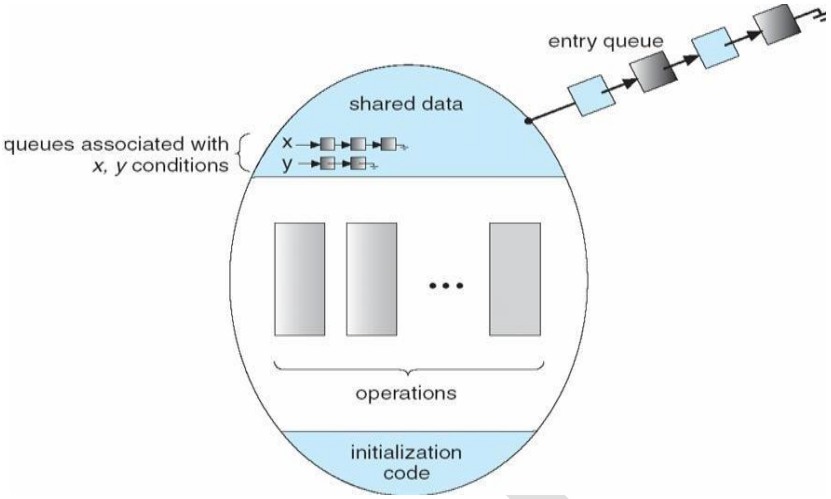


Figure 3.3 structure of monitor

Condition Variables used in monitor:

- Synchronization achieved by condition variables within a monitor, only accessible by the monitor.
- Two operations on a condition variable:
    - x.wait() –process that invokes the operation is suspended until x.signal()
    - x.signal() –resumes one of the processes(if any)that invokedx.wait()

3.9 DEADLOCK

In a multiprogramming environment several processes may compete for a finite number of resources. A process request resources; if the resource is available at that time a process enters the wait state. Waiting process may never change its state because the resources requested are held by other waiting process. This situation is known as deadlock.

Example

- System has 2 disk drives, P1 and P2 each hold one disk drive and each needs another one.
- 2 train approaches each other at crossing, both will come to full stop and neither shall start until other has gone.

System Model:

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types each of which consists of a number of identical instances. A process may utilized a resources in the following sequence

- **Request:** In this state one can request a resource.
- **Use:** In this state the process operates on the resource.
- **Release:** In this state the process releases the resources.

3.10 DEADLOCKCHARACTERISTICS (NECESSARY CONDITIONS FOR DEADLOCK)

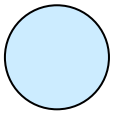
In a deadlock process never finish executing and system resources are tied up. A deadlock situation can arise if the following four conditions hold simultaneously in a system.

- **Mutual Exclusion:** At a time only one process can use the resources. If another process requests that resource, requesting process must wait until the resource has been released.
- **Hold and wait:** A process must be holding at least one resource and waiting to additional resource that is currently held by other processes.
- **No Preemption:** Resources allocated to a process can't be forcibly taken out from it unless it releases that resource after completing the task.
- **Circular Wait:** A set $\{P_0, P_1, \ldots\ldots P_n\}$ of waiting state/ process must exists such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for the resource that is held by P_2 ….. $P_{(n-1)}$ is waiting for the resource that is held by P_n and P_n is waiting for the resources that is held by P_4.

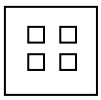
3.10.1 Resource Allocation Graph:

- Deadlock can be described more clearly by directed graph which is called system resource allocation graph.
- The graph consists of a set of vertices 'V' and a set of edges 'E'.
- The set of vertices 'V' is partitioned into two different types of nodes such as:
    - $P = \{P_1, P_2, \ldots\ldots P_n\}$, the set of all the active processes in the system
    - $R = \{R_1, R_2, \ldots\ldots R_m\}$, the set of all the resource type in the system.
- There are two types of edges:
    - request edge – directed edge $P_i \rightarrow R_j$
    - assignment edge – directed edge $R_j \rightarrow P_i$

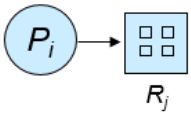
- Process



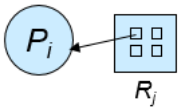
- Resource Type with 4 instances



- P_i is holding an instance of R_j



- P_i is holding an instance of R_j



Example of Resource Allocation Graph:

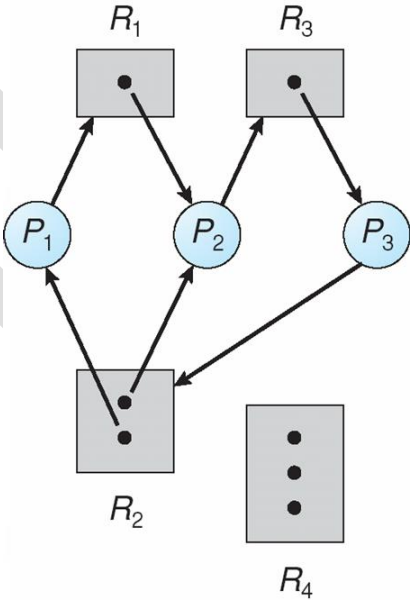


Figure 3.4 ***Resource Allocation Graph***

Basic Facts about Resource Allocation Graph:

- If graph contains no cycles → no deadlock.
- If graph contains cycle →

- Ifonly one instance per resource type then deadlock.
- Ifseveral instances per resource type, possibility of deadlock.

Resource Allocation Graph With a Deadlock

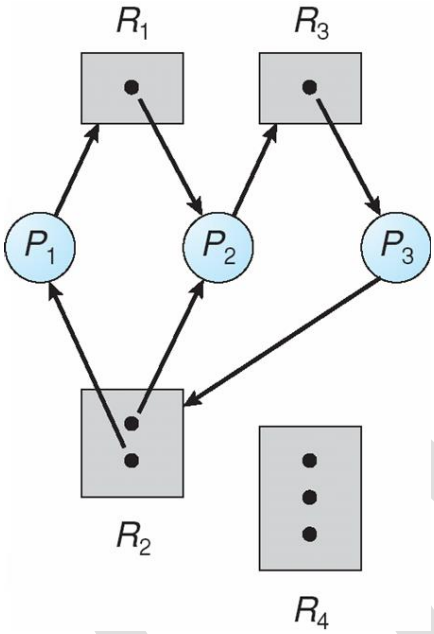


Figure 3.5 Resource Allocation Graph With a Deadlock

Above resource allocation graph shows two cycles:

1. P1→ R1→ P2→ R3→ P3→ R2→ P1
2. P2→ R3→ P3→ R2→ P2

Resource Allocation Graph with a Cycle but No Deadlock

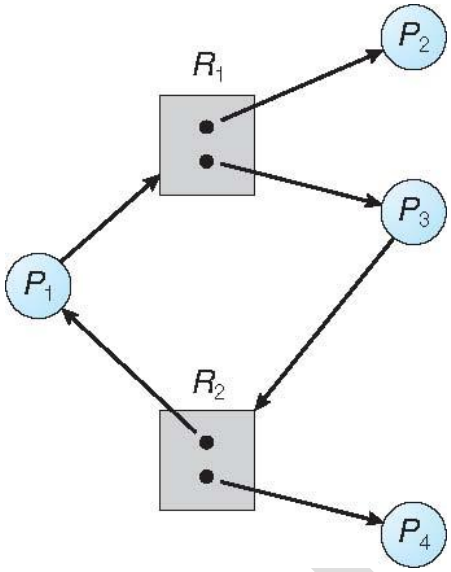


Figure 3.6 Resource Allocation Graph with a Cycle but No Deadlock

- Above graph shows a cycle P1→ R1→ P3→ R2→ P1
- Process P4 may release its instances of resource type R2. ThatresourcecanbeallocatedtoP3, thus breakingthecycle.

3.10.2 Methods for Handling Deadlocks

- Ensure that the system will never enter a deadlock state.
    - Prevention: prevent any one of the four conditions from happening.
    - Avoidance: Allow all deadlock conditions, but calculate cycles about to happen and stop dangerous operations. (Knowledge of requests & resources.)
- Allow the system to enter a deadlock state and then recover.
    - Detection: know a deadlock has occurred.
    - Recovery: regain the resources
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX. (manually system restarted)

3.11 DEADLOCK PREVENTION

Deadlock prevention is a set of methods for ensuring that at least one of these necessary conditions cannot hold.

Mutual Exclusion: The mutual exclusion condition holds for non-sharable. The example is a printer cannot be simultaneously shared by several processes. Sharable resources do not require mutual exclusive access and thus cannot be involved in a dead lock. The example is read only files which are in sharing condition. If several processes attempt to open the read only file at the same time they can be guaranteed simultaneous access.

Hold and wait:

- It must guarantee that whenever a process requests a resource, it does not hold any other resources.
- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
- Low resource utilization; starvation possible

No Preemption:

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Pre-empted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Circular Wait:

- We can ensure that this condition never holds by ordering of all resource type and to require that each process requests resource in an increasing order of enumeration.
- Let R = {R_1, R_2, …….R_n}be the set of resource types.
- Each resource type has a unique integer number, which allows us to compare two resources and to determine whether one precedes another in ordering.
- Imposea total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

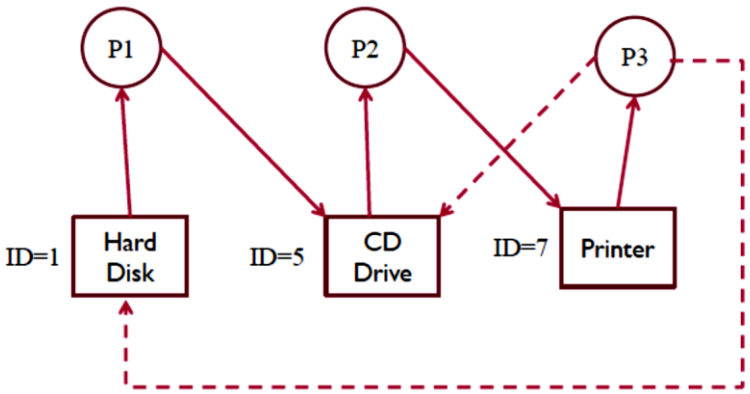


Figure 3.7 circular wait solution

3.12DEADLOCK AVOIDANCE

Deadlock avoidance requiresthat the system has some additional a priori informationavailable.

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

3.12.1Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in safe state if there exists a sequence <P1, P2, …, Pn> of ALL the processes in the systems such that for each P_i, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the P_j, with j <i .
- That is:
    - If P_i resource needs are not immediately available, then P_i can wait until all $P_j$have finished.
    - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

- Basic facts about safe state:
  - If system is in safe state => No deadlock
  - If system in not in safe state => possibility of deadlock
  - Avoidance means to ensure that system will never enter an unsafe state, prevent getting into deadlock

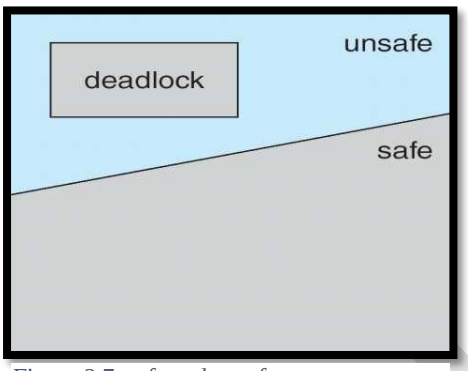


Figure 3.7 safe and unsafe states

Types of Deadlock Avoidance Algorithms:

1. Resource-allocation graph: It is used when single instance of a resource type is available.
2. Banker's algorithm: It is used when multiple instances of a resource type are available.

3.12.2 Resource Allocation Graph Algorithm

In this graph a new type of edge has been introduced is known as claim edge. Claim edge $P_i \rightarrow R_j$ indicates that process P_j may request resource R_j; represented by a dashed line. Claim edge converts to request edge when a process requests a resource. Request edge converted to an assignment edge when the resource is allocated to the process. When a resource is released by a process, assignment edge reconverts to a claim edge. Resources must be claimed a priori in the system.

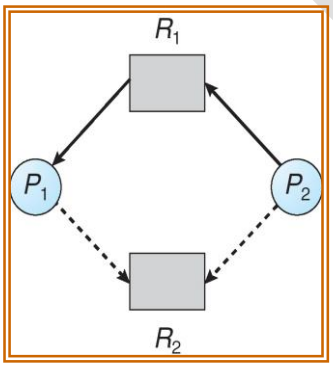


Figure 3.8 Resource Allocation Graph 1

P2 requesting R1, but R1 is already allocated to P1. Both processes have a claim on resource R2

What happens if P2 now requests resource R2?

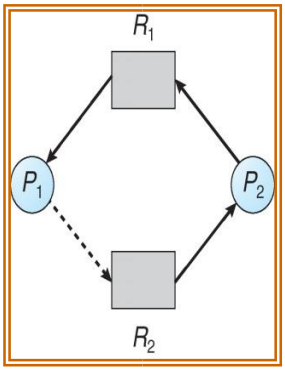


Figure 3.9 Resource Allocation Graph 2

Cannot allocate resource R2 to process P2

Why? Because resulting state is unsafe

- P1 could request R2, thereby creating deadlock!

Use only when there is a single instance of each resource type

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.
- Here we check for safety by using cycle-detection algorithm.

3.12.3 Banker's Algorithm

This algorithm can be used in banking system to ensure that the bank never allocates all its available cash such that it can no longer satisfy the needs of all its customer. This algorithm is applicable to a system with multiple instances of each resource type. When a new process enter in to the system it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. Several data structure must be maintained to implement the banker's algorithm.

Let,

- n = number of processes
- m = number of resources types

Available: Vector of length m. If Available[j] = k, there are k instances of resource type $R_j$available.

Max: n x m matrix. If Max [i,j] = k, then process $P_i$may request at most k instances of resource type R_j.

Allocation: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j.

Need: n x m matrix. If Need[i,j] = k, then P_i may need k more instances of R_jo complete its task.

Need [i,j] = Max[i,j] – Allocation [i,j].

Safety Algorithm

1. Let Workand Finish be vectors of length m and n, respectively.

   Initialize: Work = Available

Finish [i] = false for i = 0, 1, …,n- 1.

2. Find and i such that both:

   (a) Finish [i] = false

   (b) $Need_i \leq Work$

If no such i exists, go to step 4.

3. Work = Work + $Allocation_i$

   Finish[i] = true go to step 2.

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

Example of Banker's Algorithm:

- Suppose there are 5 processes P_0 through P_4;
- 3 resource types: A (10 instances), B (5instances), and C (7 instances).
- Snapshot at time T_0:

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

- The content of the matrix Need is defined to be Max – Allocation.

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

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

Resource Request Algorithm

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:
   Available = Available – $Request_i$;
   $Allocation_i$ = $Allocation_i$ + $Request_i$;
   $Need_i$=$Need_i$ – $Request_i$;

- If safe ⇒ the resources are allocated to Pi.
- If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

Example of Resource Request Algorithm:

If process P_1 requests additional request of resources (1, 0, 2), can it be granted immediately?

- Check that Request ≤ Available (that is, (1, 0, 2) ≤ (3, 3, 2) ⇒ true.

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

- Executing safety algorithm shows that sequence <P_1, P_3, P_4, P_0, P_2> satisfies safety requirement.
- Can request for (3,3,0) by P_4 be granted? –NO
- Can request for (0,2,0) by P_0 be granted? –NO (Results Unsafe)

3.13 DEADLOCK DETECTION

If a system doesn't employ either a deadlock prevention or deadlock avoidance, then deadlock situation may occur. In this environment the system does the following things:

1. Allowsystem to enter deadlock state
2. Apply Deadlock Detection algorithm
3. Apply Recovery scheme

The system can have two variations. According to the type of a system, different type of deadlock detection method is used.

1. Single Instance of a Resource type
2. Several Instances of each Resource type

Single Instance of each Resource type

If all resources only a single instance then we can define a deadlock detection algorithm which uses a new form of resource allocation graph called "Wait for graph". We obtain this graph from the resource allocation graph by removing the nodes of type resource and collapsing the appropriate edges. The below figure describes the resource allocation graph and corresponding wait for graph.

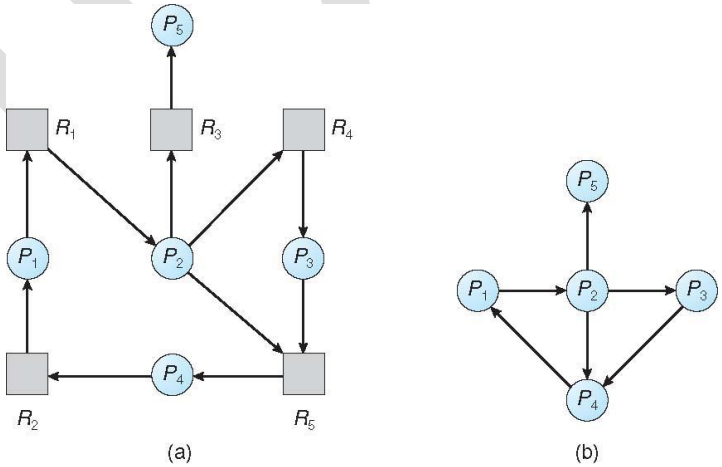


Figure3.10 (a) Resource Allocation graph (b) corresponding wait for graph

- For single instance
- P_i ->P_j(P_i is waiting for P_j to release a resource that P_i needs)
- P_i->P_j exist if and only if RAG contains 2 edges P_i ->R_q and R_q ->P_j for some resource R_q

Several Instances of a Resource type

The wait for graph scheme is not applicable to a resource allocation system with multiple instances of reach resource type. For this case the algorithm employs several data structures which are similar to those used in the banker's algorithm like available, allocation and request.

- **Available**: A vector of length m indicates the number of available resources 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.

1. Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if $Allocation_i \neq 0$, then Finish[i] = false;otherwise, Finish[i] = true.

2. Find an index i such that both:

(a) Finish[i] == false

(b) $Request_i \leq Work$

If no such i exists, go to step 4.

3. Work = Work + Allocation

Finish [i] = true

Go to step 2

4. If Finish [i] = false, for some i, $1 \leq i \leq n$, then the system is in a deadlock state. Moreover, if Finish [i] = false, then process P_i is deadlocked.

Example of deadlock detection:

Consider a system with

- Five processes P_0 through P_4;
- three resource types A (7 instances), *B* (2 instances), and *C* (6 instances)

	Allocation	Need	Available

	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

In above example Sequence <P0, P2, P3, P1, P4> will result in Finish[i] = true for all i.

If process P2 requests an additional instance of type C, The Request matrix is modified as follows:

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

Now the system is deadlocked. Although the resources held by process P_0, are reclaimed, the number of available resources are not sufficient to fulfil the requests of the other processes. Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

3.14 RECOVERY FROM DEADLOCK

When a detection algorithm determines that a deadlock exists, several alternatives exist. One possibility is to inform the operator that a deadlock has occurred, and to let the operator deal with the deadlock manually. The other possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock. One solution is simply to abort one or more processes to break the circular wait. The second option is to preempt some resources from one or more of the deadlocked processes.

3.14.1 Process Termination:

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

- **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at a great expense; these processes may have computed for a long time, and the results of these partial computations must be discarded and probably recomputed later.
- **Abort one process at a time until the deadlock cycle is eliminated:** This method incurs considerable overhead, since after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

3.14.2 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. If preemption is required to deal with deadlocks, then three issues need to be addressed.

- **Selecting a victim:** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the numbers of resources a deadlock process is holding, and the amount of time a deadlocked process has thus far consumed during its execution.
- **Rollback:** If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must rollback the process to some safe state, and restart it from that state.
- **Starvation:** In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation that needs to be dealt with in any practical system. Clearly, we must ensure that a process can be picked as a victim only a small finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

3.15
BIBLIOGRAPHY

---

4. Operating System Concepts (9th Ed) by Silberschatz and Galvin, Wiley, 2000.
5. Operating Systems (5th Ed) – Internals and Design Principles by William Stallings, Prentice Hall, 2000.
6. Modern Operating Systems by Andrew S Tanenbaum, Prentice Hall India, 1992.

3.16 UNIT END EXERCISES

6. Explain Peterson's solution to solve critical section problem.
7. What is semaphore? Explain with example.
8. Differentiate between semaphore and monitor.
9. What is deadlock? What are necessary conditions to happen deadlock?
10. Explain Banker's algorithm with example.
11. Consider the following snapshot of a system:

	Allocation	Need	Available
	A B C D	A B C D	A B C D
P_0	0012	0012	1520
P_1	1000	1750	
P_2	1354	2356	
P_3	0632	0652	
P_4	0014	0656	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need?*

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately?

UNIT 4 - Chapter 4

Memory Management

Unit Structure

4.0 OBJECTIVE

After completion of this unit, you will be able to answer:

- Detailed insights of memory hardware organization in a computer system.
- Different techniques of memory management in Operating System.
- Detailed concept of virtual memory and related concepts.
- Concepts of Page Replacement and Allocation algorithms

4.1 INTRODUCTION

The Central Processing Unit (CPU) in a computer system can only load the instructions from memory, so if there are any programs, then these mustbe loaded first into the memory to be get executed. Generalpurpose computers systems run mostof their programs from available memory, called random access memory (RAM) or the main memory which is rewritable. Most commonly, the main memory is implemented using asemiconductor-basedtechnology which is also called as dynamic random-access memory (DRAM).Other forms of memory, a computer system uses, are also available. For example, the first program to run when a computer is switched on, is called as a bootstrap program, which further loadsthe operating

system. As RAM is volatile memory it loses its content when power of computer system is turned off otherwise lost when the bootstrap program does not run properly.Instead of this,there is one more form of a memory called as electricallyerasable programmable read-only memory (EEPROM) and other forms offirmware storage. This memory is so infrequently written to and is non-volatile. EEPROMcan be changed but not so frequently. In addition to this, it is low speed, and therefore it contains largely static programs and data that are not often used. For example, the cellular phone uses EEPROM to store serial numbers and hardware information about the device. All forms of memory provide an array of bytes. Each byte has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses. The load instruction pushes a byte orwordfrom main memory to an internal register within the CPU, whereas the storeinstruction moves the content of a register to main memory. Apart fromexplicitloads and stores, the CPU automatically loads instructions from main memoryfor execution from the location stored in the program counter.A conventional instruction–execution cycle, as executed on a system with a, first fetches an instruction from memory and storesthat instruction in the instruction register. The instruction is then decodedand may cause operands to be retrieved from memory and stored in someinternal register. After the instruction on the operands has been executed, theresult may be stored back in memory. Notice that the memory unit sees onlya stream of memory addresses. It does not know how they are generatedi.e. either by the instruction counter, indexing, indirection, literal addresses, or some othermeans or what they are for (instructions or data).Preferably, we want the programs and data to reside in main memory permanently.There are two main reasons for non-possibility of this type of arrangementin so many types of computer systems:

1. The main memory is mostly too small to store all required programs and datapermanently.
2.The main memory is volatile that means it loses all its contentswhen turning off the power.Therefore, most of the computer systems provide secondary storageas an extension ofmain memory. The main requirement for secondary storage is that it be able toholdlarge quantities of data permanently.

Thesecondarystorage devices are hard-disk drives (HDDs)and other nonvolatile memory devices, which provide storage for bothprograms and data. Most systems and applications are stored insecondary type of storage until they are loaded into memory. Many programs make useof secondary storage as both the source and the destination of their processing.Secondary storage is also a lot slower than main memory. Therefore, the propermanagement of secondary storage is of central importance to a computer system.Other possible components includecache memory, CD-ROM or DVD, magnetic tapes, etc. Those that areslow in speed as well as large enough as per the capacity are used only for special purposeslike to store backup copies of material stored on other devices, also called as tertiary storage. The main differences among the various storage systems lie in speed, size, andvolatility.If you refer Figure. 4.1 then you can see that there are wide variety of storage systems which are organized in a hierarchyaccording to access time and the storage capacity. As per a general rule, there is a trade-off between size and speed,with smaller and faster memory closer to theCPU. As shown in the figure, in addition to differing in speed and capacity, thevarious

storage systems are either volatile or nonvolatile. Volatile storage, asmentioned earlier, loses its contents when the power is turned off to the device,so data must be stored or written to some nonvolatile storage device for safety and further usage.The upper four levels of memory in the figure are constructed using semiconductormemory, which consists of semiconductor-based electronic circuits.Non VolatileMemory (NVM) devices, at the fourth level, have several variants but in general are fasterthan hard disks. Themost common form of NVM device is flash memory,whichis popular in mobile devices such as smartphones and tablets. Flash memory is now being used for long-term storage on laptops, desktops, andservers as well.

Examples of a Volatile storage which is also termed simply as memory are registers, RAM etc.

Examples of nonvolatile storage are typically categorised into two types:

a) Mechanical storage such as HDDs, opticaldisks, holographic storage, magnetic tape, etc.
b) Electrical storage such as flash memory,FRAM, NRAM, and SSD.

Mechanical storage is generally larger and less expensive per byte as compared to electrical storage. Whereas electrical storage is typically costly, smaller,and faster than mechanical storage. Caches canbe installed to improve performance where a large disparity in access time ortransfer rate exists between two components.

The main memory is central to the operation of a modern computer system. Main memory is a large array of bytes, ranging in size from hundreds of thousands to billions. Each byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The CPU reads instructions from main memory during the instruction-fetch cycle and both reads and writes data from main memory during the data-fetch cycle (on a von Neumann architecture). As noted earlier, the main memory is generally the only large storage device that the CPU is able to address and access directly. For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls. In the same way, 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. Eventually, the program terminates, its memory space is declared available, and the next program can be loaded and executed. To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, results into a need for memory management.

Many different memory management techniques & schemes are used. These schemes reflect various approaches, and the effectiveness of any given algorithm depends on the situation. In selecting a memory-management scheme for a specific system, we must take into account many factors—especially the hardwaredesign of the system. Each algorithm requires its own hardware support.

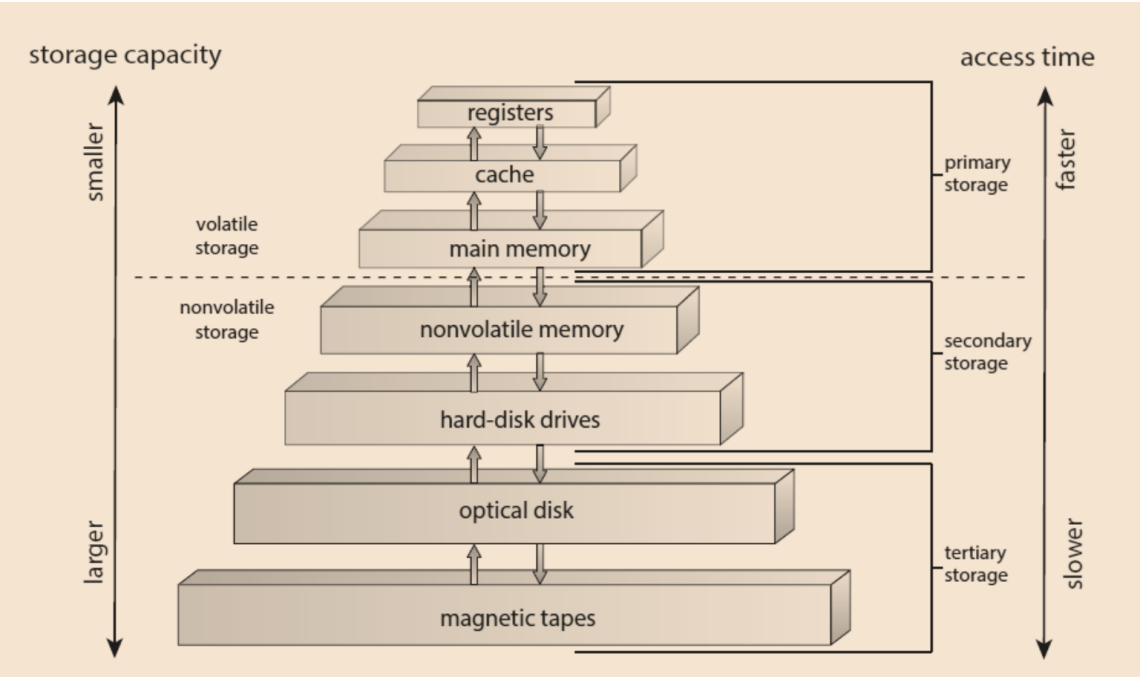


Fig 4.1 Storage Device Order

4.2 Memory Management

In this unit, we would have a detailed view of various ways to manage memory. The memorymanagementalgorithms vary from a primitive bare-machine approach to ascheme that uses paging. Each approach towards memory management has its own advantages and disadvantages.Selection of a memorymanagement method for a specific systemdepends on various factors, mainly on the hardware design of the system.Most of the memory management-basedalgorithms require hardware support, guiding manysystems to get closely integrated hardware and operating-system.

4.2.1 Memory Partitioning

The binding of instructions and data to memory addressescan be done at any step along the way:

- **Compile time:** If it is known at compile time where theprocess will reside in memory, then absolute code can begenerated.
- **Load time:** If it is not known at compile time where theprocess will reside in memory, then the compiler mustgenerate re-locatable code.
- **Execution time:** If the process can be moved during itsexecution from one memory segment to another, thenbinding must be delayed until run time.

Dynamic Loading

Better memory-space utilization can be done by dynamicloading. It has been necessary for the entire program and all data of a process to be in physical memory for the process to implement. The size of a process has thus been limited to the size of physical memory. To obtain better memory-space utilization, we can use dynamic loading. With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If it has not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine. The advantage of dynamic loading is that a routine is loaded only when it is needed. This method is particularly useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines. In such a situation, although the total program size may be large, the portion that is used (and hence loaded) may be much smaller. Dynamic loading does not require special support from the operating system. It is the responsibility of the users to design their programs to take advantage of such a method. Operating systems may help the programmer, however, by providing library routines to implement dynamic loading.

Dynamic Linking

Most of the operating systems support only static linking, in whichDynamic Link Libraries (DLL) system language libraries are treated like any other object moduleand are combined by the loader into the binary program image. Dynamic linking, in contrast, is very similar to dynamic loading. Here, though, linking is postponed until execution time. This feature is usually used with system libraries, such as the standard C language library. Without this facility, each program on a system must include a copy of its language library (or at least the routines referenced by the program) in the executable image. This requirement not only increases the size of an executable image but also may waste main memory. A second advantage of DLLs is that these libraries can be shared among multiple processes, so that only one instance of the DLL in main memory. For this reason, DLLs are also known as shared libraries, and are used extensively in Windows and Linux systems. When a program references a routine that is in a dynamic library, the loader locates the DLL, loading it into memory if necessary. It then adjusts addresses that reference functions in the dynamic library to the location in memory where the DLL is stored. Dynamically linked libraries can be extended to library updates (such asbug fixes). In addition, a library may be replaced by a new version, and allprograms that reference the library will automatically use the new version.Without dynamic linking, all such programs would need to be relinked to gainaccess to the new library. So that programs will not accidentally execute new,incompatible versions of libraries, version information is included in both theprogram and the library. More than one version of a library may be loadedinto memory, and each program uses its version information to decide whichcopy of the library to use. Versions with minor changes retain the same versionnumber, whereas versions with major changes increment the number. Thus,only programs that are compiled with the new library version are affected byany incompatible changes incorporated in it. Other programs linked before thenew library was installed will continue using the older library.Unlike dynamic loading, dynamic linking and shared libraries

generallyrequire help from the operating system. If the processes in memory are protectedfrom one another, then the operating system is the only entity that cancheck to see whether the needed routine is in another process's memory spaceor that can allow multiple processes to access the same memory addresses.
Theconcept of dynamic linking is very similar to that of dynamic loading.Rather than loading being postponed until execution time, linking ispostponed. This feature is usually used with system libraries, suchas language subroutine libraries. With dynamic linking, a stub isincluded in the image for each library-routine reference. This stub isa small piece of code that indicates how to locate the appropriatememory-resident library routing.The entire program and data of a process must be inphysical memory for the process to execute. The size of a processis limited to the size of physical memory. So that a process can belarger than the amount of memory allocated to it, a technique calledoverlays is sometimes used. The idea of overlays is to keep inmemory only those instructions and data that are needed at anygiven time. When other instructions are needed, they are loadedinto space that was occupied previously by instructions that are nolonger needed.

Example: consider a two-pass assembler. During pass 1, itconstructs a symbol table; then, during pass 2, it generatesmachine-language code. We may be able to partition such anassembler into pass 1 code, pass 2 code, the symbol table1, and common support routines used by both pass 1 and pass 2.
Let us consider,
Pass1 60K
Pass 2 70K
Symbol table 20K
Common routines 30K

To load everything at once, we would require around 200K ofmemory. If only 130K is available, we cannot run our process. But pass 1 and pass 2 do not require to be in the memory at the same time.Therefore, we can define two overlays: First is Overlay A which is the symbol table,common routines, and pass 1, and the second is the overlay B which is the symbol table,common routines, and pass 2.We add an overlay driver (10K) and start with overlay A inmemory. When we finish pass 1, we jump to the overlay driver,which reads overlay B into memory, overwriting overlay A, and thentransfers control to pass 2. Overlay A needs only 110K, whereasoverlay B needs 120K.But in the case of dynamic loading, overlays do not require any specialsupport from the operating system.

4.2.2 Swapping

Process instructions and the data they operate on must be in memory to beexecuted. However, a process, or a portion of a process, can be **swapped**temporarily out of memory to a **backing store** and then brought back intomemory for continued execution (Figure 4.2). Swapping makes it possiblefor the total physical address space of all processes to exceed the

real physicalmemory of the system, thus increasing the degree of multiprogramming in a system.

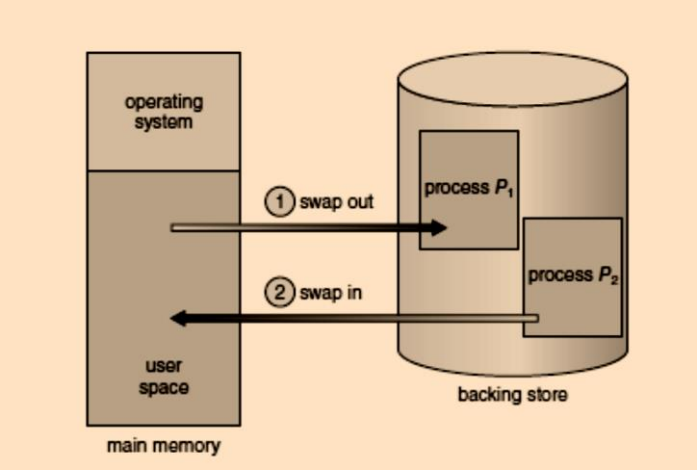


Fig. 4.2 Swapping of two processes using a disk as a backing store

Standard Swapping

Standard swapping involves moving entire processes between main memoryand a backing store. The backing store is commonly fast secondary storage.It must be large enough to accommodate whatever parts of processes need tobe stored and retrieved, and it must provide direct access to these memoryimages. When a process or part is swapped to the backing store, the datastructures associated with the process must be written to the backing store.For a multithreaded process, all per-thread data structures must be swappedas well. The operating system must also maintain metadata for processes thathave been swapped out, so they can be restored when they are swapped backinto memory.The advantage of standard swapping is that it allows physical memory tobe oversubscribed, so that the system can accommodate more processes thanthere is actual physical memory to store them. Idle or mostly idle processesare good candidates for swapping; any memory that has been allocated tothese inactive processes can then be dedicated to active processes. If an inactiveprocess that has been swapped out becomes active once again, it must then beswapped back in. This is illustrated in Figure 4.2

Swapping with Paging

Standard swapping was used in traditional UNIX systems, but it is generally nolonger used in contemporary operating systems, because the amount of timerequired to move entire processes between memory and the backing store isprohibitive. (An exception to this is Solaris, which still uses standard swapping,however only under dire circumstances when available memory is extremelylow.)Most systems, including Linux andWindows, now use a variation of swappingin which pages of a process—rather than an entire process—can beswapped. This strategy still allows physical memory to be oversubscribed butdoes not

incur the cost of swapping entire processes, as presumably only asmall number of pageswill be involved in swapping. In fact, the term **swapping**now generally refers to standard swapping, and **paging** refers to swappingwith paging. A**page out** operation moves a page from memory to the backingstore; the reverse process is known as a **page in**. Swapping with paging is illustratedin Figure 4.3 where a subset of pages for processes A and B are beingpaged-out and paged-in respectively.

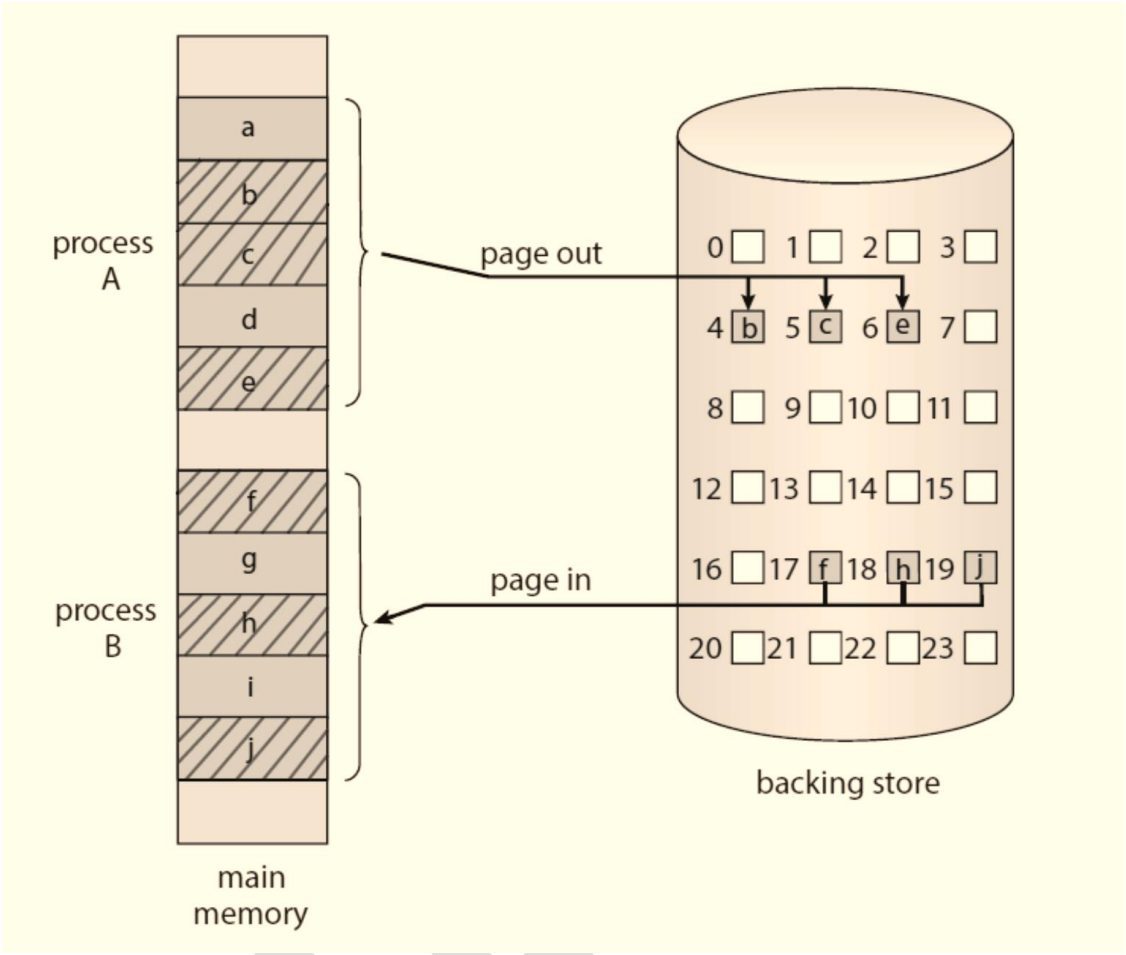


Fig. 4.3 Swapping with paging

Swapping on Mobile Systems

Most operating systems for PCs and servers support swapping pages. In contrast,mobile systems typically do not support swapping in any form. Mobiledevices generally use flash memory rather than more spacious hard disks fornonvolatile storage. The resulting space constraint is one reason why mobileoperating-system designers avoid swapping.Other reasons include the limitednumber of writes that flash memory can tolerate before it becomes unreliableand the poor throughput between main memory and flash memory in thesedevices.

Instead of using swapping, when freememory falls below a certain threshold,Apple's iOS asksapplications to voluntarily relinquish allocated memory.Read-only data (such as code) are removed from main memory and laterreloaded from flash memory if necessary. Data that have been modified (suchas the stack) are never removed. However, any applications that fail to free upsufficient memory may be terminated by the operating system.Android adopts a

strategy similar to that used by iOS. It may terminate aprocess if insufficient free memory is available. However, before terminating aprocess, Android writes its **application state** to flash memory so that it can bequickly restarted.Because of these restrictions, developers for mobile systems must carefullyallocate and release memory to ensure that their applications do not use toomuch memory or suffer from memory leaks.

4.2.3 Paging

Memory management discussed thus far has required the physical addressspace of a process to be contiguous. The concept of introduce **paging**, a memorymanagementscheme that permits a process's physical address space to be non-contiguous.Paging avoids external fragmentation and the associated need forcompaction, two problems that plague contiguous memory allocation. Becauseit offers numerous advantages, paging in its various forms is used in most operatingsystems, from those for large servers through those for mobile devices.Paging is implemented through cooperation between the operating system andthe computer hardware.

Basic Method

The basic method for implementing paging involves breaking physical memoryinto fixed-sized blocks called **frames** and breaking logical memory intoblocks of the same size called **pages**.When a process is to be executed, its pagesare loaded into any available memory frames fromtheir source (a file system orthe backing store). The backing store is divided into fixed-sized blocks that arethe same size as the memory frames or clusters of multiple frames. This rathersimple idea has great functionality and wide ramifications. For example, thelogical address space is now totally separate from the physical address space,so a process can have a logical 64-bit address space even though the system hasless than 264 bytes of physical memory.Every address generated by the CPU is divided into two parts: a **pagenumber** (**p**) and a **page offset** (**d**): (Figure 4.4)

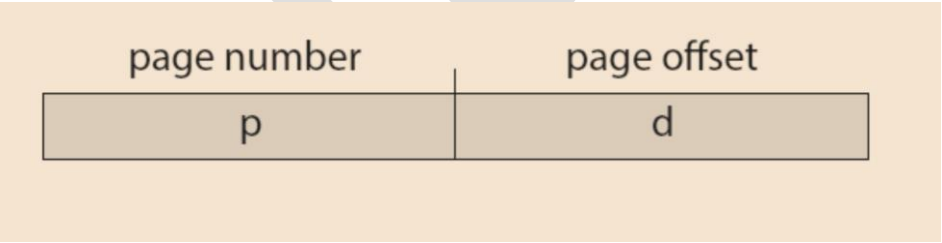


Fig 4.4address format in CPU

The page number is used as an index into a per-process **page table**. This is illustrated in Figure 4.5. The page table contains the base address of each frame in physical memory, and the offset is the location in the frame being referenced. Thus, the base address of the frame is

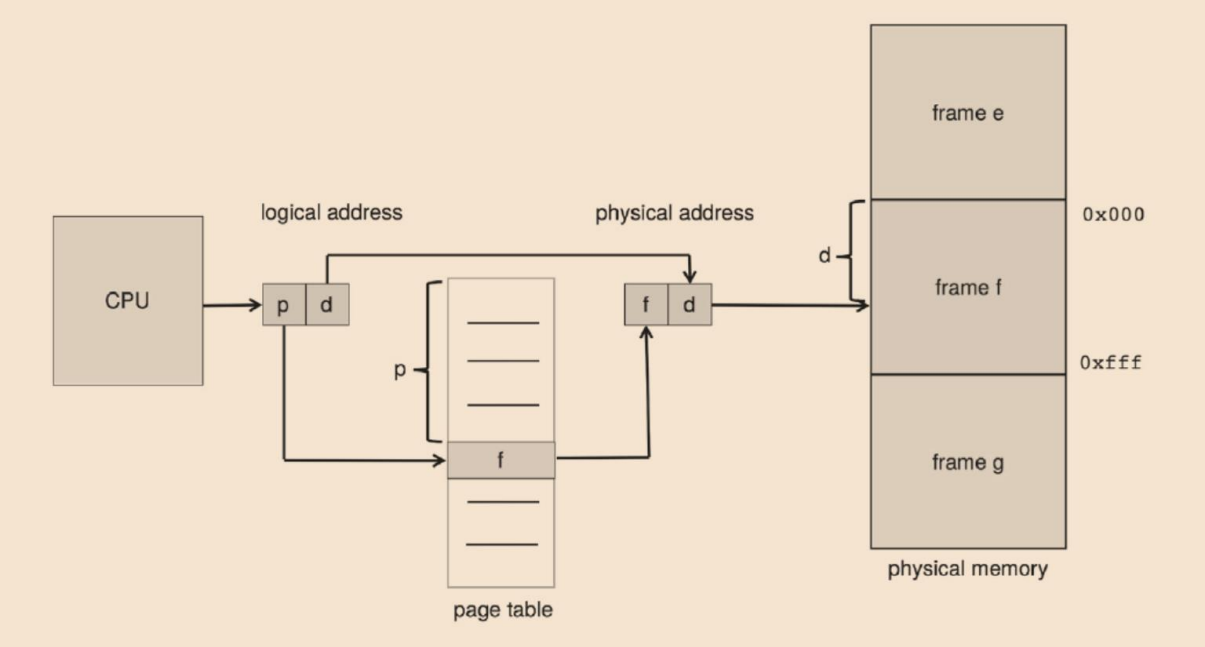


Fig. 4.5 Paging Hardware

combined with the page offset to define the physical memory address. The paging model of memory is shown in Figure 4.7.

The following outlines the steps taken by the MMU to translate a logicaladdress generated by the CPU to a physical address:

1. Extract the page number p and use it as an index into the page table.

2. Extract the corresponding frame number f fromthe page table.

3. Replace the page number p in the logical address with the frame numberf.

As the offset d does not change, it is not replaced, and the frame number andoffset now comprise the physical address.

The page size (like the frame size) is defined by the hardware. The sizeof a page is a power of 2, typically varying between 4 KB and 1 GB per page,depending on the computer architecture. The selection of a power of 2 as apage size makes the translation of a logical address into a page number andpage offset particularly easy. If the size of the logical address space is 2m, and apage size is 2n bytes, then the high-orderm−n bits of a logical address designatethe page number, and the n low-order bits designate the page offset. Thus, thelogical address is as follows (Figure 4.6):

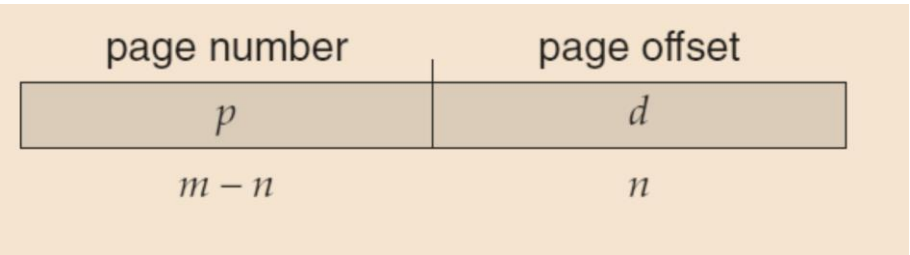


Fig. 4.6 Logical address

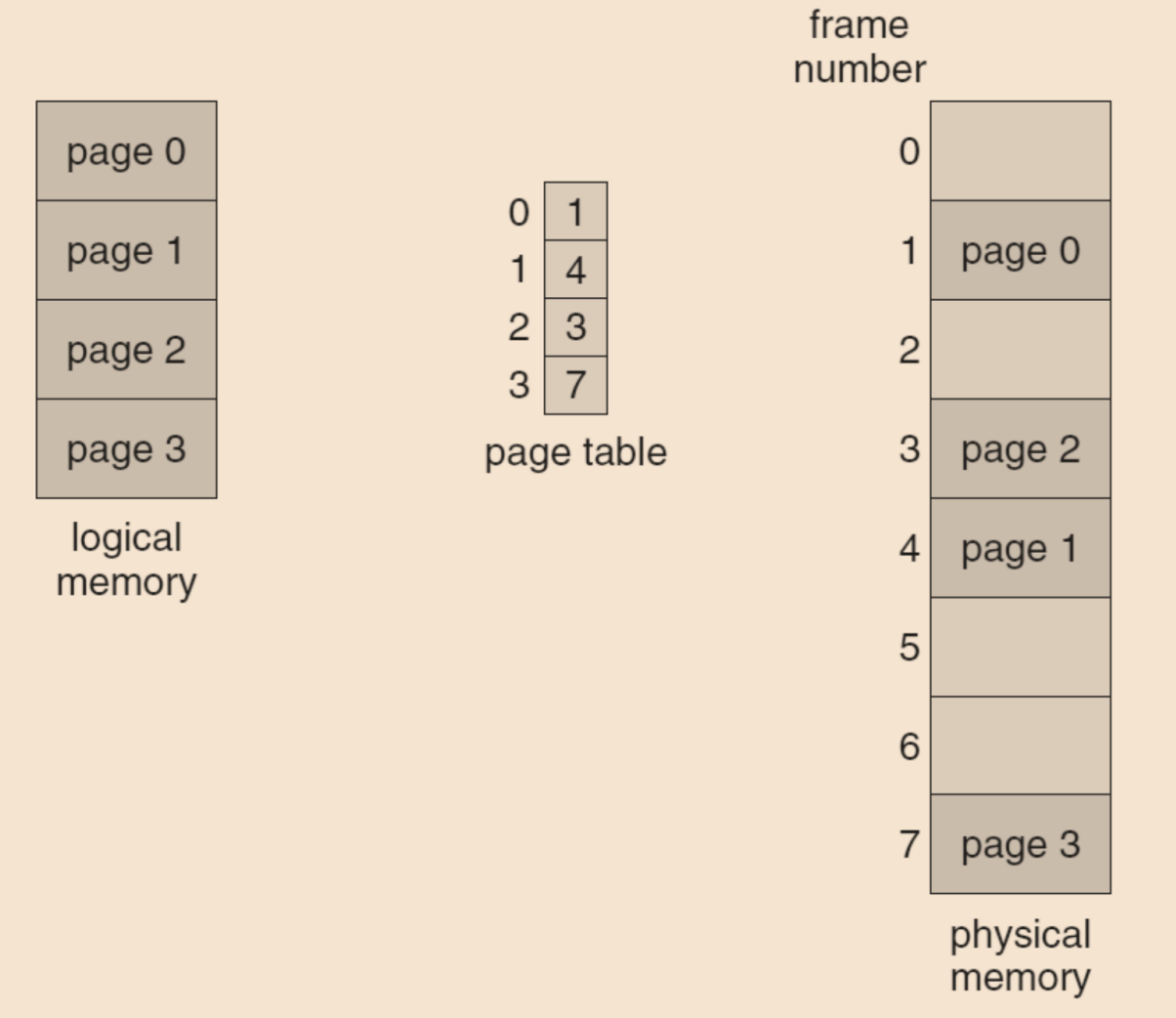


Fig. 4.7 Logical and Physical memory Paging model

where p is an index into the page table and d is the displacement within thepage.As a concrete (although minuscule) example, consider the memory inFigure 4.8. Here, in the logical address, n = 2 and m = 4. Using a page sizeof 4 bytes and a physical memory of 32 bytes (8 pages), we show how theprogrammer's view of memory can be mapped into physical memory. Logicaladdress 0 is page 0, offset 0. Indexing into the page table, we find that page 0is in frame 5.

Thus, logical address 0 maps to physical address 20 [= (5 × 4) +0].

Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) +3].

Logical address 4 is page 1, offset 0;

according to the page table, page 1 ismapped to frame6.

Thus, logical address 4 maps to physical address 24 [= (6× 4) + 0].

Logical address 13 maps to physical address 9.

You may have noticed that paging itself is a form of dynamic relocation.Every logical address is bound by the paging hardware to some physicaladdress. Using paging is similar to using a table of base (or relocation) registers,one for each frame of memory.When we use a paging scheme, we have no external fragmentation: anyfree frame can be allocated to a

process that needs it. However, we may havesome internal fragmentation. Notice that frames are allocated as units. If thememory requirements of a process do not happen to coincidewith page boundaries,the last frame allocated may not be completely full. For example, if pagesize is 2,048 bytes, a process of 72,766 byteswill need 35 pages plus 1,086 bytes.It will be allocated 36 frames, resulting in internal fragmentation of 2,048 −1,086 = 962 bytes. In the worst case, a process would need n pages plus 1 byte.Itwouldbe allocatedn + 1 frames, resulting in internal fragmentation of almostan entire frame.

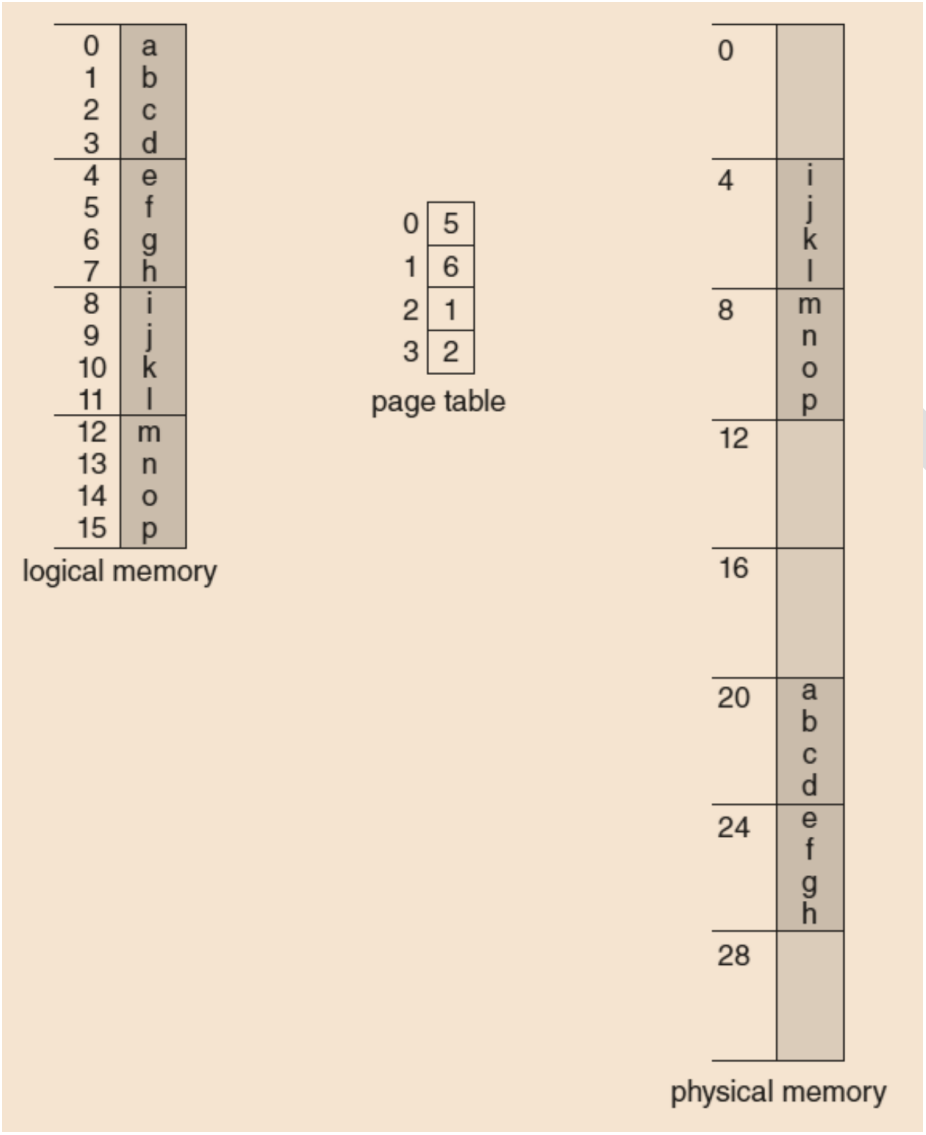


Fig. 4.8 Paging Example for a 32 byte memory with 4 byte pages.

If process size is independent of page size, we expect internal fragmentationto average one-half page per process. This consideration suggests thatsmall page sizes are desirable. However, overhead is involved in each pagetableentry, and this overhead is reduced as the size of the pages increases. Also,disk I/O is more efficient when the amount of data being

transferred is larger. Generally, page sizes have grown over time as processes, datasets, and main memory have become larger. Today, pages are typically either4 KB or 8 KB in size, and some systems support even larger page sizes. SomeCPUs and operating systems even support multiple page sizes. For instance, onx86-64 systems,Windows 10 supports page sizes of 4 KB and 2 MB. Linux alsosupports twopage sizes: a default page size (typically 4 KB) and an architecturedependentlarger page size called **huge pages**.Frequently, on a 32-bit CPU, each page-table entry is 4 bytes long, but thatsize can vary as well. A 32-bit entry can point to one of 232 physical pageframes. If the frame size is 4 KB (212), then a system with 4-byte entries canaddress 244 bytes (or 16 TB) of physical memory. We should note here that thesize of physical memory in a paged memory system is typically different fromthe maximum logical size of a process. As we further explore paging, we willintroduce other information that must be kept in the page-table entries. Thatinformation reduces the number of bits available to address page frames. Thus,a system with 32-bit page-table entries may address less physical memory thanthe possible maximum.When a process arrives in the system to be executed, its size, expressedin pages, is examined. Each page of the process needs one frame. Thus, if theprocess requires n pages, at least n frames must be available in memory. If nframes are available, they are allocated to this arriving process. The first pageof the process is loaded into one of the allocated frames, and the frame numberis put in the page table for this process. The next page is loaded into anotherframe, its frame number is put into the page table, and so on (Figure 4.9).An important aspect of paging is the clear separation between the programmer'sview of memory and the actual physical memory. The programmerviews memory as one single space, containing only this one program. In fact,the user program is scattered throughout physical memory which also holds other programs.

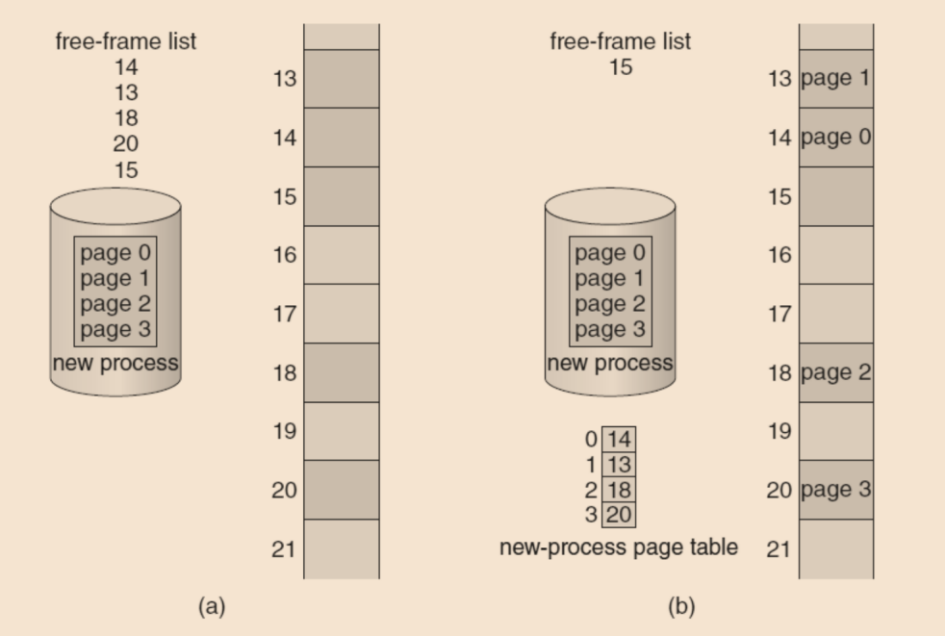


Fig. 4.9 Free Frames (a) before allocation and (b) after allocation

The difference between the programmer's view of memoryand the actual physical memory is reconciled by the address-translation hardware.The logical addresses are translated into physical addresses. This mappingis hidden fromthe programmer and is controlled by the

operating system.Notice that the user process bydefinition is unable to access memory it doesnot own. It has no way of addressing memory outside of its page table, and thetable includes only those pages that the process owns.Since the operating system is managing physical memory, it must be awareof the allocation details of physical memory—which frames are allocated,which frames are available, how many total frames there are, and so on. Thisinformation is generally kept in a single, system-wide data structure calleda **frame table**. The frame table has one entry for each physical page frame,indicating whether the latter is free or allocated and, if it is allocated, to whichpage of which process (or processes).
In addition, the operating system must be aware that user processes operatein user space, and all logical addresses must be mapped to produce physicaladdresses. If a user makes a system call (to do I/O, for example) and providesan address as a parameter (a buffer, for instance), that addressmust bemappedto produce the correct physical address. The operating system maintains a copyof the page table for each process, just as it maintains a copy of the instructioncounter and register contents. This copy is used to translate logical addresses tophysical addresseswhenever the operating systemmustmap a logical addressto a physical address manually. It is also used by the CPU dispatcher to definethe hardware page table when a process is to be allocated the CPU. Pagingtherefore increases the context-switch time.

4.2.4 Segmentation

A user program can be subdivided using segmentation, inwhich the program and its associated data are divided into anumber of**segments.** It is not required that all segments of allprograms be of the same length, although there is a maximumsegment length. As with paging, a logical address usingsegmentation consists of two parts, in this case a segment numberand an offset. Because of the use of unequal-size segments, segmentation is similar to dynamic partitioning. In segmentation, a program may occupy more than one partition, and these partitions need not be contiguous. Segmentation eliminates internal fragmentation but, like dynamic partitioning, it suffers from external fragmentation. However, because a process is broken up into a number of smaller pieces, the external fragmentation should be less. Whereas paging is invisible to the programmer, segmentation usually visible and is provided as a convenience for organizing programs and data. Another consequence of unequal-size segments is that there is no simple relationship between logical addresses and physical addresses. Segmentation scheme would make use of a segment table for each process and a list of free blocks of main memory. Each segment table entry would have to as in paging give the starting address in main memory of the corresponding segment. The entry should also provide the length of the segment, to assure that invalid addresses are not used. When a process enters the Running state, the address of its segment table is loaded into a special register used by the memory management hardware. Consider an address of n + m bits, where the leftmost n bits are the segment number and the rightmost m bits are the offset.
The following steps are needed for address translation:

- Extract the segment number as the leftmost n bits of the logical address.
- Use the segment number as an index into the process segment table to find the starting physical address of the segment.

- Compare the offset, expressed in the rightmost m bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid.
- The desired physical address is the sum of the starting physical address of the segment plus the offset.

Segmentation and paging can be combined to have a good result.

4.2.5 Virtual Memory

In the previous sections we have covered various memory-management strategies used incomputer systems. All these strategies have the same goali.e. to keep manyprocesses in memory simultaneously to allow multiprogramming. However,they tend to require that an entire process be in memory before it can execute.Virtual memory is a technique that allows the execution of processes thatare not completely in memory. One major advantage of this scheme is that programscan be larger than physical memory. Further, virtual memory abstractsmain memory into an extremely large, uniform array of storage, separatinglogical memory as viewed by the programmer from physical memory. Thistechnique frees programmers from the concerns of memory-storage limitations.Virtual memory also allows processes to share files and libraries, andto implement shared memory. In addition, it provides an efficient mechanismfor process creation. Virtual memory is not easy to implement, however, andmay substantially decrease performance if it is used carelessly. In this chapter,we provide a detailed overview of virtual memory, examine how it isimplemented, and explore its complexity and benefits.the instructions being executed must be in physical memory. The first approach to meeting this requirement is to placethe entire logical address space in physical memory. Dynamic linking can helpto ease this restriction, but it generally requires special precautions and extrawork by the programmer.The requirement that instructions must be in physical memory to be executedseems both necessary and reasonable; but it is also unfortunate, since itlimits the size of a program to the size of physical memory. In fact, an examinationof real programs shows us that, in many cases, the entire program is notneeded.

There cases where the entire program is needed, it may not all beneeded at the same time.The ability to execute a program that is only partially in memory wouldconfer many benefits:

- A program would no longer be constrained by the amount of physicalmemory that is available. Users would be able to write programs for anextremely large **virtual** address space, simplifying the programming task.
- Because each program could take less physical memory, more programscouldbe runat the same time,witha correspondingincrease inCPU utilizationand throughput but with no increase in response time or turnaroundtime.
- Less I/O would be needed to load or swap portions of programs intomemory, so each program would run faster.Thus, running a program that is not entirely inmemorywould benefit both thesystem and its users.

Virtual memory involves the separation of logical memory as perceivedby developers from physical memory. This separation allows an extremelylarge virtual memory to be provided for programmers when only a smallerphysical memory is available (Figure 4.10). Virtual memory makes the task ofprogramming much easier because the programmer no longer needs to worryabout the amount of physical memory available; she can concentrate insteadon programming the problem that is to be solved.

The **virtual address space** of a process refers to the logical (or virtual) viewof how a process is stored in memory. Typically, this view is that a processbegins at a certain logical address—say, address 0—and exists in contiguousmemory, as shown in Figure 4.11. Physical memory is organized in page frames and that the physical pageframes assigned to a process may not be contiguous. It is up to the memorymanagement unit (MMU) to map logical pages to physical page frames inmemory.

Note in Figure 4.11 that we allow the heap to grow upward in memory asit is used for dynamic memory allocation. Similarly, we allow for the stack togrow downward in memory through successive function calls. The large blankspace (or hole) between the heap and the stack is part of the virtual addressspace but will require actual physical pages only if the heap or stack grows.Virtual address spaces that include holes are known as **sparse** address spaces. Using a sparse address space is beneficial because the holes can be filled as thestack or heap segments grow or if we wish to dynamically link libraries (orpossibly other shared objects) during program execution.

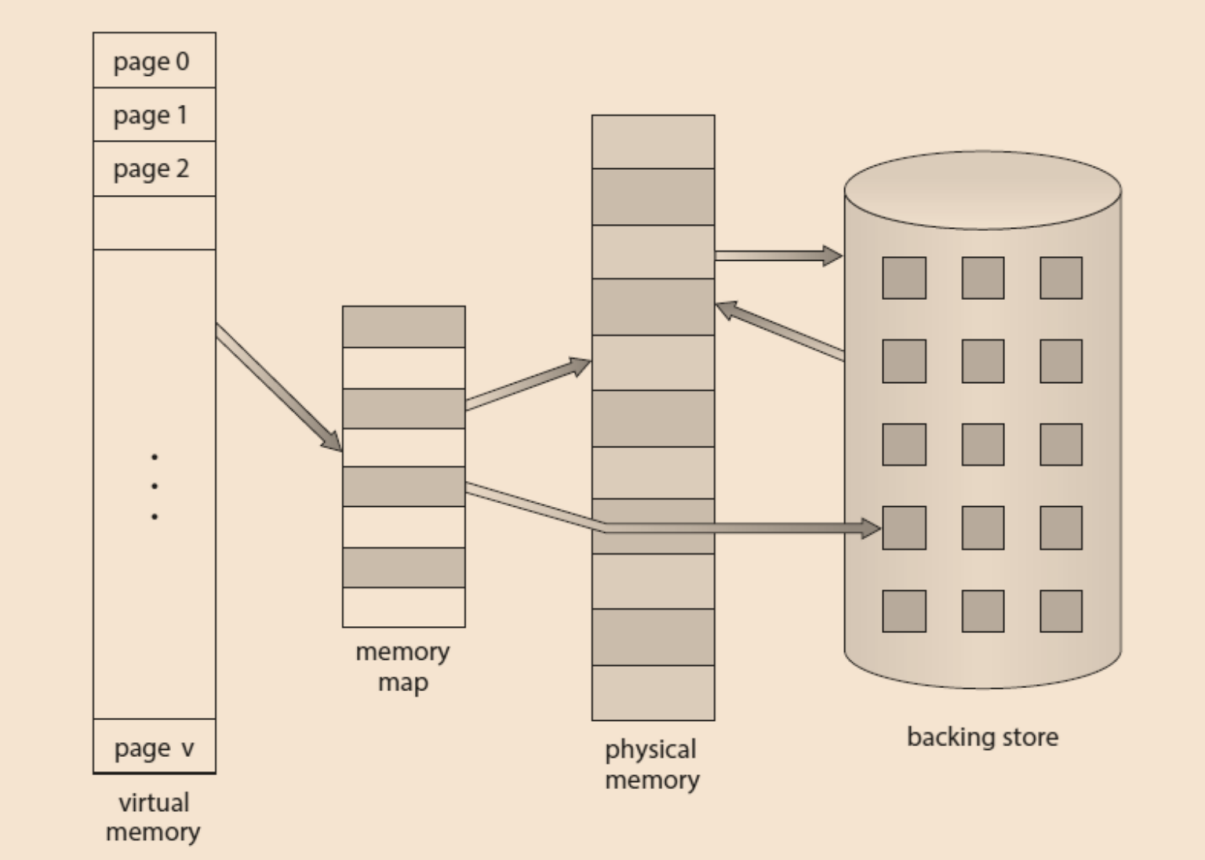


Fig. 4.10 Large size of Virtual memory compared to physical memory

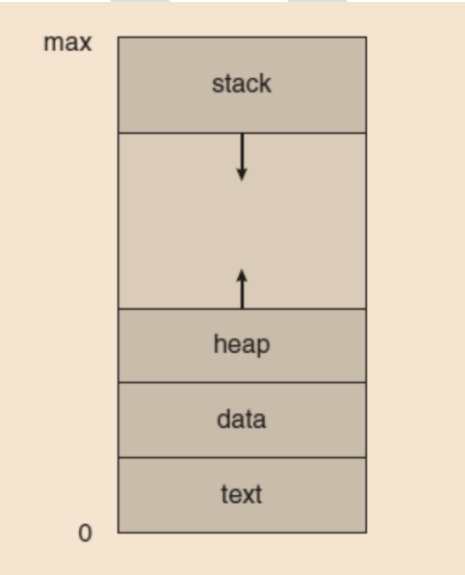


Fig. 4.11 Diagram representing Virtual address space of a process in memory

In addition to separating logical memory from physical memory, virtualmemory allows files and memory to be shared by two or more processesthrough page sharing. This results into the following benefits:

- System libraries such as the standard C library can be shared by severalprocesses through mapping of the shared object into a virtual addressspace. Although each process considers the libraries to be part of its virtualaddress space, the actual pages where the libraries reside in physicalmemory are shared by all the processes (Figure 4.12). Typically, a library ismapped read-only into the space of each process that is linked with it.
- Similarly, processes can share memory. Twoor more processes can communicate through the use of shared memory.Virtual memory allows one process to create a region of memory that it canshare with another process. Processes sharing this region consider it partof their virtual address space, yet the actual physical pages of memory areshared, much as is illustrated in Figure 4.12.
- Pages can be shared during process creation with the fork() system call,thus speeding up process creation.

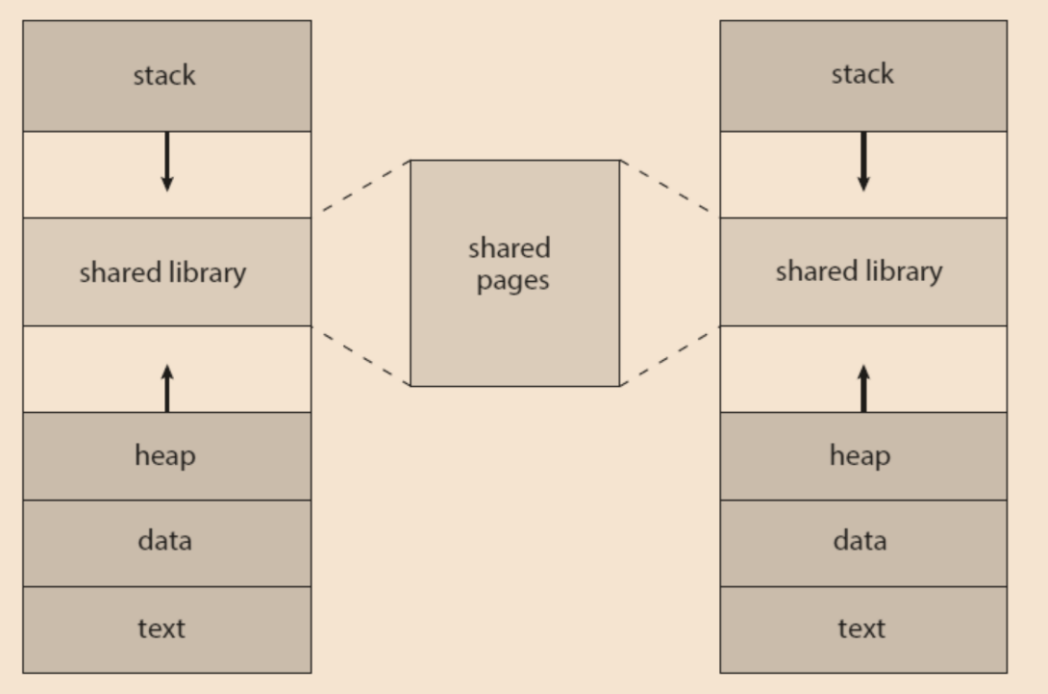


Fig 4.12 Shared library using virtual memory

4.2.5.1 Overlays

In the previous years of the computer era, people were first encountered with programs that were too huge to fit in the available memory. The solution usually adopted was to split the program into pieces, called **overlays**. Overlay 0 would start running first. When it was done, it would call another overlay. Some overlay systems were highly complex, allowing multiple overlays in memory at once. Theoverlays were kept on the disk and swapped in and out of memory by the operating system,dynamically, as needed.Although the actual work of swapping overlays in and out was done by the system, the decision ofhow to split the program into pieces had to be done by the programmer. Splitting up largeprograms into small, modular pieces was time consuming and boring. It did not take long beforesomeone thought of a way to turn the whole job over to the computer.

The method that was devised has come to be known as **virtual memory**.The basic idea behind virtual memory is that the combined size of the program, data, and stackmay exceed the amount of physical memory available for it. The operating system keeps thoseparts of the program currently in use in main memory, and the rest on the disk. For example, a512-MB program can run on a 256-MB machine by carefully choosing which 256 MB to keep inmemory at each instant, with pieces of the program being swapped between disk and memory asneeded.Virtual memory can also work in a multiprogramming system, with bits and pieces of manyprograms in memory at once. While a program is waiting for part of itself to be brought in,it iswaiting for I/O and cannot run, so the CPU can be given to another process, the same way as inany other multiprogramming system.

4.2.5.2 Demand Paging

Consider how an executable program might be loaded fromsecondary storageinto memory. One option is to load the entire program in physical memoryat program execution time. However, a problem with this approach is thatstarts with a list of available options from which the user is to select. Loadingthe entire programintomemory results in loading the executable code for alloptions, regardless of whether an option is ultimately selected by theuser or not.

An alternative strategy is to load pages only as they are needed. This techniqueis known as **demand paging** and is commonly used in virtual memorysystems. With demand-paged virtual memory, pages are loaded only whenthey are **demanded** during program execution. Pages that are never accessedare thus never loaded into physical memory. Ademand-paging system is like a paging system with swapping, where processes reside insecondarymemory. Demand paging explainsthe primary benefit of virtual memory by loading only the portionsof programs that are needed so that the memory is used more efficiently.

Fundamental Concepts of Demand Paging

The fundamental concept behind demand paging, is to load a page inmemory only when it is needed. As a result,while a process is executing, somepageswill be in memory, and some will

be in secondary storage. Thus, we needsome form of hardware support to distinguish between the two. The valid-invalid bit schemecan be used for this purpose. Thistime, however, when the bit is set to "valid," the associated page is both legaland in memory. If the bit is set to "invalid," the page either is not valid (thatis, not in the logical address space of the process) or is valid but is currently insecondary storage. The page-table entry for a page that is brought into memoryis set as usual, but the page-table entry for a page that is not currently inmemory is simply marked invalid. This situation is depicted in Figure 4.13.

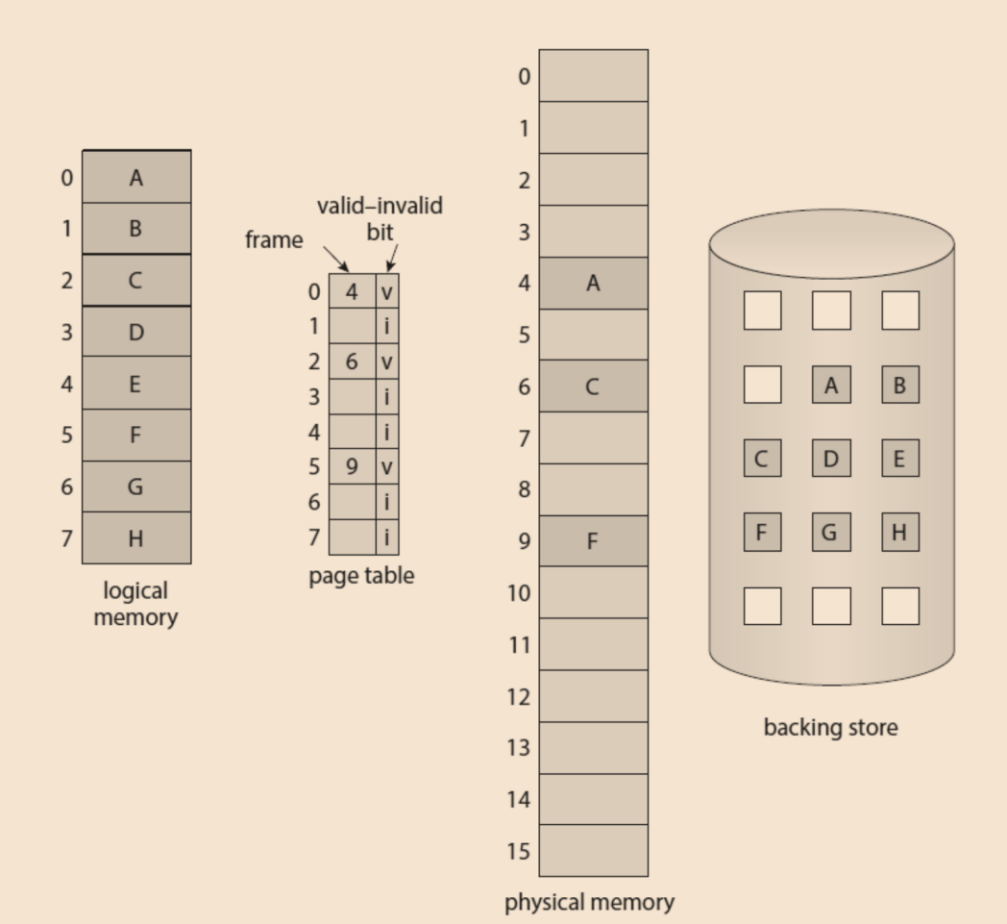


Fig. 4.13 Diagram representing the page table when some pages are not in main memory.

Here, there is one point to be noticed, that marking a page invalid will have no effect if the process neverattempts to access that page.But what will be the situation if the process tries to access a page that was not broughtinto memory?Access to a page marked invalid causes a **page fault**. The paginghardware, in translating the address through the page table,will notice that theinvalid bit is set, causing a trap to the operating system. This trap is the resultof the operating system's failure to bring the desired page into memory. Theprocedure for handling this page fault is straightforward (Figure 4.14):

1. We check an internal table which is usually kept with the process control blockfor this process to determine whether the reference was a valid or aninvalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid butwe have not yet brought in that page, we now page it in.

3. We find a free frame by taking one from the free-frame list.
4. We schedule a secondary storage operation to read the desired page into the newly allocated frame.
5. When the storage read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The processcan now access the page as though it had always been in memory.

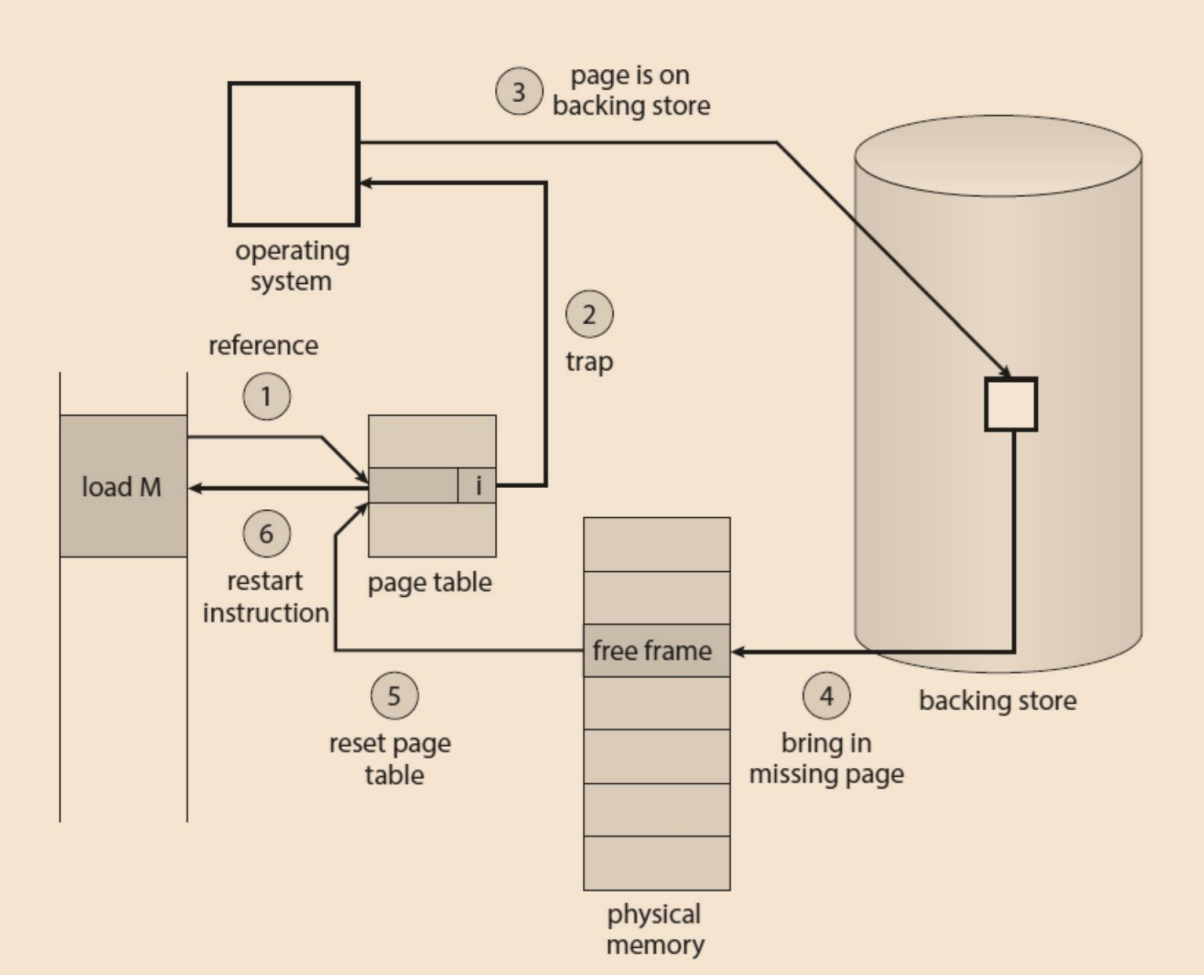


Fig. 4.14 Steps in handling a page fault.

In the intense case, we can start executing a process with nopages inmemory. When the operating system sets the instruction pointer to the firstinstruction of the process, which is on a non-memory-resident page, the processimmediately faults for the page. After this page is brought into memory, theprocess continues to execute, faulting as necessary until every page that itneeds is in memory. At that point, it can execute with no more faults. Thisscheme is **pure demand paging**: never bring a page into memory until it isrequired.Ideally, some programs could access several new pages of memorywith each instruction execution where one page for the instruction and many fordata. This possibly causes multiple page faults per

instruction. This situationwould result in unacceptable system performance. Fortunately, analysis ofrunning processes shows that this behaviour is exceedingly unlikely.

The hardware to support demand paging is the same as the hardware forpaging and swapping:

- **Page table**. This table can mark an entry invalid through avalid–invalid bit or a special value of protection bits.
- **Secondary memory**. This memory holds those pages that are not presentin main memory. The secondary memory is usually a high-speed disk orNVM device. It is known as the swap device, and the section of storageused for this purpose is known as **swap space**.

A crucial requirement for demand paging is the ability to restart anyinstruction after a page fault. Because we save the state (registers, conditioncode, instruction counter) of the interrupted process when the page faultoccurs, we must be able to restart the process in the same place andstate, except that the desired page is now in memory and is accessible. In mostcases, this requirement is easy to meet. Apage fault may occur at any memoryreference. If the page fault occurs on the instruction fetch, we can restart byfetching the instruction again. If a page fault occurs while we are fetching anoperand, we must fetch and decode the instruction again and then fetch theoperand.

As a worst-case example, consider a three-address instruction such as ADDthe content of Ato B, placing the result in C. These are the steps to execute thisinstruction:

1. Fetch and decode the instruction (ADD).
2. Fetch A.
3. Fetch B.
4. Add Aand B.
5. Store the sum in C.

If we fault when we try to store in C as C is in a page not currentlyin memory, we will have to get the desired page, bring it in, correct thepage table, and restart the instruction. The restart will require fetching theinstruction again, decoding it again, fetching the two operands again, andthen adding again. However, there is not much-repeated work (less than onecomplete instruction), and the repetition is necessary only when a page faultoccurs.The major difficulty arises when one instruction may modify several differentlocations. For example, consider the IBM System 360/370 MVC (movecharacter) instruction, which can move up to 256 bytes from one location toanother (possibly overlapping) location. If either block (source or destination)straddles a page boundary, a page fault might occur after the move is partiallydone. In addition, if the source and destination blocks overlap, the sourcelock may have been modified, in which case we cannot simply restart theinstruction.This problem can be solved in two different ways. In one solution, themicrocode computes and attempts to access both ends of both blocks. If a pagefault is going to occur, it will happen at this step, before anything is modified.The move can then take place; we know that no page fault can occur, since allthe relevant pages are in memory.

The other solution uses temporary registersto hold the values of overwritten locations. If there is a page fault, all the oldvalues arewritten back into memory before the trap occurs. This action restoresmemory to its state before the instruction was started, so that the instructioncan be repeated.This is by no means the only architectural problem resulting from addingpaging to an existing architecture to allow demand paging, but it illustratessome of the difficulties involved. Paging is added between the CPU and thememory in a computer system. It should be entirely transparent to a process.

Thus, people often assume that paging can be added to any system. Althoughthis assumption is true for a non-demand-paging environment, where a pagefault represents a fatal error, it is not true where a page fault means only thatan additional page must be brought into memory and the process restarted.

4.2.5.3 Performance of Demand Paging

Demand paging can significantly affect the performance of a computer system.Let us compute the **effective access time** for a demand-paged memory.Assume the memory-access time, denoted ma, is 10 nanoseconds. If we have no page faults, the effective access time is equal to the memoryaccess time. If, however, a page fault occurs, we must first read the relevantpage from secondary storage and then access the desired word.Let p be the probability of a page fault ($0 \leq p \leq 1$). We would expect p tobe close to zero—that is, we would expect to have only a few page faults. Theeffective access time is theneffective 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. A page fault causes the following sequence tooccur:

1. Trap to the operating system.
2. Save the registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page referencewas legal and determine the location of thepage in secondary storage.
5. Issue a read from the storage to a free frame:
a. Wait in a queue until the read request is serviced.
b. Wait for the device seek and/or latency time.
c. Begin the transfer of the page to a free frame.
6. While waiting, allocate the CPU core to some other process.
7. Receive an interrupt from the storage I/O subsystem (I/O completed).
8. Save the registers and process state for the other process (if step 6 isexecuted).
9. Determine that the interrupt was from the secondary storage device.
10. Correct the page table and other tables to show that the desired page isnow in memory.
11. Wait for the CPU core to be allocated to this process again.
12. Restore the registers, process state, and new page table, and then resumethe interrupted instruction.

Not all these steps are necessary in every case. For example,we are assumingthat, in step 6, the CPU is allocated to another process while the I/O occurs.This arrangement allows

multiprogramming to maintain CPU utilization butrequires additional time to resume the page-fault service routine when the I/Otransfer is complete.

In any case, there are three major task components of the page-fault servicetime:
1. Service the page-fault interrupt.
2. Read in the page.
3. Restart the process.

The first and third tasks can be reduced, with careful coding, to severalhundred instructions. These tasks may take from 1 to 100 microseconds each.Let us consider the case of HDDs being used as the paging device. The pageswitchtime will probably be close to 8 milliseconds. (A typical hard disk hasan average latency of 3 milliseconds, a seek of 5 milliseconds, and a transfertime of 0.05 milliseconds. Thus, the total paging time is about 8 milliseconds,including hardware and software time.) Remember also that we are looking atonly the device-service time. If a queue of processes is waiting for the device,we have to add queuing time as we wait for the paging device to be free toservice our request, increasing even more the time to page in.With an average page-fault service time of 8 milliseconds and a memoryaccesstime of 200 nanoseconds,
the effective access time in nanoseconds iseffective access time = $(1 - p) \times (200) + p$ (8 milliseconds)
$= (1 - p) \times 200 + p \times 8{,}000{,}000$
$= 200 + 7{,}999{,}800 \times p.$
We see, then, that the effective access time is directly proportional to the**page-fault rate**. If one access out of 1,000 causes a page fault, the effective accesstime is 8.2 microseconds. The computer will be slowed down by a factor of 40because of demand paging. If we want performance degradation to be less than10 percent, we need to keep the probability of page faults at the following level:

$220 > 200 + 7{,}999{,}800 \times p,$
$20 > 7{,}999{,}800 \times p,$
$p < 0.0000025.$

That is, to keep the slowdown due to paging at a reasonable level,we can allowfewer that one memory access out of 399,990 to page-fault. In sum, it is importantto keep the page-fault rate low in a demand-paging system. Otherwise,the effective access time increases, slowing process execution dramatically.An additional aspect of demand paging is the handling and overall use ofswap space. I/O to swap space is generally faster than that to the file system. Itis faster because swap space is allocated in much larger blocks, and file lookupsand indirect allocation methods are not used. One option for thesystem to gain better paging throughput is by copying an entire file image intothe swap space at process startup and then performing demand paging fromthe swap space. The obvious disadvantage of this approach is the copying ofthe file image at program start-up. A second option which is practiced byseveral operating systems, including Linux andWindowsis to demand-pagefrom the file system initially but to write the pages to swap space as they arereplaced. This approach will ensure

that only needed pages are read from thefile system but that all subsequent paging is done from swap space.Some systems attempt to limit the amount of swap space used throughdemand paging of binary executable files. Demand pages for such files arebrought directly from the file system. However, when page replacement iscalled for, these frames can simply be overwritten, and the pages can be read in from the file system again if needed.Using this approach, the file system itself serves as the backing store. However,swap space must still be used for pages not associated with a file also known as**anonymous memory**; these pages include the stack and heap for a process.This method appears to be a good compromise and is used in several systems,including Linux and BSD UNIX.Mobile operating systems typically do notsupport swapping. Instead, these systems demand-page from the file systemand reclaim read-only pages from applications if memorybecomes constrained. Such data can be demand-paged from the file system ifit is later needed.

4.2.5.4 Page Replacement Algorithms

Page Replacement

In our earlier discussion of the page-fault rate, we assumed that each pagefaults at most once,when it is first referenced. This representation is not strictlyaccurate, however. If a process of ten pages uses only half of them, thendemand paging saves the I/O necessary to load the five pages that are neverused. We could also increase our degree of multiprogramming by runningtwice as many processes. Thus, if we had forty frames, we could run eightprocesses, rather than the four that could run if each required ten frames (fiveof which were never used).If we increase our degree of multiprogramming, we are **over-allocating**memory. If we run six processes, each of which is ten pages in size but actually uses only five pages, we have higher CPU utilization and throughput, withten frames to spare. It is possible, however, that each of these processes, fora particular data set, may suddenly try to use all ten of its pages, resulting in aneed for sixty frames when only forty are available.Further, consider that system memory is not used only for holding programpages. Buffers for I/O also consume a considerable amount of memory. This usecan increase the strain on memory-placement algorithms. Deciding how muchmemory to allocate to I/O and how much to program pages is a significantchallenge. Some systems allocate a fixed percentage of memory for I/O buffers,whereas others allow both processes and the I/O subsystem to compete for allsystem memory. Section 14.6 discusses the integrated relationship between I/Obuffers and virtual memory techniques.Over-allocation of memory manifests itself as follows. While a process isexecuting, a page fault occurs. The operating system determines where thedesired page is residing on secondary storage but then finds that there are**no** free frames on the free-frame list; all memory is in use. This situation isillustrated in Figure 4.15, where the fact that there are no free frames is depictedby a question mark.The operating system has several options at this point. It could terminatethe process. However, demand paging is the operating system's attempt toimprove the computer system's utilization and throughput. Users should notbe aware that their processes are running on a paged system—paging shouldbe logically transparent to the user. So this option is not the best

choice.The operating system could instead use standard swapping and swapout a process, freeing all its frames and reducing the level of multiprogramming.The standard swapping is no longerused by most operating systems due to the overhead of copying entire processesbetween memory and swap space. Most operating systems now combineswapping pageswith **page replacement**, a techniquewe describe indetailin the remainder of this section.

Basic Page Replacement

Page replacement takes the following approach. If no frame is free,we find onethat is not currently being used and free it. We can free a frame by writing its

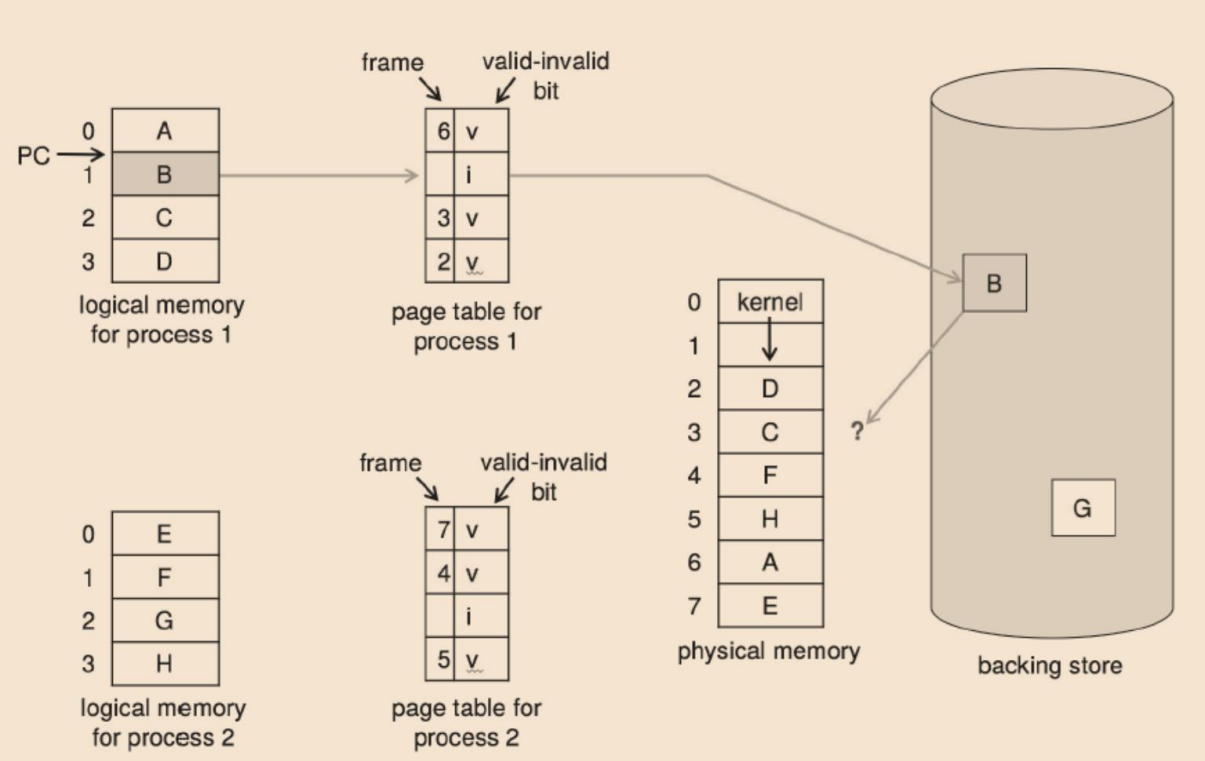


Fig. 4.15 Requirement for the page replacement

contents to swap space and changing the page table toindicate that the page is no longer in memory (Figure 4.16). We can now usethe freed frame to hold the page for which the process faulted.We modify thepage-fault service routine to include page replacement:

1. Find the location of the desired page on secondary storage.

2. Find a free frame:

a. If there is a free frame, use it.

b. If there is no free frame, use a page-replacement algorithm to selecta **victim frame**.

c. Write the victim frame to secondary storage (if necessary); changethe page and frame tables accordingly.

3. Read the desired page into the newly freed frame; change the page andframe tables.

4. Continue the process from where the page fault occurred.

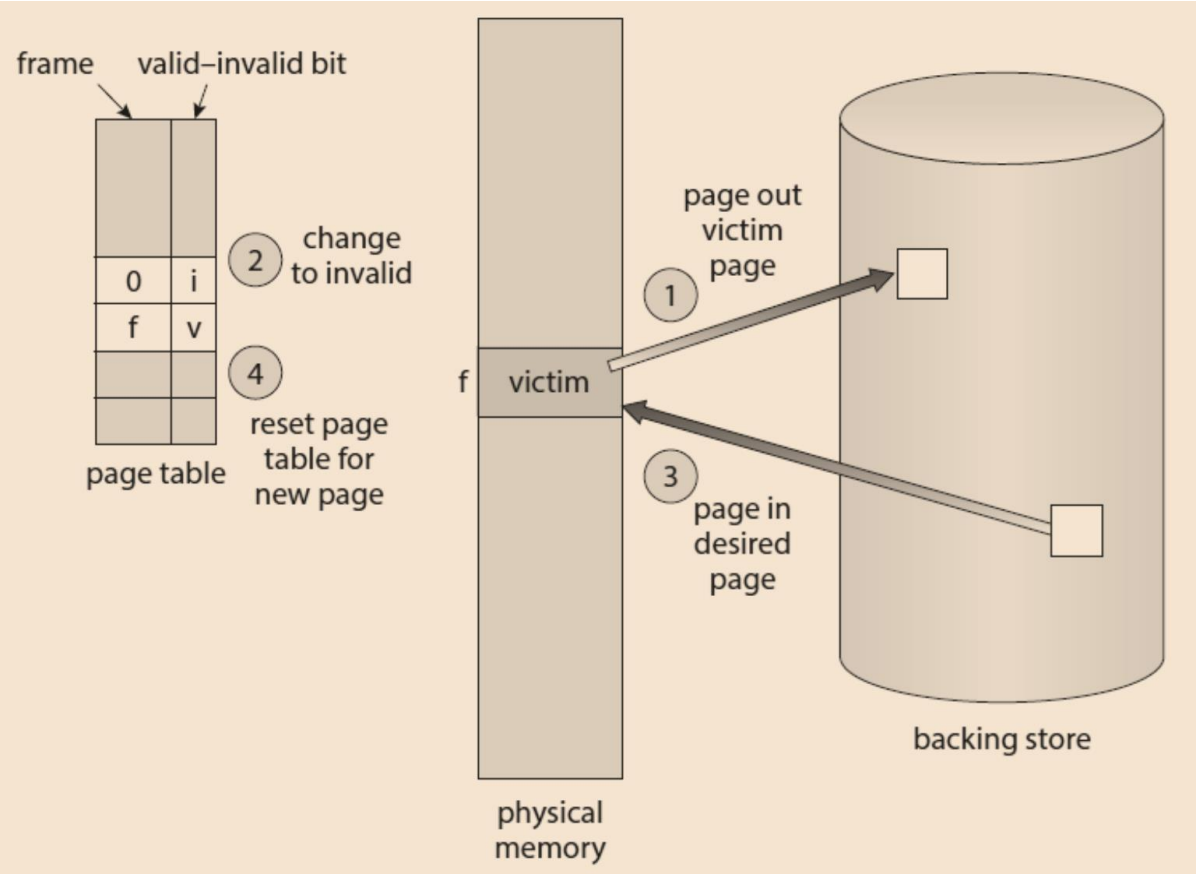


Fig. 4.16 Page Replacement

Notice that, if no frames are free, two-page transfers (one for the page-outand one for the page-in) are required. This situation effectively doubles thepage-fault service time and increases the effective access time accordingly.We can reduce this overhead by using a modify bit (or dirty bit). When thisscheme is used, each page or frame has a modify bit associated with it in thehardware. The modify bit for a page is set by the hardware whenever any bytein the page is written into, indicating that the page has been modified. Whenwe select a page for replacement, we examine its modify bit. If the bit is setwe know that the page has been modified since it was read in from secondarystorage. In this case, we must write the page to storage. If the modify bit is notset, however, the page has notbeen modified since itwas read intomemory. Inthis case,we need notwrite thememory page to storage: it is already there. Thistechnique also applies to read-only pages like the pages of binary code.Such pages cannot be modified; thus, they may be discarded when desired.This scheme can significantly reduce the time required to service a page fault,since it reduces I/O time by one-half ifthe page has not been modified.Page replacement is basic to demand paging. It completes the separationbetween logicalmemory and physicalmemory.With this mechanism, an enormousvirtual memory can be provided for programmers on a smaller physicalmemory.With no demand paging, logical addresses are mapped into physicaladdresses, and the two sets of addresses can be different. All the pages of aprocess still must be in physical memory, however. With demand paging, thesize of the logical address space is no longer constrained by physical memory.If we have a process of twenty

pages, we can execute it in ten frames simply byusing demand paging and using a replacement algorithm to find a free framewhenever necessary. If a page that has been modified is to be replaced, itscontents are copied to secondary storage. A later reference to that page willcause a page fault. At that time, the page will be brought back into memory,perhaps replacing some other page in the process.We must solve two major problems to implement demand paging:we mustdevelop a **frame-allocation algorithm** and a **page-replacement algorithm**.That is, if we have multiple processes in memory, we must decide how manyframes to allocate to each process; and when page replacement is required,we must select the frames that are to be replaced. Designing appropriate algorithmsto solve these problems is an important task, because secondary storageI/O is so expensive. Even slight improvements in demand-paging methodsyield large gains in system performance.There are many different page-replacement algorithms. Every operatingsystem probably has its own replacement scheme. How do we select a particularreplacement algorithm? In general, we want the one with the lowestpage-fault rate.We evaluate an algorithm by running it on a particular string of memoryreferences and computing the number of page faults. The string of memoryreferences is called a **reference string**. We can generate reference strings artificially(by using a random-number generator, for example), or we can tracea given system and record the address of each memory reference. The latterchoice produces a large number of data (on the order of 1 million addressesper second). To reduce the number of data, we use two facts.First, for a given page size (and the page size is generally fixed by the hardwareor system), we need to consider only the page number, rather than theentire address. Second, if we have a reference to a page p, then any referencesto page p that immediatelyfollow will never cause a page fault. Page p willbe in memory after the first reference, so the immediately following referenceswill not fault.

For example, ifwe trace a particular process, we might record the followingaddress sequence:
0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105
At 100 bytes per page, this sequence is reduced to the following referencestring:
1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

To determine the number of page faults for a particular referencestring andpage-replacement algorithm, we also need to know the number of page framesavailable. Obviously, as the number of frames available increases, the numberof page faults decreases. For the reference string considered previously, forexample, if we had three or more frames, we would have only three faults, where one fault for the first reference to each page. In contrast, with only one frameavailable, we would have a replacement with every reference, resulting ineleven faults. In general, we expect a curve such as that in Figure 4.17. As thenumber of frames increases, the number of page faults drops to some minimallevel. Of course, adding physical memory increases the number of frames.We next illustrate several page-replacement algorithms. In doing so, weuse the reference string,

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

for a memory with three frames.

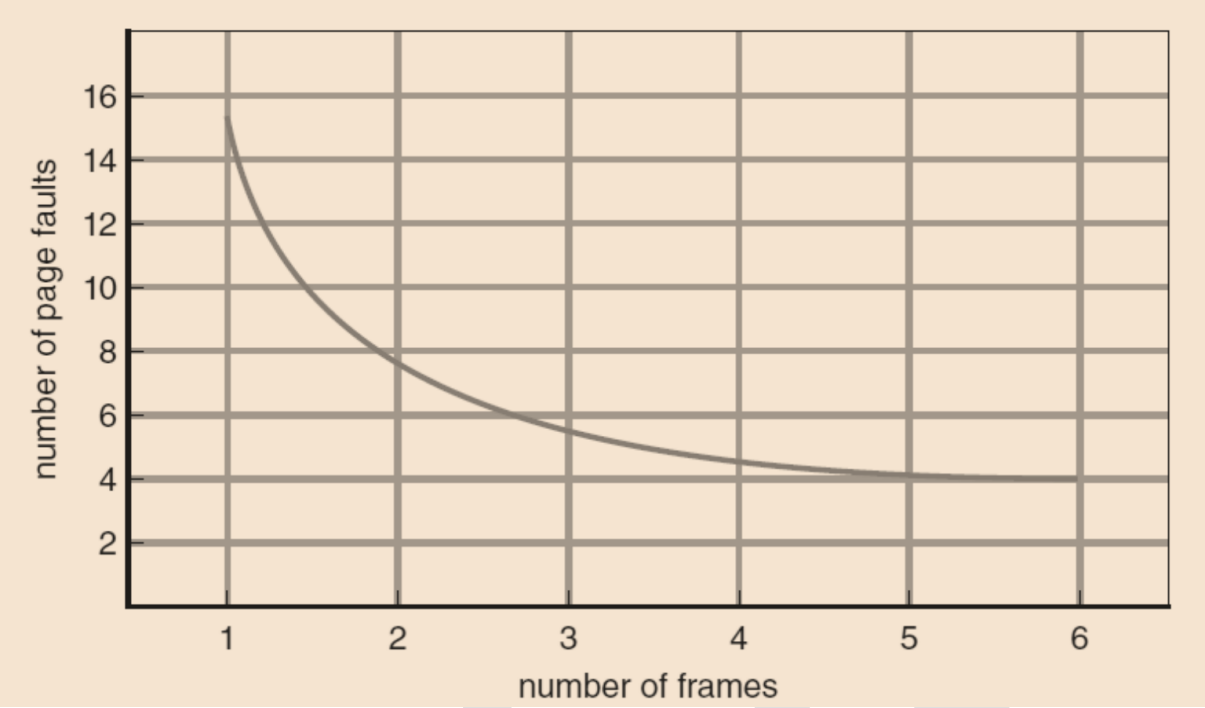


Fig. 4.17 Graph plotted for curve representing faults Vs number of frames.

FIFO Page Replacement

The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.AFIFO replacement algorithm associates with each page the time whenthat page was brought into memory.When a page must be replaced, the oldestpage is chosen. Notice that it is not strictly necessary to record the time when a page is brought in.We can create a FIFO queue to hold all pages inmemory.Wereplace the page at the head of the queue.When a page is brought into memory,we insert it at the tail of the queue.For our example reference string, our three frames are initially empty. Thefirst three references (7, 0, 1) cause page faults and are brought into these emptyframes. The next reference (2) replaces page 7, because page 7 was brought infirst. Since 0 is the next reference and 0 is already in memory, we have no faultfor this reference. The first reference to 3 results in replacement of page 0, sinceit is now first in line. Because of this replacement, the next reference, to 0, willfault. Page 1 is then replaced by page 0. This process continues as shown inFigure 4.18. Every time a fault occurs, we show which pages are in our threeframes. There are fifteen faults altogether.The FIFO page-replacement algorithm is easy to understand and program.However, its performance is not always good. On the one hand, the pagereplaced may be an initialization module that was used a long time ago and isno longer needed. On the other hand, it could contain a heavily used variablethat was initialized early and is in constant use.Notice that, even if we select for

replacement a page that is in active use,everything still works correctly. After we replace an active page with a new one, a fault occurs almost immediately to retrieve the active page.

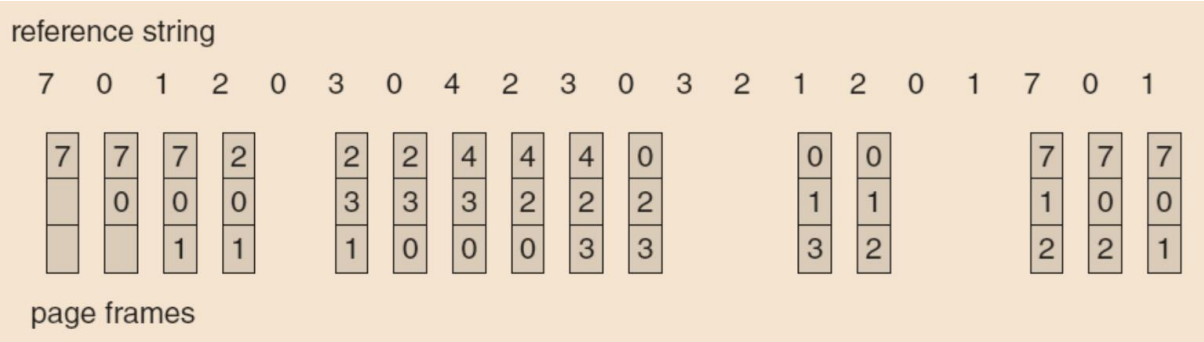


Fig. 4.18 FIFO page replacement algorithm.

Some otherpage must be replaced to bring the active page back into memory. Thus, a badreplacement choice increases the page-fault rate and slows process execution.It does not, however, cause incorrect execution.To illustrate the problems that are possible with a FIFO page-replacementalgorithm, consider the following reference string:

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

Figure 4.19 shows the curve of page faults for this reference string versus thenumber of available frames. Notice that the number of faults for four frames(ten) is **greater** than the number of faults for three frames (nine)! This mostunexpected result is known as **Belady's anomaly**: for some page-replacementalgorithms, the page-fault rate may increaseas the number of allocated framesincreases. We would expect that giving more memory to a process wouldimprove its performance. In some early research, investigators noticed that this assumption was not always true. Belady's anomaly was discovered as a result.

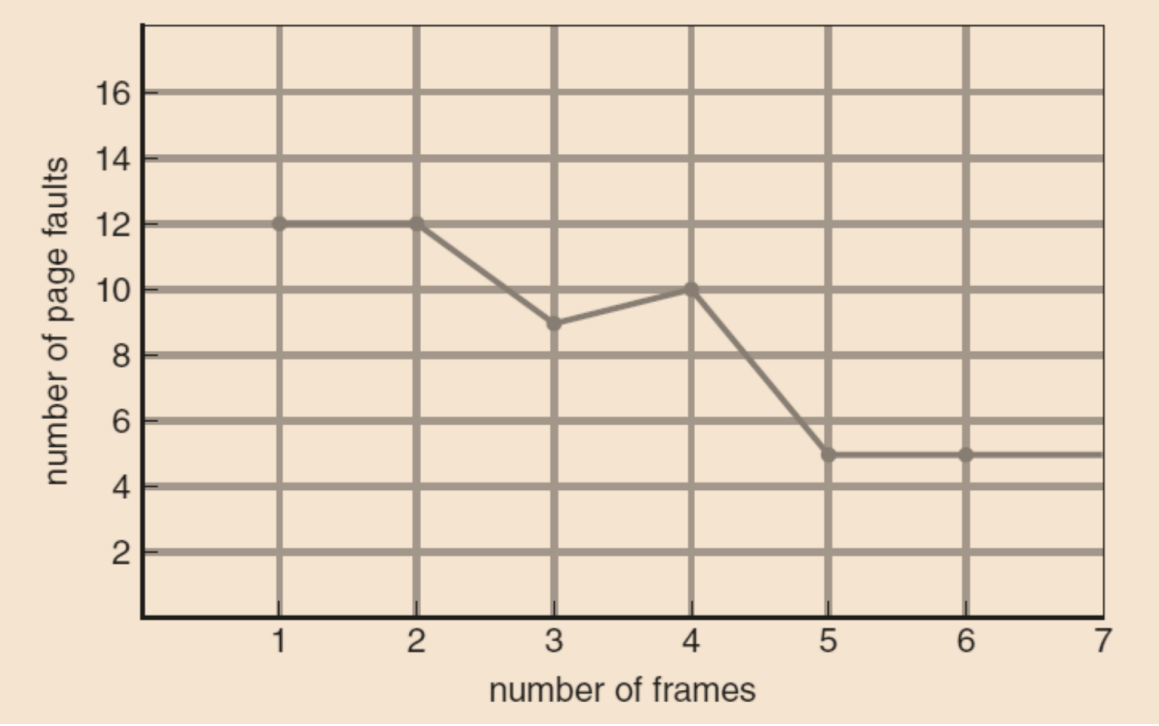


Fig 4.19 Page fault curve for FIFO replacement on a reference string.

Optimal Page Replacement

One result of the discovery of Belady's anomaly was the search for an **optimalpage-replacement algorithm**—the algorithm that has the lowest page-faultrate of all algorithms and will never suffer from Belady's anomaly. Such analgorithm does exist and has been called OPT or MIN. It is simply this:

Replace the page that will not be used for the longest period.

Use of this page-replacement algorithm guarantees the lowest possible pagefaultrate for a fixed number of frames.

For example, on our sample reference string, the optimal page-replacement algorithm would yield nine-page faults, as shown in Figure 4.20. The first threereferences cause faults that fill the three empty frames. The reference to page2 replaces page 7, because page 7 will not be used until reference 18, whereaspage 0 will be used at 5, and page 1 at 14. The reference to page 3 replacespage 1, as page 1 will be the last of the three pages in memory to be referencedagain. With only nine-page faults, optimal replacement is much better thana FIFO algorithm, which results in fifteen faults. (If we ignore the first three,which all algorithms must suffer, then optimal replacement is twice as good asFIFO replacement.) In fact, no replacement algorithmcan process this referencestring in three frames with fewer than nine faults.Unfortunately, the optimal page-replacement algorithm is difficult toimplement because it requires future knowledge of the reference string.As a result, the optimal algorithm is used mainly for comparisonstudies. For instance, it may be useful to know that, although a new algorithmis not optimal, it is within 12.3 percent of optimal at worst and within 4.7percent on average.

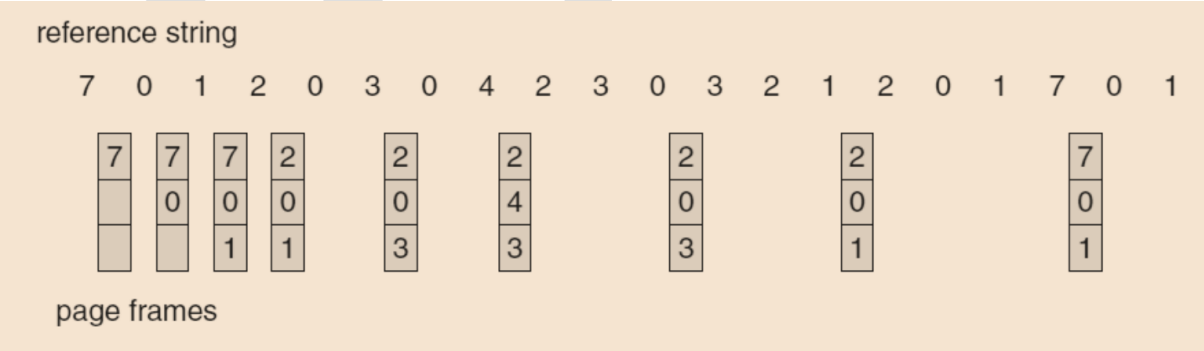


Fig. 4.20 Optimal page replacement algorithm

LRU Page Replacement

If the optimal algorithm is not feasible, perhaps an approximation of the optimalalgorithm is possible. The key distinction between the FIFO and OPT algorithms(other than looking backward versus forward in time) is that the FIFOalgorithm uses the time when a page was brought into memory, whereas theOPT algorithm uses the time when a page is to be used.If we use the recent pastas an approximation of the near future, then we can replace the page

that hasnot been usedfor the longest period of time. This approach is the **least recentlyused (LRU) algorithm**.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 usedfor the longest period of time. We can think of this strategy as the optimalpage-replacement algorithm looking backward in time, rather than forward.If we let SR be the reverse of a reference string S, then the page-faultrate for the OPT algorithm on S is the same as the page-fault rate for the OPTalgorithm on SR. Similarly, the page-fault rate for the LRU algorithm on S is thesame as the page-fault rate for the LRU algorithm on SR.The result of applying LRU replacement to our example reference string isshown in Figure 4.21. The LRU algorithm produces twelve faults. Notice thatthe first five faults are the same as those for optimal replacement. When thereference to page 4 occurs, however, LRU replacement sees that, of the threeframes in memory, page 2 was used least recently. Thus, the LRU algorithmreplaces page 2, not knowing that page 2 is about to be used.When it then faults

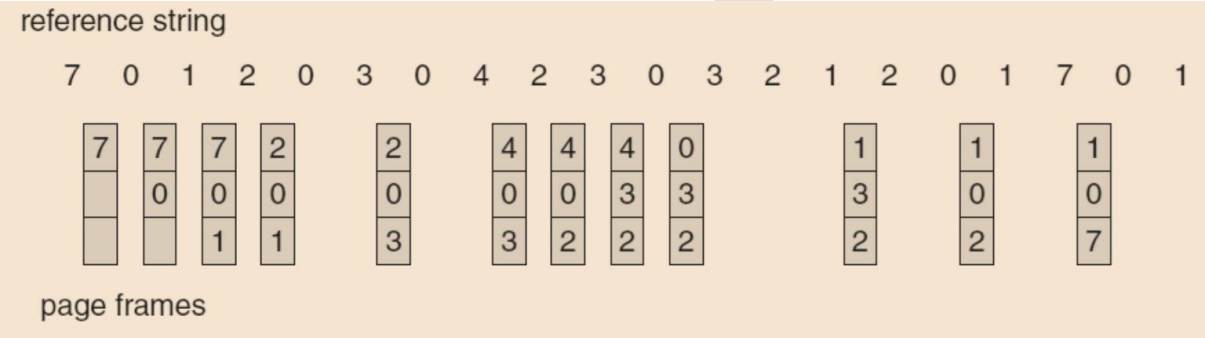


Fig. 4.21 LRU page-replacement algorithm.

for page 2, the LRU algorithm replaces page 3 since it is now the least recentlyused of the three pages in memory. Despite these problems, LRU replacementwith twelve faults is much better than FIFO replacement with fifteen.The LRU policy is often used as a page-replacement algorithm and is consideredto be good. The major problem is howto implement LRU replacement.An LRU page-replacement algorithm may require substantial hardware assistance.The problem is to determine an order for the frames defined by the timeof last use. Two implementations are feasible:

- **Counters**. In the simplest case, we associate with each page-table entry atime-of-use field and add to the CPU a logical clock or counter. The clock isincremented for every memory reference.Whenever a reference to a pageis made, the contents of the clock register are copied to the time-of-usefield in the page-table entry for that page. In this way, we always havethe "time" of the last reference to each page. We replace the page with thesmallest time value. This scheme requires a search of the page table to findthe LRU page and a write to memory (to the time-of-use field in the pageable) for each memory access. The times must also be maintained whenpage tables are changed (due to CPU scheduling). Overflow of the clockmust be considered.

- **Stack**. Another approach to implementing LRU replacement is to keep astack of page numbers.Whenever a page is referenced, it is removed fromthe stack and put on the top. In this way, the most recently used page isalways at the top of the stack, and the least recently used page is alwaysat the bottom (Figure 4.22). Because entries must be removed from themiddle of the stack, it is best to implement this approach by using a doublylinked list with a head pointer and a tail pointer. Removing a page andputting it on the top of the stack then requires changing six pointers atworst. Each update is a little more expensive, but there is no search fora replacement; the tail pointer points to the bottom of the stack, which isthe LRU page. This approach is particularly appropriate for software ormicrocode implementations of LRU replacement.

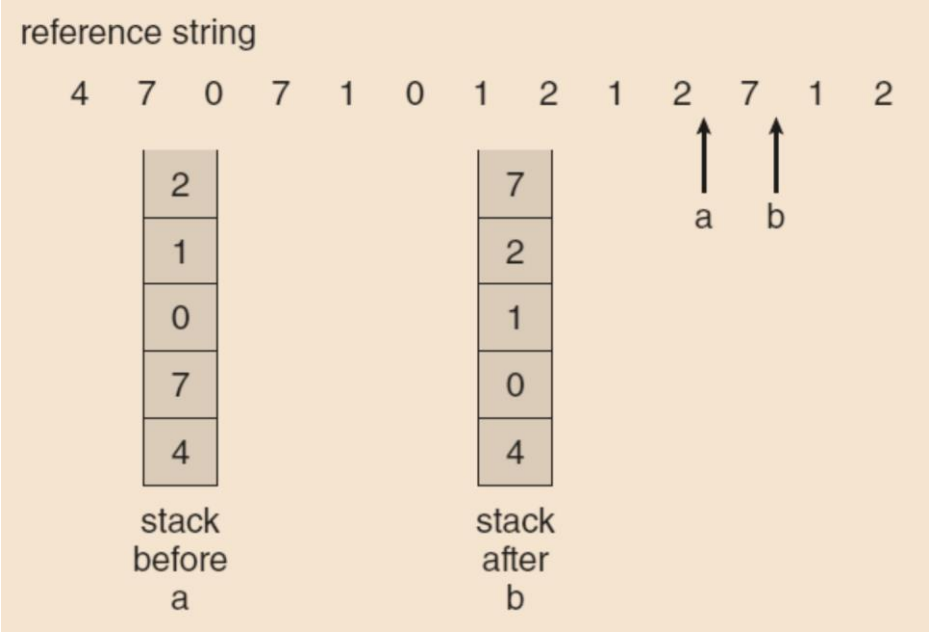


Fig. 4.22 Use of stack to record the most recent page references.

Like optimal replacement, LRU replacement does not suffer from Belady'sanomaly. Both belong to a class of page-replacement algorithms, called **stackalgorithms**, that can never exhibit Belady's anomaly. A stack algorithm is analgorithm for which it can be shown that the set of pages in memory for nframes is always a **subset** of the set of pages that would be in memory with n+ 1 frames.

For LRU replacement, the set of pages in memory would be the nmost recently referenced pages. If the number of frames is increased, these npages will still be the most recently referenced and so will still be in memory.Note that neither implementation of LRU would be conceivable withouthardware assistance beyond the standard TLB registers. The updating of theclock fields or stack must be done for **every** memory reference. If we wereto use an interrupt for every reference to allow software to update such datastructures, it would slow every memory reference by a factor of at least ten,hence slowing every process by a factor of ten. Few systems could tolerate thatlevel of overhead for memory management.

LRU-Approximation Page Replacement

Not many computer systems provide sufficient hardware support for true LRUpage replacement. In fact, some systems provide no hardware support, andother page-replacement algorithms (such as a FIFO algorithm) must be used.Many systems provide some help, however, in the form of a **reference bit**. Thereference bit for a page is set by the hardwarewhenever that page is referenced(either a read or a write to any byte in the page). Reference bits are associatedwith each entry in the page table.Initially, all bits are cleared (to 0) by the operating system. As a processexecutes, the bit associated with each page referenced is set (to 1) by thehardware.After some time,we can determinewhich pages have been used andwhich have not been used by examining the reference bits, although we do notknow the **order** of use. This information is the basis for many page-replacementalgorithms that approximate LRU replacement.

4.2.5.5 Allocation Algorithm

Allocation of Frames

We turn next to the issue of allocation. How do we allocate the fixed amount offree memory among the various processes? If we have 93 free frames and twoprocesses, how many frames does each process get?Consider a simple case of a system with 128 frames. The operating systemmay take 35, leaving 93 frames for the user process. Under pure demandpaging, all 93 frames would initially be put on the free-frame list. When a userprocess started execution, itwould generate a sequence of page faults. The first93 page faults would all get free frames from the free-frame list. When thefree-frame list was exhausted, a page-replacement algorithm would be usedto select one of the 93 in-memory pages to be replaced with the 94th, and soon. When the process terminated, the 93 frames would once again be placedon the free-frame list.There are many variations on this simple strategy. We can require that the operating system allocate all its buffer and table space from the free-frame list.When this space is not in use by the operating system, it can be used to supportuser paging.We can try to keep three free frames reserved on the free-frame listat all times. Thus, when a page fault occurs, there is a free frame available topage into.While the page swap is taking place, a replacement can be selected,which is then written to the storage device as the user process continues toexecute. Other variants are also possible, but the basic strategy is clear: theuser process is allocated any free frame.

Minimum Number of Frames

Our strategies for the allocation of frames are constrained in various ways.We cannot, for example, allocate more than the total number of availableframes (unless there is page sharing).We must also allocate at least a minimumnumber of frames. Here, we look more closely at the latter requirement.One reason for allocating at least a minimum number of frames involvesperformance. Obviously, as the number of frames allocated to each processdecreases, the page-fault rate increases, slowing process execution. In

addition,remember that, when a page fault occurs before an executing instructionis complete, the instruction must be restarted. Consequently, we must haveenough frames to hold all the different pages that any single instruction canreference.

For example, consider a machine in which all memory-reference instructionsmayreference only one memory address. In this case,we need at least oneframe for the instruction and one frame for the memory reference. In addition,if one-level indirect addressing is allowed (for example, a load instruction onframe 16 can refer to an address on frame 0, which is an indirect reference toframe 23), then paging requires at least three frames per process. (Think aboutwhat might happen if a process had only two frames.)The minimum number of frames is defined by the computer architecture.

For example, if the move instruction for a given architecture includes morethan one word for some addressing modes, the instruction itself may straddletwo frames. In addition, if each of its two operands may be indirect references,a total of six frames are required. As another example, the move instructionfor Intel 32- and 64-bit architectures allows data to move only from register toregister and between registers and memory; it does not allow direct memoryto-memory movement, thereby limiting the required minimum number offrames for a process.Whereas the minimum number of frames per process is defined by thearchitecture, the maximum number is defined by the amount of availablephysical memory. In between, we are still left with significant choice in frameallocation.

Allocation Algorithms

The easiest way to split m frames among n processes is to give everyone anequal share, m/n frames (ignoring frames needed by the operating system forthe moment). For instance, if there are 93 frames and 5 processes, each processwill get 18 frames. The 3 leftover frames can be used as a free-frame buffer pool.This scheme is called **equal allocation**.An alternative is to recognize that various processes will need differingamounts of memory. Consider a system with a 1-KB frame size. If a smallstudent process of 10 KB and an interactive database of 127 KB are the onlytwo processes running in a system with 62 free frames, it does not make muchsense to give each process 31 frames. The student process does not need morethan 10 frames, so the other 21 are, strictly speaking, wasted.To solve this problem, we can use **proportional allocation**, in which weallocate available memory to each process according to its size. Let the size ofthe virtual memory for process pi be si, and define

$S = \Sigma s_i.$

Then, if the total number of available frames is m, we allocate ai frames toprocess pi, where ai is approximatelyai = si/S × m.

Of course, we must adjust each ai to be an integer that is greater than theminimum number of frames required by the instruction set, with a sum notexceeding m.

With proportional allocation, we would split 62 frames between two processes,one of 10 pages and one of 127 pages, by allocating 4 frames and 57frames, respectively,

since 10/137 × 62 ≈ 4 and127/137 × 62 ≈ 57.

In this way, both processes share the available frames according to their"needs," rather than equally.

In both equal and proportional allocation, of course, the allocation mayvary according to the multiprogramming level. If the multiprogramming levelis increased, each processwill lose some frames to provide the memory neededfor the new process. Conversely, if the multiprogramming level decreases, theframes that were allocated to the departed process can be spread over theremaining processes.Notice that, with either equal or proportional allocation, a high-priorityprocess is treated the same as a low-priority process. By its definition, however,we may want to give the high-priority process more memory to speed its execution, to the detriment of low-priority processes. One solution is to usea proportional allocation scheme wherein the ratio of frames depends not onthe relative sizes of processes but rather on the priorities of processes or on acombination of size and priority.

4.3 Summary

- Memory is central to the operation of a modern computer system andconsists of a large array of bytes, each with its own address.
- One way to allocate an address space to each process is through the use ofbase and limit registers. The base register holds the smallest legal physicalmemory address, and the limit specifies the size of the range.
- Binding symbolic address references to actual physical addresses mayoccur during (1) compile, (2) load, or (3) execution time.
- An address generated by the CPU is known as a logical address, whichthe memory management unit (MMU) translates to a physical address inmemory.
- One approach to allocating memory is to allocate partitions of contiguousmemory of varying sizes. These partitions may be allocated based on threepossible strategies: (1) first fit, (2) best fit, and (3) worst fit.
- Modern operating systems use paging to manage memory. In this process,physical memory is divided into fixed-sized blocks called frames andlogical memory into blocks of the same size called pages.
- When paging is used, a logical address is divided into two parts: a pagenumber and a page offset. The page number serves as an index into a perprocesspage table that contains the frame in physical memory that holdsthe page. The offset is the specific location in the frame being referenced.
- Swapping allows the system to move pages belonging to a process to diskto increase the degree of multiprogramming.
- Virtual memory abstracts physical memory into an extremely large uniformarray of storage.
- The benefits of virtual memory include the following: (1) a program can belarger than physical memory, (2) a program does not need to be entirely inmemory, (3) processes can share memory, and (4) processes can be createdmore efficiently.

- Demand paging is a technique whereby pages are loaded only when theyare demanded during program execution. Pages that are never demandedare thus never loaded into memory.
- A page fault occurs when a page that is currently not in memory isaccessed. The page must be brought fromthe backing store into an availablepage frame in memory.
- Copy-on-write allows a child process to share the same address space asits parent. If either the child or the parent process writes (modifies) a page,a copy of the page ismade.
- When available memory runs low, a page-replacement algorithmselects an existing page in memory to replace with a new page. Pagereplacementalgorithms include FIFO, optimal, and LRU. Pure LRUalgorithms are impractical to implement, and most systems instead use LRU-approximation algorithms.

4.4 List of References

1. Operating System Concepts (9th Ed) by Silberschatz and Galvin, Wiley, 2000.
2. Operating Systems Design & Implementation Andrew S. Tanenbaum, AlbertS. Woodhull Pearson

4.5 List of Exercises

1. What is Memory Partitioning? Explain the concept of Dynamic Loading and Linking in memory space utilization.
2. Define swapping in memory management. Explain the concept of swapping with paging.
3. Define paging in memory management. Explain in detail the Physical and Logical memory Paging Model.
4. What is Virtual memory and virtual address space? How virtual memory is larger than the physical memory?
5. What are overlays? Explain the importance of overlays in memory management.
6. What is demand paging? Under what circumstances do page faults occur? Describe the stepstaken by the operating system in handling the page fault?
7. Explain the following Page Replacement Algorithm in detail.
   a) Basic Page Replacement
   b) FIFO Page Replacement
   c) Optimal Page Replacement
   d) LRU Page Replacement
8. Explain the following page allocation algorithm in detail.

Unit 5 - CHAPTER 5

MASS STORAGE STRUCTURE

5.1 OBJECTIVES

At the end of this unit, the student will be able to

- Illustrate the concepts of mass storage structure
- Describe the physical structure of storage devices
- Solve problems based on Disk Scheduling algorithms
- To discuss various operations like Disk management, Swap-space management, disk reliability and stable storage implementation.

5.2 INTRODUCTION

1. Since main memory is usually too small to accommodate all the data and programs permanently, the computer system must provide secondary storage to back up main memory.

2. Modern computer systems use disks as the primary on-line storage medium for information (both programs and data).

3. The file system provides the mechanism for on-line storage of and access to both data and programs residing on the disks.

4. A file is a collection of related information defined by its creator. The files are mapped by the operating system on to physical devices. Files are normally organized in to directories for ease of use.

5. The devices that attach to a computer vary in many aspects. Some devices transfer a character or a block of characters at a time. Some can be accessed only sequentially, others randomly. Some transfer data synchronously, some asynchronously.Some are dedicated or some are shared. So all devices attach to computer may vary greatly in speed.

6. Due to this variation, the OS needs to provide a wide range of functionality to applications to allow them to control aspects of the devices.

7. One key goal of an operating system's I/O subsystem is to provide the simplest interface possible to the rest of the system, because devices are a performance bottleneck, another key is to optimize I/O for maximum concurrency.

5.3 MASS STORAGE STRUCTURE-SECONDARY STORAGE STRUCTURE

1. Mass storage structure means physical structure of secondary and tertiary storage devices.

2. Magnetic Disks

2.1 It provide the bulk of secondary storage for modern computer systems. Each disk platter has a flat circular shape like a CD.Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. Information is stored magnetically on platters.

2.2 A read-write head "flies" just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided in to circular tracks, which are subdivided in to sectors. The set of tracks that are at one arm position makes up a cylinder.

2.3 When a disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute(RPM). Common drives spin at 5,400,7,200,10,000 and 15,000 RPM.

2.4 Transfer rate- is the rate at which data flow between the drive and the computer.

2.5 Positioning time or random-access time- the time necessary to move the disk arm to the desired cylinder, called the seek time and the time necessary for the desired sector to rotate the disk head, called the rotational latency.

2.6 Although the disk platters are coated with a thin protective layer, the head will sometimes damage the magnetic surface, this accident is called a head crash. A head crash normally cannot be repaired, the entire disk must be replaced.

2.7 A disk can be removable, allowing different disks to be mounted as needed.

2.8 A disk drive is attached to a computer by a set of wires called an I/O bus. Several kinds of buses are available including ATA(advanced technology attachment), Serial ATA, eSATA,USB, and fibre channel(FC).

2.8The data transfers on a bus are carried out by special electronic processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is built on each drive.

2.9 To perform a disk I/O operation, the computer places a command in to the host controller, which in turn send the command via messages to the disk controller and the disk controller operates the disk drive hardware to carry out the command.

2.10 Disk controllers usually have a built-in cache. Data transfer at the disk drive happens between the cache and the disk surface, and data transfer to the host, at fast electronic speeds, occurs between the cache and the host controller.

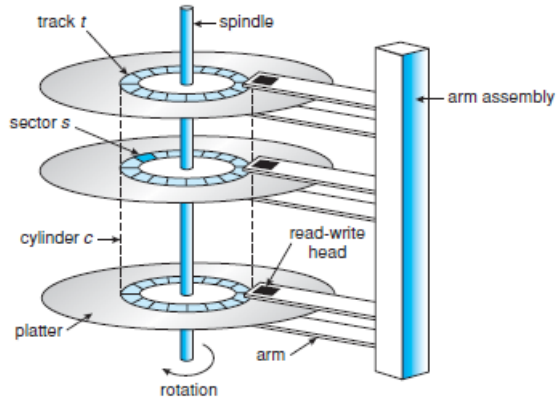


Fig 1 Moving-head disk mechanism

3 Solid-State Disks

1 Sometimes old technologies are used in new ways as economics change or the technologies evolve. An example is the growing importance of Solid-state disk or SSDs.

2. An SSD is nonvolatile memory that is used like a hard drive. There are many variations of this technology, from DRAM with a battery to allow it to maintain its state in a power failure through flash memory technologies like single-level cell (SLC) and multilevel chips.
3. SSDs have the same characteristics as traditional hard disks but can be more reliable because they have no moving parts and faster because they have no seek time or latency.
4. They consume less power, but they are expensive. One use of SSds is in storage arrays, when they hold file-system metadata that require high performance.
5. SSDs are also used in some laptop computers to make them smaller, faster and more energy-efficient.
6. SSDs are changing other traditional aspects of computer design as well, as some system use them as a direct replacement for disk drives, while others use them as a new cache tier, moving data between magnetic disks, SSDs and memory to optimize performance.

3. Magnetic tape

1. Magnetic tape was used as an early secondary-storage medium, it is relatively permanent and can hold large quantities of data, but access time is comparatively slow with respect to main memory and magnetic disk.

2. Nowadays tapes are not very useful for secondary storage as random access time is also slow.
3. But tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.
4. Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
5. There is a variation on the capacities of tape as per their type which is measured usually in terms of width as 4,8 and 19 millimeters and 1/4 and 1/2 inch.

5.4 DISK STRUCTURE

1. Modern magnetic disk drives are addressed as large one-dimensional arrays of logical block, where the logical block is the smallest unit of transfer.

2. The size of a logical block is usually 512 bytes, although some disks can be low-level formatted to have a different logical block size, such as 1024 bytes.

3. The one dimensional array of logical blocks is mapped on to the sectors of the disk sequentially.

4. Sector 0 is the first track on the outermost cylinder. The mapping proceeds in order through that track,then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

5.Ideally converting a logical block number in to an old style disk address which consist of cylinder, track number within that cylinder and a sector number within that track, is possible theoretically, but practically is not possible due to two reasons such a)as most disks have some defective sectors, but the mapping hides this by substituting spare sectors from elsewhere on the disk.b)The number of sectors per track is not a constant on some drives.

6. The farther a track is from the center of the disk, the greater its length, so the more sectors it can hold. The number of sectors per track decreases as we move from outer zones to inner zones.

7. The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head.

8. The media that use constant linear velocity(CLV), the density of bits per track is uniform, so this method is used in CD-ROM and DVD-ROM drives.

9. The media that use Constant Angular Velocity(CAV) is used by hard disk where disk rotation speed is constant.

5.5DISK SCHEDULING

1. One of the responsibilities of the OS is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.
2. For magnetic disk, the access time has two major components as 1)Seek time- is the time for the disk arm to move the heads to the cylinder containing the desired sector.2)Rotational latency-is the additional time for the disk to rotate the desired sector to the disk head. 3)Bandwidth- is the total number of bytes transferred, divided by the total time between the first request of service and completion of the last transfer.
3. Bandwidth and access time can be improved by managing the order in which disk I/O request are serviced.
4. Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information

- ✓ Whether this operation is input or output
- ✓ What the disk address for the transfer is
- ✓ What the memory address for the transfer is
- ✓ What the number of sectors to be transferred is

5. If the desired disk drive and controller are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.
6. For a multi-programming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the OS chooses which operating system chooses which pending request to service next.
7. So do this operation, there is a requirement of disk scheduling algorithm, described below as follows

1. FCFS scheduling- It is the simplest form of disk scheduling I.e first-come, first-served(FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service.
2. For eg A disk queue with request for I/O to block on cylinders,

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

Queue=98,183,37,122, 14,124,65,67

So initial head start position is 53

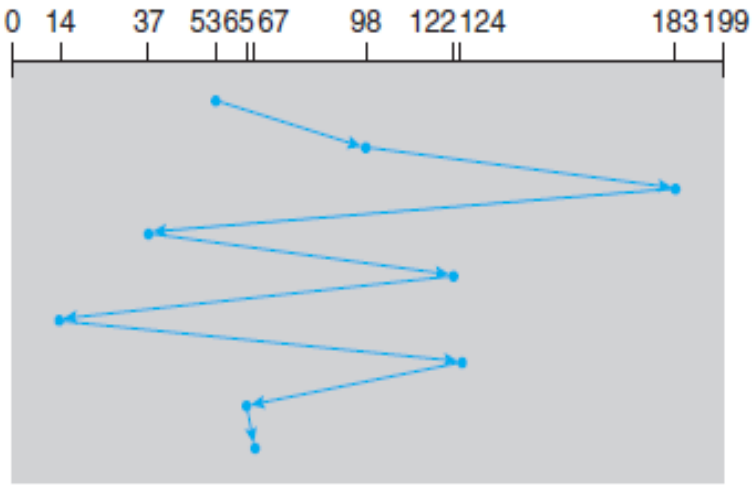


Fig 2 FCFS disk scheduling

So total seek time is - So initially head is 53. So considering first request as 98 so disk will from current head to 98, then calculation will be

Total Seek time = (98-53)+(183-98)+(183-37)+(122-37)+(122-14)+(124-14)+(124-65)+(67-65)

= 640

3. Advantages- a)Every request gets a fair chance b)No indefinite postponement
4. Disadvantages-a) Does not try to optimize seek time b) May not provide the best possible service

2. SSTF(Shortest Seek Time First)

1. It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests.

2. The SSTF algorithm selects the request with the least seek time from the current head position. In other words, SSTF chooses the pending request closest to the current head position.
3. So for eg Queue consist of following request

Queue=98,183,37,122,14,124,65,67

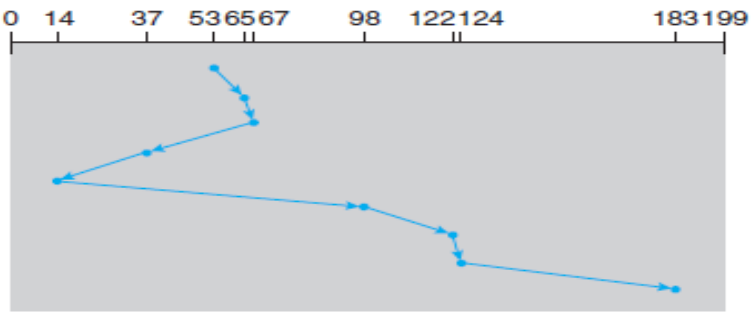


Fig 3 SSTF disk scheduling

Total Seek time- So initially disk start at head position 53, this algorithm will find 14 request is closest, instead of 37, So from 53, the disk will move from 53 to 14 first and then served all request accordingly

Total seek time=(53-14)+(37-14)+(65-37)+(67-65)+(98-67)+(122-98)+(124-122)+(183-122)

=208 (which is less than FCFS)

4. Advantages- a)Average Response time decreases b)Throughput increases
5. Disadvantage- a)Overhead increased as required to calculate seek time in advance b) Can cause starvation for a request if it has high higher seek time as compared to incoming request.c)High variance of response time as SSTF favours only some requests.

3. SCAN

1. In SCAN algorithm the disk arm moves in to a particular direction and services the request coming in its path and after reaching the end of disk, it reverses the direction and again services the request arriving in its path.
2. So this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the mid range are serviced more and those arriving behind the disk arm will have to wait.

3. For eg Request Queue=98,183,37,122,14,124,65,67. So initially disk is at 53, so from 53 disk head move to 37, then 14, then 0 and from 0 it will go to 65,67,98,122,124,183

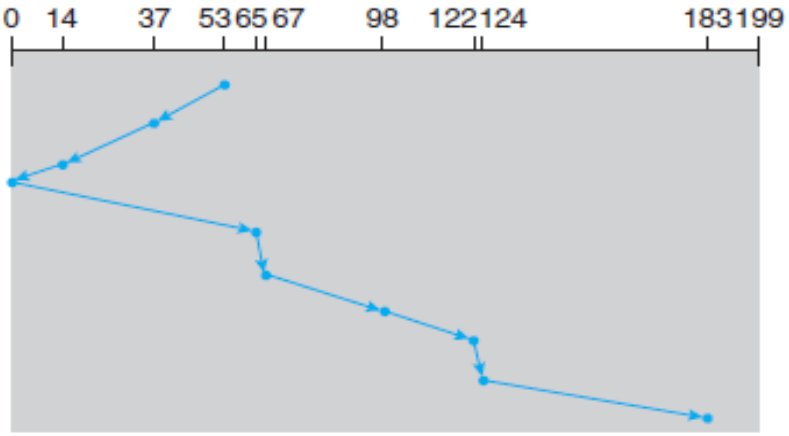


Fig 4 SCAN disk scheduling

Total seek time

= (53-37)+(37-14)+(65-0)+(65-67)+(98-67)+(122-98)+(124-122)+(183-124)

=218(Greater than SSTF)

4. Advantage- a)High throughput.b)Low variance of response time.c)Average response time
5. Disadvantage-a)Long waiting time fr requests for locations just visited by disk arm

4. C-SCAN

1. Circular SCAN(C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other end, servicing the request along the way.
2. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any request on the return trip.
3. This algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

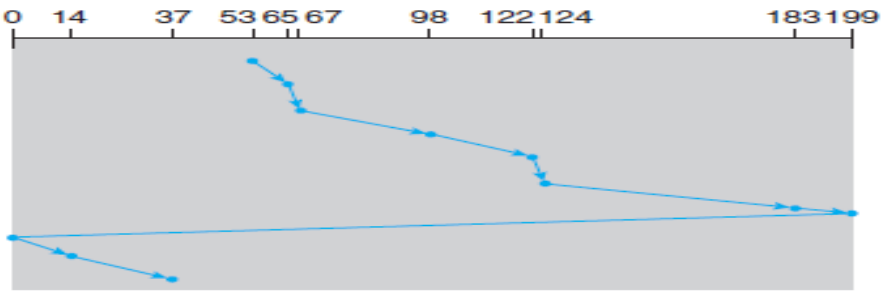


Fig 5 C-SCAN disk scheduling

4. In above diagram head starts at 53 and complete the servicing request in increasing order like from 53 ,nearest request is 65, so disk arm first move to 65,then 67,98,122,124,183,199 and then disk arm turns back to 0, serve the remaining request in the queue.

So total seek time= (65-53)+(67-65)+(98-67)+(122-98)+(124-122)+(183-124)+(14-0)+(37-14)

=167(less than SCAN)

5. Advantage- a)Provide more uniform wait time compared to SCAN

5.Look Scheduling

1.SCAN and C-SCAN move the disk arm across the full width of the disk.

2. In practice, neither algorithm is often implemented this way. More commonly, the arm 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.

3. Versions of SCAN ad C-SCAN that follow this pattern are called LOOK and C-LOOK scheduling, as they look for a request before continuing to move in a given direction.

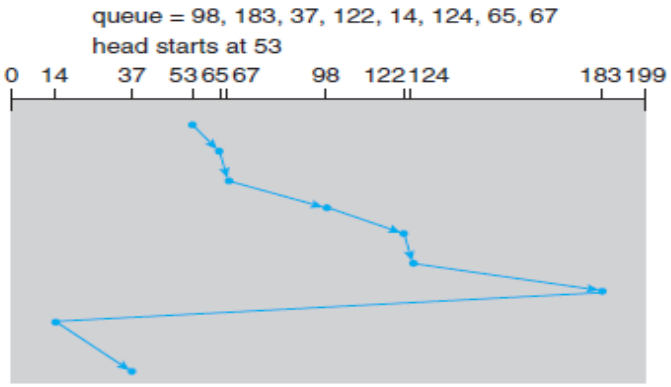


Fig 6 C-LOOK disk scheduling

6. Selection of a Disk-Scheduling algorithm

1. SSTF is common and has a natural appeal because it increases performance over FCFS, SCAN an C-SCAN perform better for systems that place a heavy load on the disk, as they are less likely to cause a starvation problem.

2. With any scheduling algorithm, however performance depends heavily on the number and types of requests.

3. Requests for disk service can be greatly influenced by the file-allocation method. A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.

4. The location of directories and index blocks is also important. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently. Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. In this case, the disk head has to move the entire width of the disk.

5. Caching the directories and index blocks in main memory can also help to reduce disk-arm movement, particularly for read request.

6. Because of these complexities, the disk-scheduling algorithm should be written as a separate module of the operating system, so that it can be replaced with a different algorithm if necessary. Either SSTF or LOOK is a reasonable choice for the default algorithm.

7. For modern disks, the rotational latency can be nearly as large as the average seek time. It is difficult for the operating system to schedule for improved rotational latency, though, because modern disks do not disclose the physical location of logical blocks. Disk

manufacturers have been alleviating this problem by implementing disk-scheduling algorithms in the controller hardware built into the disk drive.

8. If I/O performance were the only consideration, the operating system would gladly turn over the responsibility of disk scheduling to the disk hardware. In practice, however, the operating system may have other constraints on the service order for requests. For instance, demand paging may take priority over application I/O, and writes are more urgent than reads if the cache is running out of free pages. Also, it may be desirable to guarantee the order of a set of disk writes to make the file system robust in the face of system crashes.

5.6 DISK MANAGEMENT

1.The OS is responsible for several other aspects of disk management too. So Disk management strategies is described as follows

1.Disk Initialization/ Formatting

1. A new magnetic disk is a blank slate, it is just platter of a magnetic recording material. Before a disk can store data, it must be divided in to sectors that the disk controller can read and write. This process is called low-level formatting or physical formatting.

2.The data structure for a sector typically consists of a header, a data area(usually 512 bytes in size) and a trailer.

3. The header and trailer contain information used by the disk controller such as a sector number and an error-correcting code(ECC).

4.When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area. When the sector is read, the ECC value is recalculated and compared with the stored value. If there is a difference between stored value and the calculated number, that indicated that the sector got corrupted and the disk sector may be bad.

5. The ECC is a error correcting code, to correct few corrupted bit, with the corrected value, it generates a recoverable soft error. The controller automatically does the ECC processing whenever a sector is read or written.

6. For many hard disk, when the disk controller is instructed to low-level format the disk, it can also be told how many bytes of data space to leave between the header and trailer of all sectors, possibly to choose among a few sizes, such as 256,512 and 1024 bytes.

7. Before it can use a disk to hold files, the OS still needs to record its own data structures on the disk. It does so in two steps, first step is to partition the disk in to one or more groups of cylinders. For eg one partition can hold a copy of the OS executable files while other can hold user file. Each partition work independently of each other . The second step is logical formatting or creation of a file system, where OS stores the initial file-system data structure on the disk, which include maps of free and allocated space and an initial empty directory.

8. To increase efficiency, most file systems group blocks together in to larger chunks, frequently called clusters.

9.Some OS give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures. This array is sometimes called the raw disk and the I/O to this array is termed as raw I/O. For eg some database system prefer raw I/O because it enables them to control the exact disk location where each database record is stored.

10.We can make certain applications more efficient by allowing them to implement their own special-purpose storage services on a raw partition, but most applications perform better when they use the regular file-system services.

2.Boot Block

1. For a computer to start running, for eg when it is powered up or rebooted- it must have an initial program to run. The initial bootstrap program tends to be simple, the bootstrap program finds the operating system kernel on disk, loads that kernel in to memory and jumps to an initial address to begin the operating-system execution.

2. For most computers, the bootstrap is stored in read-only memory(ROM). This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset.

3. The full bootstrap program is stored in the boot blocks at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

4.The code in the boot ROM instructs the disk controller to read the boot blocks in to memory and then starts executing that code.

5. For eg booting in windows, first note that windows allows a hard disk to be divided in to partitions and one partition identified as the boot partition contains the OS and device drivers. The windows system places its boot code in the first sector on the hard disk, which in terms the master boot record or MBR.

6. In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from. Once the system identifies the boot partition, it reads the first sector from that partition (which is called the boot sector) and continues with the remainder of the boot process, which includes loading the various subsystems and system services.

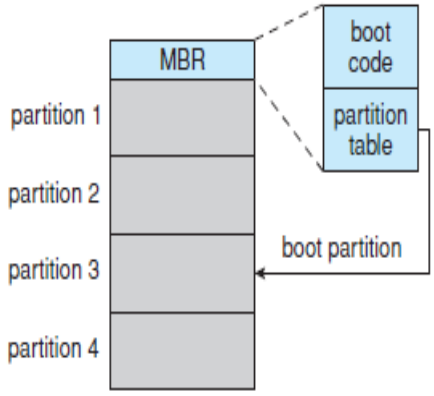


Fig 7 Booting from disk in windows

3.Bad Block

1.Disks have moving parts and small tolerances, they are prone to failure. Sometimes the failure is complete,in this case the disk needs to be replaced and its contents restored from backup media to the new disk.

2.Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of was.

3.On simple disk, such as some disks with IDE bad blocks are manually handled,strategy is to scan the disk to find bad blocks while the disk is being formatted.

4. Any bad blocks that are discovered are flagged as unusable so that the file system does not allocate them. If blocks go bad during normal operation, linux bad blocks command special program is executed to search for the bad blocks.

5.The controller maintains a list of bad blocks on the disks. The list is initialized during the low-level formatting at the factory and is updated over the life of the disk. Low-level formatting also sets aside spare sectors not visible to the operating system. 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.

6. A typical bad-sector transaction might be as follows

1. The Operating system tries to read logical block 87.

2. The controller calculates the ECC and finds that the sector is bad. It reports

this finding to the operating system.

3.The next time the system is rebooted, a special command is run to tell the

controller to replace the bad sector with a spare.

4.After that, whenever the system requests logical block 87, the request is

translated into the replacement sector's address by the controller.

7. When a bad block is remapped, the controller uses a spare sector from the same cylinder, if possible. As an alternative to sector sparing, some controllers can be instructed to replace a bad block by sector slipping. For eg Suppose that logical block 17 becomes defective and the first available spare follows sector 202. Sector slipping then remaps all the sectors from 17 to 202, moving them all down in one spot.That is, sector 202 is copied into the spare, then sector 201 In to 202, then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18 so that sector 17 can be mapped to it.
8. Soft errors may trigger a process in which a copy of the block data is made and the block is spared or slipped. An unrecoverable hard error, however, results in lost data. Whatever file was using that block must be repaired (for instance, by restoration from a backup tape), and that requires manual intervention.

5.7 SWAP-SPACE MANAGEMENT

1.Swapping in that setting occurs when the amount of physical memory reaches a critically low point and processes are moved from memory to swap space to free available memory.

2.Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance.

3.The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system. Swap Space strategies is discussed as follows

1. Swap-Space Use

1.Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use. For instance, systems that implement swapping may use swap space to hold an entire process image, including the code and data segments.

2.The amount of swap space needed on a system can therefore vary froma few megabytes of disk space to gigabytes, depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used,it may be safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort processes or may crash entirely.

3.Overestimation wastes disk space that could otherwise be used for files, but it does no other harm. Some systems recommend the amount to be set aside for swap space.

4.Solaris, for example, suggests setting swap space equal to the amount by which virtual memory exceeds pageable physical memory.

5.Some operating systems—including Linux—allow the use of multiple swap spaces, including both files and dedicated swap partitions. These swap spaces are usually placed on separate disks so that the load placed on the I/O system by paging and swapping can be spread over the system's I/O bandwidth.

2. Swap-Space Location

1.A 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 simply a large file within the file system, normal file-system routines can be

used to create it, name it, and allocate its space.

2.This approach, though easy to implement, is inefficient. Navigating the directory structure and the disk allocation data structures takes time and (possibly) extra disk accesses.

3.External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image.

4.We can improve performance by caching the block location information in physical memory and by using special tools to allocate physically contiguous blocks for the swap file, but the cost of traversing the file-system data structures remains.

5.Alternatively, swap space can be created in a separate raw partition. No file system or directory structure is placed in this space. Rather, a separate swap-space storage manager is used to allocate and deallocate the blocks from the raw partition.

6.This manager uses algorithms optimized for speed rather than for storage efficiency, because swap space is accessed much more frequently than file systems (when it is used).

7.Since swap space is reinitialized at boot time, any fragmentation is short-lived. The raw-partition approach creates a fixed amount of swap space during disk partitioning.

8.Some operating systems are flexible and can swap both in raw partitions and in file-system space. Linux is an example: the policy and implementation are separate, allowing the machine's administrator to decide which type of swapping to use.

3.Swap-Space Management- Example

1.How swapping and paging is done in various unix systems. The traditional unix kernel started with an implementation of swapping that copied entire processes between contiguous disk regions and memory.

2.Unix later on using both swapping and paging together as paging hardware is available in unix.

3.In solaris 1 (Sun Os), the designers changed standard UNIX methods to improve efficiency and reflect technological developments.

4. When a process executes, text-segment pages containing code are brought in from the file systems, accessed in main memory and thrown away if selected for page out.

5. 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.

6.More changes were made in later versions of Solaris. The biggest change is that Solaris now allocates swap space only when a page is forced out of physical memory, rather than when the virtual memory page is first created. This scheme gives better performance on modern computers, which have more physical memory than older systems and tend to page less.

7.Linux allows one or more swap areas to be established. A swap area may be in either a swap file on a regular file system or a dedicated swap partition. Each swap area consists of a series of 4-KB page slots, which are used to hold swapped pages. Associated with each swap area is a swap map—an array of integer counters, each corresponding to a page slot in the swap area.

8.If the value of a counter is 0, the corresponding page slot is available. Values greater than 0 indicate that the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page. For example, a value of 3 indicates that the swapped page is mapped to three different processes (which can occur if the swapped page is storing a region of memory shared by three processes).

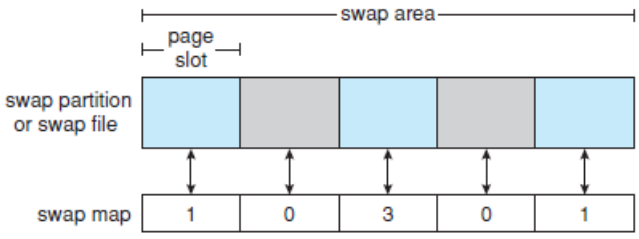


Fig 18 The data structures for swapping on Linux Systems

5.8 Disk reliability

1.The chance that some disk out of a set of N disks will fail is much higher than the chance that a specific single disk will fail. Suppose that the mean time to failure of a single disk is 100,000 hours. Then the mean time to failure of some disk in an array of 100 disks will be 100,000/100 = 1,000 hours, or 41.66 days, which is not long at all!,If we store only one copy of the data, then each disk failure will result in loss of a

significant amount of data—and such a high rate of data loss is unacceptable.

2. The solution to the problem of reliability is to introduce redundancy; we store extra information that is not normally needed but that can be used in the event of failure of a disk to rebuild the lost information. Thus, even if a disk fails, data are not lost.
3. The simplest (but most expensive) approach to introducing redundancy is to duplicate every disk. This technique is called mirroring. With mirroring, a logical disk consists of two physical disks, and every write is carried out on both disks. The result is called a mirrored volume. If one of the disks in the volume fails, the data can be read from the other. Data will be lost only if the second disk fails before the first failed disk is replaced.
4. One is the mean time to failure of the individual disks. The other is the mean time to repair, which is the time it takes (on average) to replace a failed disk and to restore the data on it. Suppose that the failures of the two disks are independent; that is, the failure of one disk is not connected to the failure of the other. Then, if the mean time to failure of a single disk is 100,000 hours and the mean time to repair is 10 hours, the mean time to data loss of a mirrored disk system is $100,\ 0002/(2 * 10) = 500 * 106$ hours, or 57,000 years.
5. Power failures and natural disasters, such as earthquakes, fires, and floods, may result in damage to both disks at the same time. Also, manufacturing defects in a batch of disks can cause correlated failures.
6. As disks age, the probability of failure grows, increasing the chance that a second disk will fail while the first is being repaired. In spite of all these considerations, however, mirrored-disk systems offer much higher reliability than do single-disk systems.

7. Even with mirroring of disks, if writes are in progress to the same block in both disks, and power fails before both blocks are fully written, the two blocks can be in an inconsistent state.
8. Solution to this problem is to write one copy first, then the next. Another is to add a solid-state nonvolatile RAM (NVRAM) cache to the RAID array.
9. This write-back cache is protected from data loss during power failures, so the write can be considered complete at that point, assuming the NVRAM has some kind of error protection and correction, such as ECC or mirroring.

5.9 STABLE STORAGE IMPLEMENTATION

1.By definition, information residing in stable storage is never lost. To implement such storage, we need to replicate the required information on multiple storage devices(usually disks) with independent failure modes.

2.We also need to coordinate the writing of updates in a way that guarantees that a failure during an update will not leave all the copies in a damaged state and that, when we are recovering from a failure, we can force all copies to a consistent and correct value, even if another failure occurs during the recovery.

3.A disk write results in one of three outcomes

1.Successful Completion- The data were written correctly on disk.

2. Partial Failure-A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted.

3.Total Failure- The failure occurred before the disk write started, so the previous data values on the disk remain intact.

4.Whenever a failure occurs during writing of a block, the system needs to detect it and invoke a recovery procedure to restore the block to a consistent state. To do that, the system must maintain two physical blocks for each logical block. An output operation is executed as follows

1.Write the information on to the first physical block.

2.When the first write completes successfully, write the same information on to the second physical clock

3.Declare the operation complete only after the second write completes successfully.

5.During recovery from a failure, each pair of physical blocks is examined. If both are the same and no detectable error exists, then no further action is necessary.

6,If one block contains a detectable error then we replace its contents with the value of the other block. If neither block contains a detectable error, but the blocks differ in content, then we replace the content of the first block with that of the second.

7.This recovery procedure ensures that a write to stable storage either succeeds completely or results in no change.

8.The data in stable storage are guaranteed to be safe unless a failure destroys all the copies.Because waiting for disk writes to complete (synchronous I/O) is time consuming, many storage arrays add NVRAM as a cache.

9.Since the memory is nonvolatile (it usually has battery power to back up the unit's power), it can be trusted to store the data en route to the disks. It is thus considered part of the stable storage. Writes to it are much faster than to disk, so performance is greatly improved.

Miscellaneous Questions

Q1. Is disk scheduling, other than FSFS scheduling, useful in a single-user environment?Explain your answer.

Q2. Explain all disk scheduling techniques?

Q3. Explain why SSTF scheduling tend to favor middle cylinder over the innermost and outermost cylinders

Q4. Describe the concept of Mass storage

Q5. Illustrate the concept of stable storage implementation

Q6. What are the ways to increase the Disk Reliability

Q7. Perform the FSCFS,SSTF,SCAN,CSCAN on given request as 82,170,43,140,24,16,190. Calculate total seek time. Also discuss which scheduling algorithm is best.

Q8. Why Swap-Space management is necessary in OS?

Unit 5 - CHAPTER 6

INTRODUCTION TO CLOCK

6.1 OBJECTIVES

At the end of this unit, the student will be able to

- Describe the use of clock in Operating System
- Illustrate the use of hardware in clock system of machine
- Explain the use of software in clock system of machine

6.2 INTRODUCTION TO CLOCK

1.Clocks are also called timers.

2.The clock software takes the form of a device driver though a clock is neither a blocking device nor a character based device.

3.The clock software is the clock driver.

4.The exact function of the clock driver may vary depending on operating system.

5.Generally, the functions of the clock driver include the following in Table 1 Task of clock

Sr.No	Task	Description
1	Maintaining the time of the day	1.The clock driver implements the time of day or the real time clock function. 2.It requires incrementing a counter at each clock tick.
2	Preventing processes from running too long	1.As a process is started, the scheduler

		initializes the quantum counter in clock ticks for the process. 2.The clock driver decrements the quantum counter by 1, at every clock interrupt. When the counter gets to zero , clock driver calls the scheduler to set up another process. 3.Thus clock driver helps in preventing processes from running longer than time slice allowed.
3	Accounting for CPU usage	1.Another function performed by clock driver is doing CPU accounting. 2.CPU accounting implies telling how long the process has run.
4	Providing watchdog timers for parts of the system itself	1. Watchdog timers are the timers set by certain parts of the system. 2. For example, to use a floppy disk, the system must turn on the motor and then wait about 500msec for it to comes up to speed.

6.3 CLOCK HARDWARE(Explanation for MINIX OS case study)

1.Two types of clocks are used in computers, and both are quite different from the clocks and watches used by people. The simpler clocks are tied to the 110- or 220-volt power line, and cause an interrupt on every voltage cycle, at 50 or 60 Hz.

2.These are essentially extinct in modern PCs.

3The other kind of clock is built out of three components: a crystal oscillator, a counter, and a holding register, as shown in Fig. 2-47.

4When a piece of quartz crystal is properly cut and mounted under tension, it can be made to generate a periodic signal of very high accuracy, typically in the range of 5 to 200 MHz, depending on the crystal chosen.

5At least one such circuit is usually found in any computer, providing a synchronizing signal to the computer's various circuits. This signal is fed into the counter to make it count down to zero.

6 When the counter gets to zero, it causes a CPU interrupt. Computers whose advertised clock rate is higher than 200 MHz normally use a slower clock and a clock multiplier circuit.

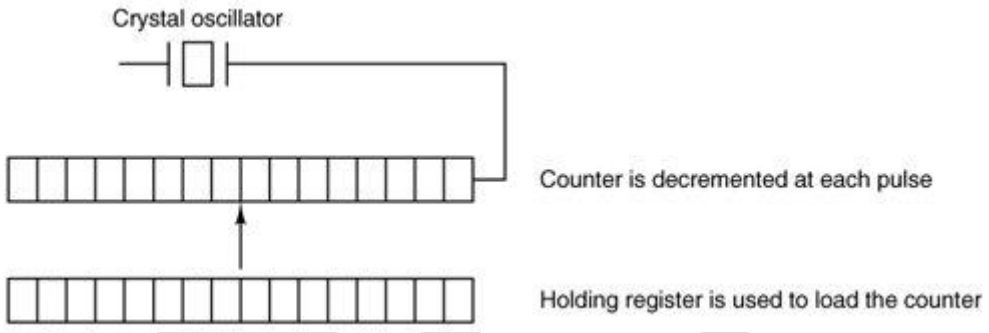


Fig 1 A programmable Clock

7 Programmable clocks typically have several modes of operation.

7.1 In one-shot mode, when the clock is started, it copies the value of the holding registerinto the counter and then decrements the counter at each pulse from the crystal.

7.2 When the counter gets to zero, it causes an interrupt and stops until it is explicitly started again by the software.

7.3 In square-wave mode, after getting to zero and causing the interrupt, the holding register is automatically copied into the counter, and the whole process is repeated again indefinitely.

7.4 These periodic interrupts are called clock ticks.

8.Advantages

8.1the programmable clock is that its interrupt frequency can be controlled bysoftware. If a 1-MHz crystal is used, then the counter is pulsed every microsecond.

8.2 With 16-bit registers, interrupts can be programmed to occur at intervals from 1 microsecond to 65.536 milliseconds.

8.3 Programmable clock chips usually contain two or three independently programmable clocks and have many other options as well (e.g., counting up instead of down, interrupts disabled, and more).

9.To prevent the current time from being lost when the computer's power is turned off, most computers have a battery-powered backup clock, implemented with the kind of low-power circuitry used in digital watches.

10.The battery clock can be read at startup. If the backup clock is not present, the software may ask the user for the current date and time. There is also a standard protocol for a networked system to get the current time from a remote host.

11.In any case the time is then translated into the number of seconds since 12 A.M. Universal Coordinated Time (UTC) (formerly known as Greenwich Mean Time) on Jan. 1, 1970, as UNIX and MINIX 3 do, or since some other benchmark.

12.Clock ticks are counted by the running system, and every time a full second has passed the real time is incremented by one count.

13.MINIX 3 (and most UNIX systems) do not take into account leap seconds, of which there have been 23 since 1970. This is not considered a serious flaw.

14.Usually, utility programs are provided to manually set the system clock and the backup clock and to synchronize the two clocks.

15.We should mention here that all but the earliest IBM-compatible computers have a separate clock circuit that provides timing signals for the CPU, internal data busses, and other components.

16.This is the clock that is meant when people speak of CPU clock speeds, measured in Megahertz on the earliest personal computers, and in Gigahertz on modern systems.

17.The basic circuitry of quartz crystals, oscillators and counters is the same, but the requirements are so different that modern computers have independent clocks for CPU control and timekeeping.

6.4 CLOCK SOFTWARE(With respect to MINIX OS Case study)

1.All the clock hardware does is generate interrupts at known intervals. Everything else involving time must be done by the software, the clock driver.

2. The exact duties of the clock driver vary among operating systems, but usually include most of the following as discussed here

- Maintaining the time of day.
- Preventing processes from running longer than they are allowed to.
- Accounting for CPU usage.
- Handling the alarm system call made by user processes.
- Providing watchdog timers for parts of the system itself.
- Doing profiling, monitoring, and statistics gathering.

3. The first clock function, maintaining the time of day (also called the real time) is not difficult. It just requires incrementing a counter at each clock tick, as mentioned before.

4. The only thing to watch out for is the number of bits in the time-of-day counter.ith a clock rate of 60 Hz, a 32-bit counter will overflow in just over 2 years.

5. Clearly the system cannot store the real time as the number of ticks since Jan. 1, 1970 in 32 bits.

6. Three approaches can be taken to solve this problem. The first way is to use a 64-bit counter, although doing so makes maintaining the counter more expensive since it has to be done many times a second.

6.1 The second way is to maintain the time of day in seconds, rather than in ticks, using a subsidiary counter to count ticks until a whole second has been accumulated. Because 2^{32} seconds is more than 136 years, this method will work until well into the twenty-second century.

6.2 The third approach is to count ticks, but to do that relative to the time the system was booted, rather than relative to a fixed external moment. When the backup clock is read or the

user types in the real time, the system boot time is calculated from the current time-of-day value and stored in memory in any convenient form.

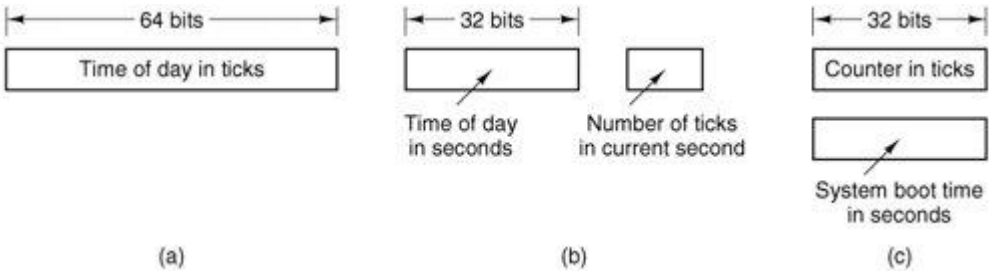


Fig 2 Three approaches to maintain a clock

7. The second clock function is preventing processes from running too long. Whenever a process is started, the scheduler should initialize a counter to the value of that process' quantum in clock ticks.

8. At every clock interrupt, the clock driver decrements the quantum counter by 1. When it gets to zero, the clock driver calls the scheduler to set up another process.

9. The third clock function is doing CPU accounting. The most accurate way to do it is to start a second timer, distinct from the main system timer, whenever a process is started. When that process is stopped, the timer can be read out to tell how long the process has run.

10. To do things right, the second timer should be saved when an interrupt occurs and restored afterward.

11. A less accurate, but much simpler, way to do accounting is to maintain a pointer to the process table entry for the currently running process in a global variable. At every clock tick, a field in the current process' entry is incremented.

12. A minor problem with this strategy is that if many interrupts occur during a process' run, it is still charged for a full tick, even though it did not get much work done. Properly accounting for the CPU during interrupts is too expensive and is rarely done.

13. In MINIX 3 and many other systems, a process can request that the operating system give it a warning after a certain interval. The warning is usually a signal, interrupt, message, or something similar.

14. One application requiring such warnings is networking, in which a packet not acknowledged within a certain time interval must be retransmitted. Another application is computer-aided instruction, where a student not providing a response within a certain time is told the answer.

15. If the clock driver had enough clocks, it could set a separate clock for each request. This not being the case, it must simulate multiple virtual clocks with a single physical clock.

16. One way is to maintain a table in which the signal time for all pending timers is kept, as well as a variable giving the time of the next one.

17. Whenever the time of day is updated, the driver checks to see if the closest signal has occurred. If it has, the table is searched for the next one to occur.

18. If many signals are expected, it is more efficient to simulate multiple clocks by chaining all the pending clock requests together, sorted on time, in a linked list, as shown in Fig given below.

19. Each entry on the list tells how many clock ticks following the previous one to wait before causing a signal. In this example, signals are pending for 4203, 4207, 4213, 4215, and 4216.

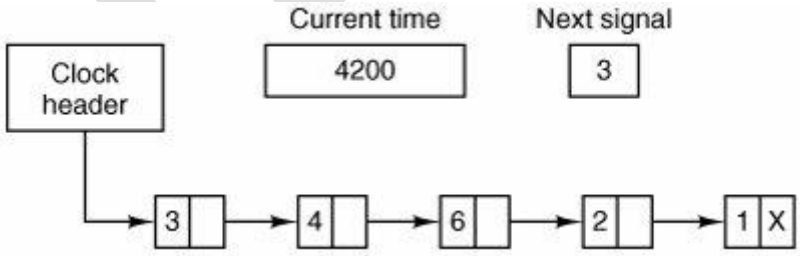


Fig 3 Simulating multiple timers with a single clock

20. On each tick, Next signal is decremented. When it gets to 0, the signal corresponding to the first item on the list is caused, and that item is removed from the list.

21. Then Next signal is set to the value in the entry now at the head of the list, in this example, 4. Using absolute times rather than relative times is more convenient in many cases, and that is the approach used by MINIX 3.

22. During a clock interrupt, the clock driver has several things to do. These things include incrementing the real time, decrementing the quantum and checking for 0, doing CPU accounting, and decrementing the alarm counter. However, each of these operations has been carefully arranged to be very fast because they have to be repeated many times a second.

23. Parts of the operating system also need to set timers. These are called watchdog timers.Floppy disk drivers use timers to wait for the disk motor to get up to speed and to shut down the motor if no activity occurs for a while. Some printers with a movable print head can print at 120 characters/sec (8.3 msec/character) but cannot return the print head to the left margin in 8.3 msec, so the terminal driver must delay after typing a carriage return.

24. The system task, which is in kernel space, can set alarms on behalf of some user-space processes, and then notify them when a timer goes off.

25. Some operating systems provide a mechanism by which a user program can have the system build up a histogram of its program counter, so it can see where it is spending its time.

26. When profiling is a possibility, at every tick the driver checks to see if the current process is being profiled, and if so, computes the bin number (a range of addresses) corresponding to the current program counter. It then increments that bin by one. This mechanism can also be used to profile the system itself.

II.Overview of Clock Driver in MINIX 3

1.The MINIX 3 clock driver is contained in the file kernel/clock.c. It can be considered to have three functional parts.

2.First, like the device drivers that we will see in the next chapter, there is a task mechanism which runs in a loop, waiting for messages and dispatching to subroutines that perform the

action requested in each message. However, this structure is almost vestigial in the clock task.

3.So for the clock this is used only when there is a substantial amount of work to be done. Only one kind of message is received, there is only one subroutine to service the message, and a reply message is not sent when the job is done.

4.The second major part of the clock software is the interrupt handler that is activated 60 times each second. It does basic timekeeping, updating a variable that counts clock ticks since the system was booted.

5.If the interrupt handler detects that a process has used its quantum or that a timer has expired it generates the message that goes to the main task loop.

6.Otherwise no message is sent. The strategy here is that for each clock tick the handler does as little as necessary, as fast as possible. The costly main task is activated only when there is substantial work to do.

7.The third general part of the clock software is a collection of subroutines that provide general support, but which are not called in response to clock interrupts, either by the interrupt handler or by the main task loop.

8.One of these subroutines is coded as PRIVATE, and is called before the main task loop is entered. It initializes the clock, which entails writing data to the clock chip to cause it to generate interrupts at the desired intervals.

9.The initialization routine also puts the address of the interrupt handler in the right place to be found when the clock chip triggers the IRQ 8 input to the interrupt controller chip, and then enables that input to respond.

10.The rest of the subroutines in clock.c are declared PUBLIC, and can be called from anywhere in the kernel binary. In fact none of them are called from clock.c itself. They are mostly called by the system task in order to service system calls related to time.

11.These subroutines do such things as reading the time-since-boot counter, for timing with clock-tick resolution, or reading a register in the clock chip itself, for timing that requires microsecond resolution. Other subroutines are used to set and reset timers.

12.Finally, a subroutine is provided to be called when MINIX 3 shuts down. This one resets the hardware timer parameters to those expected by the BIOS.

III. The Clock Task

1.The main loop of the clock task accepts only a single kind of message, HARD_INT, which comes from the interrupt handler.

2.Anything else is an error. Furthermore, it does not receive this message for every clock tick interrupt, although the subroutine called each time a message is received is named do_clocktick.

3.A message is received, and do_clocktick is called only if process scheduling is needed or a timer has expired.

IV.The Clock Interrupt Handler

1.The interrupt handler runs every time the counter in the clock chip reaches zero and generates an interrupt.

2.The clock software supplies only the current tick count to aid a system call for the real time. Further processing is done by one of the servers. This is consistent with the MINIX 3 strategy of moving functionality to processes that run in user space.

3.In the interrupt handler the local counter is updated for each interrupt received. When interrupts are disabled ticks are lost.

4.In some cases it is possible to correct for this effect. A global variable is available for counting lost ticks, and it is added to the main counter and then reset to zero each time the handler is activated. In the implementation section we will see an example of how this is used.

5.The handler also affects variables in the process table, for billing and process control purposes. A message is sent to the clock task only if the current time has passed the expiration time of the next scheduled timer or if the quantum of the running process has been decremented to zero.

6.Everything done in the interrupt service is a simple integer operationarithmetic, comparison, logical AND/OR, or assignmentwhich a C compiler can translate easily into basic machine operations.

7.At worst there are five additions or subtractions and six comparisons, plus a few logical operations and assignments in completing the interrupt service. In particular there is no subroutine call overhead.

V. Watch Dog Timers

1.A few pages back we left hanging the question of how user-space processes can be provided with watchdog timers, which ordinarily are thought of as user-supplied procedures that are part of the user's code and are executed when a timer expires. Clearly, this can not be done in MINIX 3.

2.But we can use a synchronous alarm to bridge the gap from the kernel to user space.A signal may arrive or a conventional watchdog may be activated without any relation to what part of a process is currently executing, so these mechanisms are asynchronous.

3.A synchronous alarm is delivered as a message, and thus can be received only when the recipient has executed receive.

4.So we say it is synchronous because it will be received only when the receiver expects it. If the notify method is used to inform a recipient of an alarm, the sender does not have to block, and the recipient does not have to be concerned with missing the alarm.

5.Messages from notify are saved if the recipient is not waiting. A bitmap is used, with each bit representing a possible source of a notification.

6.Watchdog timers take advantage of the timer_t type s_alarm_timer field that exists in each element of the priv table. Each system process has a slot in the priv table.

7.To set a timer, a system process in user space makes a sys_setalarm call, which is handled by the system task.

8.The system task is compiled in kernel space, and thus can initialize a timer on behalf of the calling process.

9.The procedure to execute has to be in kernel space too, of course. No problem. The system task contains a watchdog function, cause_alarm, which generates a notify when it goes off, causing a synchronous alarm for the user.

10.This alarm can invoke the user-space watchdog function. Within the kernel binary this is a true watchdog, but for the process that requested the timer, it is a synchronous alarm. It is not

the same as having the timer execute a procedure in the target's address space. There is a bit more overhead, but it is simpler than an interrupt.

11.Each system process has a copy of the priv structure, but a single copy is shared by all non-system (user) processes. The parts of the priv table that cannot be shared, such as the bitmap of pending notifications and the timer, are not usable by user processes.

12.The solution is this: the process manager manages timers on behalf of user processes in a way similar to the way the system task manages timers for system processes. Every process has a timer_t field of its own in the process manager's part of the process table.

13.When a user process makes an alarm system call to ask for an alarm to be set, it is handled by the process manager, which sets up the timer and inserts it into its list of timers.

14.The process manager asks the system task to send it a notification when the first timer in the PM's list of timers is scheduled to expire.

15.he process manager only has to ask for help when the head of its chain of timers changes, either because the first timer has expired or has been cancelled, or because a new request has been received that must go on the chain before the current head.

16.This is used to support the POSIX-standard alarm system call. The procedure to execute is within the address space of the process manager. When executed, the user process that requested the alarm is sent a signal, rather than a notification.

VI Millisecond Timing

1. A procedure is provided in clock.c that provides microsecond resolution timing. Delays as short as a few microseconds may be needed by various I/O devices.

2. There is no practical way to do this using alarms and the message passing interface. The counter that is used for generating the clock interrupts can be read directly.

3. It is decremented approximately every 0.8 microseconds, and reaches zero 60 times a second, or every 16.67 milliseconds.

4. To be useful for I/O timing it would have to be polled by a procedure running in kernel-space, but much work has gone into moving drivers out of kernel-space. Currently this function is used only as a source of randomness for the random number generator.

VII Summary of Clock Services is explained through table which has following parameter like service, Access, Response and clients

Service	Access	Response	Clients
get_uptime	Function call	Ticks	Kernel or system task
set_timer	Function call	None	Kernel or system task
reset_timer	Function call	None	Kernel or system task
read_clock	Function call	Count	Kernel or system task
clock_stop	Function call	None	Kernel or system task
Synchronous alarm	System call	Notification	Server or driver, via system task
POSIX alarm	System call	Signal	User process, via PM
Time	System call	Message	Any process, via PM

VIII Implementation of the Clock Driver in MINIX 3

1. The clock task uses no major data structures, but several variables are used to keep track of time. The variable realtime (line 10462) is basic , as it counts all clockticks.

2. A global variable, lost_ticks, is defined in glo.h (line 5333). This variable is provided for the use of any function that executes in kernel space that might disable interrupts long enough that one or more clock ticks could be lost.

3. It currently is used by the int86 function in klib386.s. Int86 uses the boot monitor to manage the transfer of control to the BIOS, and the monitor returns the number of clock ticks counted while the BIOS call was busy in the ecx register just before the return to the kernel.

4. This works because, although the clock chip is not triggering the MINIX 3 clock interrupt handler when the BIOS request is handled, the boot monitor can keep track of the time with the help of the BIOS.

5. The clock driver accesses several other global variables. It uses proc_ptr, prev_ptr, and bill_ptr to reference the process table entry for the currently running process, the process that ran previously, and the process that gets charged for time.

6. Within these process table entries it accesses various fields, including p_user_time and p_sys_time for accounting and p_ticks_left for counting down the quantum of a process.

7. When MINIX 3 starts up, all the drivers are called. Most of them do some initialization then try to get a message and block.

8. The clock driver, clock_task (line 10468), does that too. First it calls init_clock to initialize the programmable clock frequency to 60 Hz. When a message is received, it calls do_clocktick if the message was a HARD_INT (line 10486). Any other kind of message is unexpected and treated as an error.

9. One of the conditions that results in running do_clocktick is the current process using up all of its quantum.

10. If the process is preemptable (the system and clock tasks are not) a call to lock_dequeue followed immediately by a call to lock_enqueue (lines 10510 to 10512) removes the process from its queue, then makes it ready again and reschedules it.

11. The other thing that activates do_clocktick is expiration of a watchdog timer. Timers and linked lists of timers are used so much in MINIX 3 that a library of functions to support them was created.

12. The library function tmrs_exptimers called on line 10517 runs the watchdog functions for all expired timers and deactivates them.

13. Init_clock (line 10529) is called only once, when the clock task is started. There are several places one could point to and say, "This is where MINIX 3 starts running." This is a candidate; the clock is essential to a preemptive multitasking system.

14. Init_clock writes three bytes to the clock chip that set its mode and set the proper count into the master register. Then it registers its process number, IRQ, and handler address so interrupts will be directed properly. Finally, it enables the interrupt controller chip to accept clock interrupts.

15. The next function, clock_stop, undoes the initialization of the clock chip. It is declared PUBLIC and is not called from anywhere in clock.c.

16. It is placed here because of the obvious similarity to init_clock. It is only called by the system task when MINIX 3 is shut down and control is to be returned to the boot monitor.

17. As soon as (or, more accurately, 16.67 milliseconds after) init_clock runs, the first clock interrupt occurs, and clock interrupts repeat 60 times a second as long as MINIX 3 runs

18. The code in clock_handler (line 10556) probably runs more frequently than any other part of the MINIX 3 system.

19. Consequently, clock_handler was built for speed. The only subroutine calls are on line 10586; they are only needed if running on an obsolete IBM PS/2 system. The update of the current time (in ticks) is done on lines 10589 to 10591. Then user and accounting times are updated.

20. Decisions were made in the design of the handler that might be questioned.

21. Two tests are done on line 10610 and if either condition is true the clock task is notified. The do_clocktick function called by the clock task repeats both tests to decide what needs to be done.

22. This is necessary because the notify call used by the handler cannot pass any information to distinguish different conditions. We leave it to the reader to consider alternatives and how they might be evaluated.

23. Get_uptime (line 10620) just returns the value of realtime, which is visible only to functions in clock.c. Set_timer and reset_timer use other functions from the timer library that take care of all the details of manipulating a chain of timers. Finally, read_clock reads and returns the current count in the clock chip's countdown register.

Miscellaneous Questions

Q1. Describe in brief the concept of Clock Hardware?

Q2. List the clock services of MINIX OS

Q3. How Clock driver is implemented in MINIX OS?

Q4. What is the use of millisecond time in OS?

Q5. What is Watch Dog Timers?

Q6. Explain the phases of implementing clock software?

Unit 6:Chapter 7
File Systems

Unit Structure

7.0 Objectives

This chapter would make you understand the following concepts:

- Functionalities of a File system.
- File Access methods.
- File Allocation methods.
- Concept of Directory systems.
- To understand File protection concept.
- Explore Free Space Management.

7.1 Introduction

The one of the major functionality of an operating system is a file system. Operating system contains a module file system which manages the files. It manages the file that how to store the data and how to fetch the data from a file. It is used to control each and every activities regarding files which store on secondary memory such as hard disk.For example, DOS contain FAT file system, windows OS includes NTFS file system, Linux includes Extended File system.So any of the operating system contains one module which is a File system.

7.2 File Concepts

7.2.1 File

File is basically a form of data that computer understands or computer can interpret. Data cannot write on secondary storage unless it is in file form. Computer can understand many different files such as mp3 files,text files, exe files, library files and many more. So data has to be is any of the format which computer can understand and that format is called as a file.

Another definition of a file is “File is a collection of bits,bytes or characters.” Or “A file is a named collection of related information stored on secondary storage.”

The File System is called as an Information management system module of the operating system.This module stores information on secondary memory in terms of files so it is called as File system. There are different types of the files given in figure below:

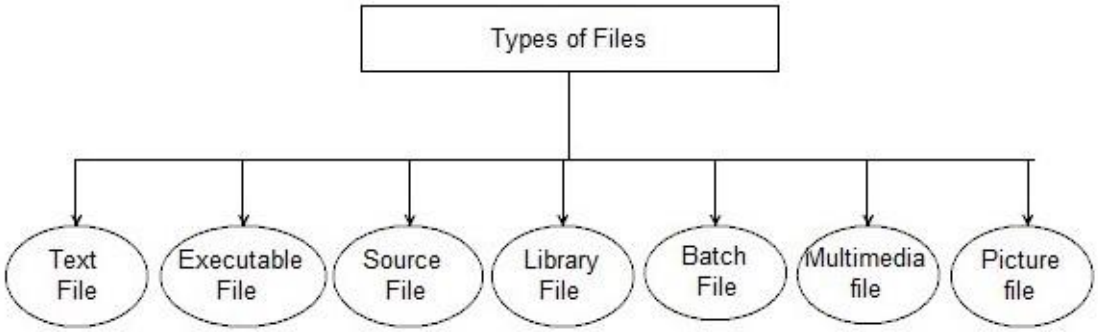


Fig 1.2.1 Types of Files

- Text file : Text files are used to store text or character set using ASCII codes.
- Executable file: It store machine language programs so they are directly executable.
- Source file: This is a text file used to store high level language programs.
- Library files: Library functions are stored in library files.
- Batch file: Commands related to the operating system stored in Batch file.
- Musical or multimedia file: This files are used to store musical data.
- Picture files: Picture files are used to store picture.

7.2.2File operations

There are many ways to define a file properly,so we must need to consider various operations executed on files.There are different system calls used to create, read, write, reposition, delete, truncate files. There are different file operations that are performed on a files which are given below:

1) **Creating a file**

To create a file two steps are necessary

- Found the space in the file system for the new file.
- Then, an entry for the new file must be made in the directory.

2) **Writing a file**

The write system call that perform write operation requires two attributes such as file name where data is to be written and second information that is to be written in a file.

After taking the name of the file,system searches the directory to find the file's location and keep a write pointer to the location in the file where the next write is to take place.

3) **Reading a file**

The read system call perform read operation which requires two attributes such as name of the file and where the next block of the file should be put. After specifying the name of the file, system searches the directory for the file and it also needs to have a read pointer pointing to the next block to read. After that file pointer is pointed to first location of the file and it reads content of files till it does not end.

4) **Repositioning within a file**

The directory is searched for the appropriate entry,and the current-file-position pointer is moved to a given value.Repositioning in a file need not involve any actual I/O.

5) **Deleting a file**

To delete a file, operating system searches the directory for the file and after finding the directory entry, it releases all file space. When the file is notnecessary, it has to be deleted to free up disk space.The space which is assigned to the file will now become available and can be allocated to the other files.

6) **Truncating a file**

The user want to delete the contents of a file but keep its attributes.Rather than forcing the user to delete the file and then recreate it,this function allows all attributes to keep it unchanged.

7.2.3File Attribute

All files have the common attributes whenever files are stored on secondary storage. Every file has a same attributes. To identify name and location of a file, file attributes are used. There are different file attributes which is listed below:

1. **Name:** Name attribute is the logical name given to the file and it is represented through the symbol.

2. **Identifier:** Identifier is unique number that can identifies file in the file system by a number and that number is assigned by a file system.
3. **Location:** Location attribute mention the address of a file on a disk or a device.
4. **Type:** Type attribute locates the address of a file on a disk or a device.
5. **Size:** This attribute is used to show the current size of the file which is counted in bytes, words or blocks as well as it shows maximum allowed size are included.
6. **Protection:** This attribute determines the control access that who have read, write or execute.
7. **Usage Count:** It counts how many users can open the particular file.
8. **Time and Date:** In contains the creation, last modification time and date information related to the file.
9. **User information:** This attribute stores the information related to the user who create the particular file and how that file is accessible to other users.

7.3 File Support

7.3.1 File Types

In a design of a file system, consider whether the operating system should recognize and support file types. If the type of a file is recognize by an operating system, it can then operate on the file in a reasonable ways. A common method for implementing file types is to include the type as part of the file name. The name is divided into two parts-a name and an extension, usually separated by a period character. In this manner, the user and the operating system can tell from the name only what the type of a file is. Most operating systems allow users to define a file name as a sequence of characters followed by a period and terminated by an extension of additional characters. File name examples include result.doc, advance.java, and program. c.Following are some of the common file types with extension and itsfunctionality:

File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked

Source Code	C, 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, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
library	lib, a, so, dll	libraries o.troutines for .programmers

7.4 Access methods

Methods those provide reading information from a file and writing information to a file are known as Access Methods.The files stores information, so the information stored in the file can be accessed in different ways. Some of the system provide single access method for a file or some of the systems supports more than one access methods for a file. There are basically three different types of file access methods.

- Sequential Access
- Direct Access
- Indexed Access

7.4.1 Sequential Access

In sequential access, records are accessed in sequential manner. Sequential access method emulates magnetic tape models operations. This is the simplest method among these

three types of access methods, Information in the file is accessed in order that is one record is accessed after the other. Mostly used operations in sequential access are read and write.

Sequential Access supports the following three main operations:

i. Read next
ii. Write next
iii. Rewind

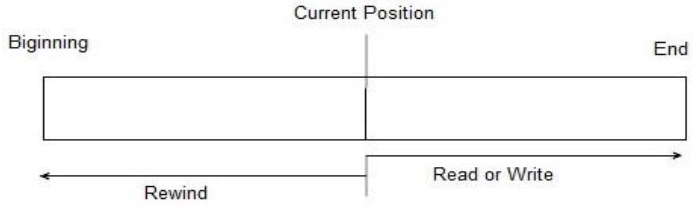


Fig 1.4.1 Sequential Access

In above figure, consider the file which contain the positions beginning of a file, end of a file and current position of a file.

- Read next- When file is going to read next operation then pointer moves next from the beginning position and where it ends the read operation which is the current position of a file.
- Write next- Write next operation appends information to the end of the file and moves end position to the new end of the file.
- Rewind- If the file pointer is currently at current position and if the user wants to move back to the beginning position then rewind operation performs.

7.4.2 Direct Access

The direct access is depends upon the disk model of a file since disk permits random access to any file block or a direct access to any of a file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and write on block 17. There is no limitation on the order of reading and writing for a direct access file. The disk are semi random access,this logical address is to be converted into three dimensional physical address and it can move read write head to that

record directly. Direct access files are useful for immediate access to large amount of information.

7.4.3 Indexed Access

In Index access of a file, initially there is a table which stores index of each record called as index file and data is stored into another file called as master file. When user want to search any record from such a file then user needs to give value of some key fields of the record, then operating system search that record into index file with the help of binary search method and find out address of that record from the index table. After searching the address of record it immediately shift read write head to that address in master file. There are two types of the indexed access files:

1. Indexed non sequential access

In this access method master file is not present in sorted order and entry of each record is in index table. When user wants to search any record, then operating system finds address using its key value from index table and move read write head to that position directly.

2. Indexed sequential

In this access method master file is present in some sorted order in some specific key and some key of the records is in index table. When user wants to search any record, then operating system finds address using its key value from index table and apply sequential search to find record from master table.

7.5 Allocation Methods

Files are stored on secondary storage devices such as disk by allocating a space for each file. As part of their implementation, files must be stored in the hard disk and space is utilized effectively and the files can be accessed quickly using different allocation methods. There are some of the major allocation methods which are as follows:

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

7.5.1Contiguous Allocation

In Contiguous allocation, each file occupies a contiguous set of blocks on the disk. Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block., For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* The directory entry for each file with this allocation includes

- Address of the starting block
- Length of the allocated area for the file.

In the following diagram, consider the file 'mail.' which starts from the 19th block and the length of this file is 6. So, the file occupies 6 blocks in a contiguous manner such as 19, 20, 21, 22, 23, 24 blocks.

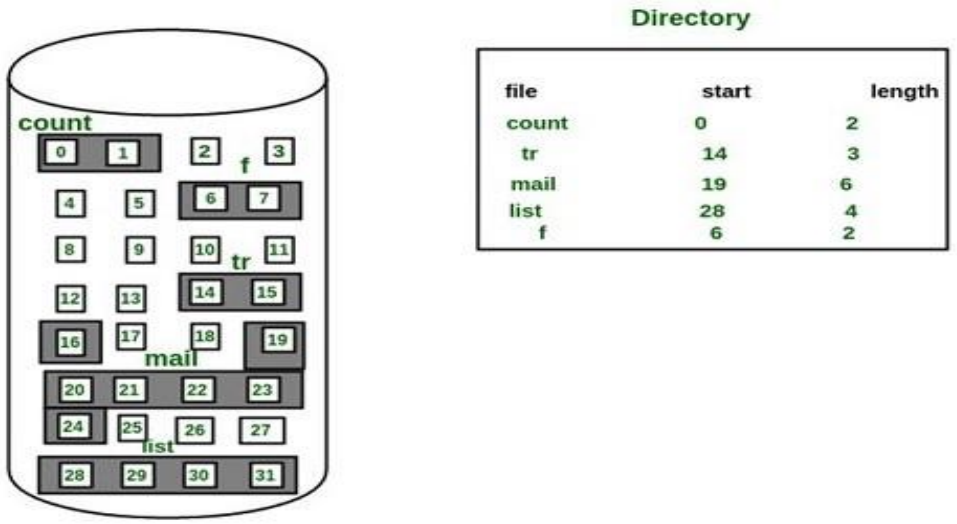


Fig 1.5.1 Contiguous Allocation

Advantages:

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

Disadvantages:

- This method suffers from both internal and external fragmentation. Therefore makes it inefficient in terms of memory utilization.

- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

7.5.2. Linked Allocation

In this type of allocation, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be distributed anywhere on the disk.The directory entry includes a pointer to the starting and the ending file block. Each block contains a pointer to the subsequent block occupied by the file.

The file 'jeep' in following diagram shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and doesn't point to the other block.

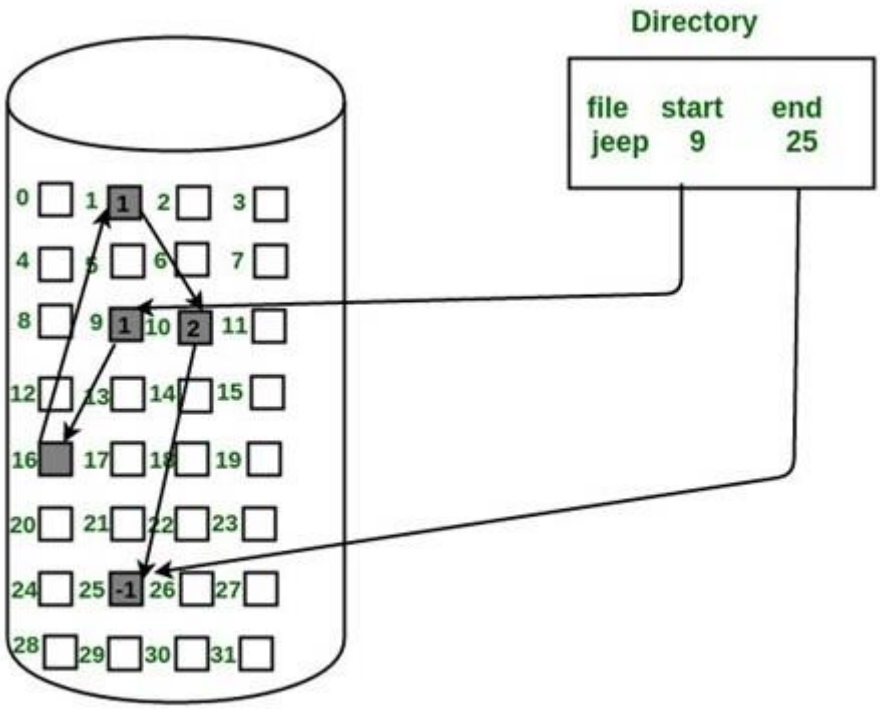


Fig 1.5.2. Linked Allocation

Advantages:

- This is very flexible in terms of file size. File size can be increased easily since the system doesn't need to search for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it comparatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can't directly access the blocks of a file. A block k of a file are often accessed by traversing k blocks serially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

7.5.3. Indexed Allocation

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

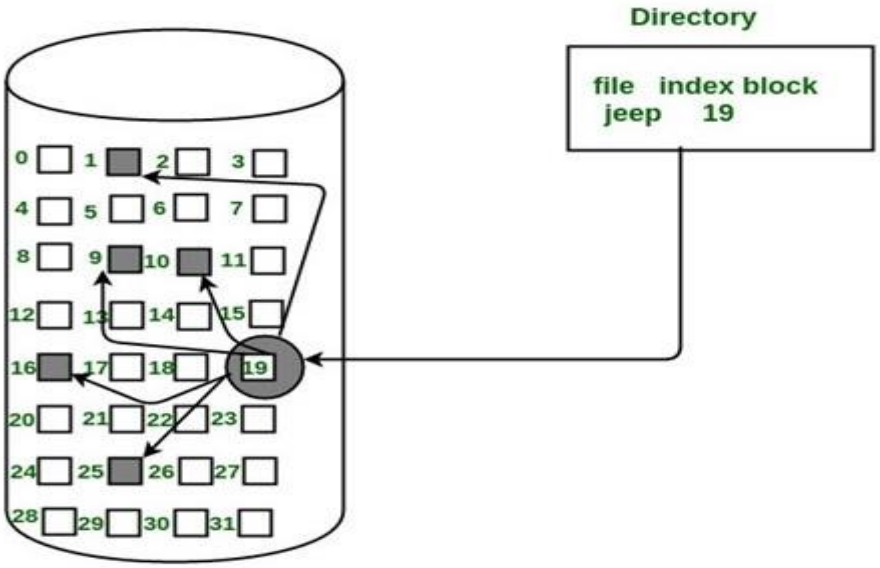


Fig 1.5.3. Indexed Allocation

Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of

memory utilization. However, in linked allocation we lose the space of just one pointer per block.

7.6Directory systems

A Directory is the collection of the correlated files on the disk. In other words, a directory is used to store file and folder. In a directory, we can store the complete file attributes or some attributes of the file. A directory consist of various files. With the help of the directory, we can maintain the information related to the files.

Every Directory supports a number of common operations on a directory:

1. Create file
2. Search for a file
3. Delete a file
4. Renaming a file
5. Traversing Files
6. List a directory

There are different directory structures available, these are given below.

7.6.1 Single-level directory

A single-level directory structure is the simplest directory structure. All the files are contained in same directory. It contain only one directory called as root directory which is used to store all the files present in the file system and it also store the files created by the several users. The directory contains one entry for each file present on the file system. Each file must have unique file name.There is no permission to users to create subdirectories under the root directory. This type of directory is given as follows:

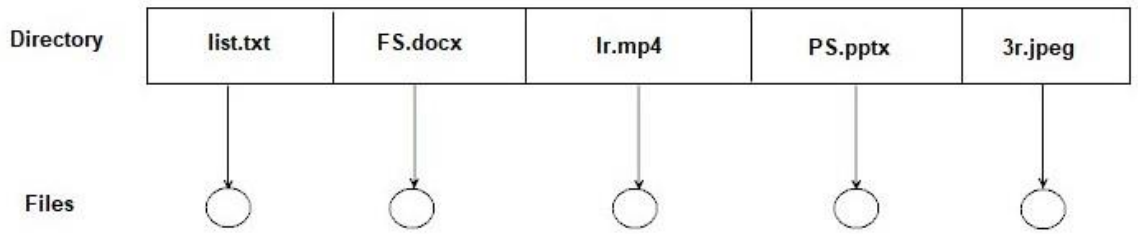


Fig 1.6.1 Single-level directory

Advantages:

- It is a single directory, so its implementation is very easy.
- If files are smaller in size, searching will faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- File require unique file name.
- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if directory will large.
- In this directory structure, can not group the same type of files together.

7.6.2 Two-level directory

Operating system creates a first level directory which is called as root directory or Master File Directory. After that user can create their own directory under the root directory. In two level directory structure, Usercan create a separate directory. It allows each user to create their files separately inside their own directory. Users does not allow to access other users directories. But this structure allows to use the same name for the files but under different user directories. This type of directory is shown in following figure:

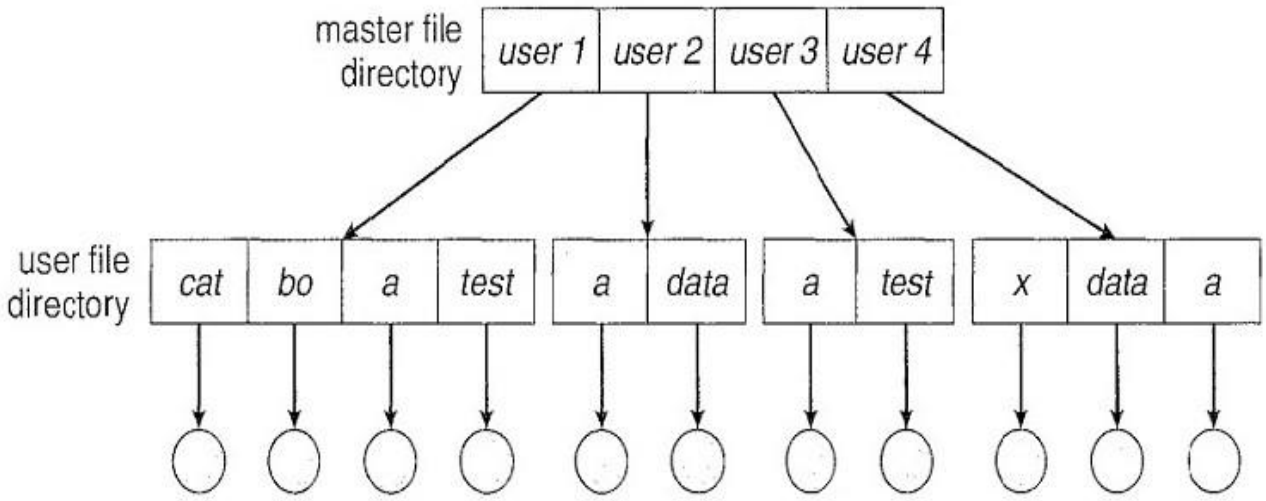


Fig 1.6.2 Two-level directory

Advantages:

- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages:

- A user is not allowed to share files with other users.
- Still it not very scalable, two files of the same type cannot be grouped together in the same user.

7.6.3 Tree-structured directory

Tree structured directory overcomes the disadvantages of two level directory system. In Tree structured directory, any directory entry can either be a file or sub directory. This directory structure allows users to create their own directories, subdirectories and arrange files accordingly. In tree directory structure, all directories have the same internal format. It enters one bit in each directory to define the entry as a file or as a subdirectory. If the entered bit is 0 then it is a file and if entered bit is 1 then it is subdirectory.

In tree directory structure, Files can be accessed using either

- Absolute pathnames-It is relative to the root of the tree.
- Relative pathnames - It is relative to the current directory.

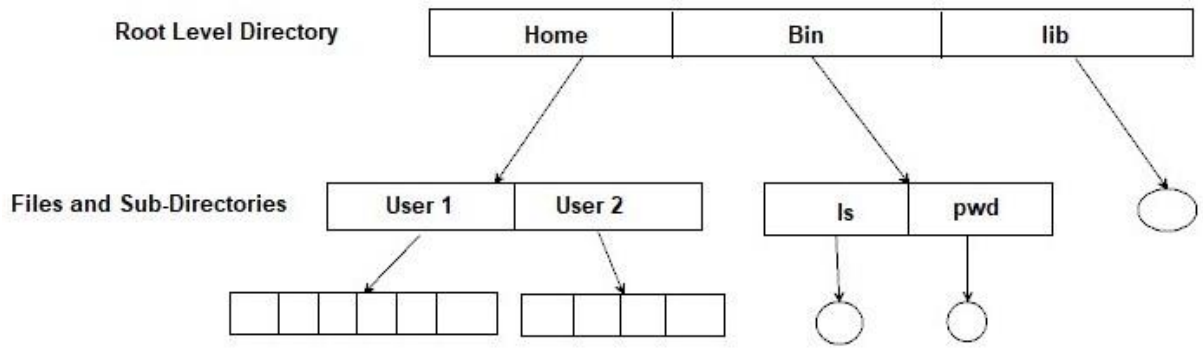


Fig 1.6.3 Tree-structured directory

Advantages:

- It overcomes the drawback of two level directory structure.
- Users are completely independent.
- Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

- Every file does not fit into the tree structure, files may be saved into multiple directories.
- Files are not share.
- Duplicate copies of the file are to be created.

7.6.4 Acyclic graph directory

A graph with no cycle and allows to share subdirectories and files is called as an acyclic graph directory. The same file or subdirectories may be in two different directories. This is a natural generalization of the tree-structured directory.

An acyclic graph structure is usefulwhen the same files need to be accessed in more than one place in the directory structure.In this files are shared by more than one user.

This directory structure is used in the situation like when two programmers are working on a same project and they need to access files. The associated files are stored in a

subdirectory, separating them from other projects and files of other programmers, since they are working on a same project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use acyclic directories.

Advantages:

- Files are share.
- Searching is easy.

Disadvantages:

- If the link is softlink then after deleting the file we left with a dangling pointer.
- In case of hardlink, to delete a file we have to delete all the reference associated with it.

7.7File Protection

Information is stored into a computer system in a secondary storage devices. We need to provide the security when we store an information on a secondary storage devices such as:

- Need to protect that information from physical damage such as the issue of reliability. User can achieve reliability by keeping multiple back-up copies of the data in a secondary storage devices.
- Need to protect data from improper access such as the issue of protection. File protection deals with protecting the data which is stored into the computer system from improper access

File systems can be scratched by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs within the file-system software also can cause file contents to be lost.

7.7.1 Types of Access

Protection mechanisms offers controlled access by limiting the types of file access that can be made. Types of access gives restricted access to the users. Several different types of operations are there such as:

• Read. It gives access to read information from the file or not.

• Write. Whether the users are allowed to write or rewrite the file or not.

• Execute. Whether the user is able to load the file into memory and execute it or not.

• Append. Adding new information at the end of the file.

• Delete. Delete the file and free its space for another reuse.

• List. We should be able to list the name as well as the attributes of the file.

7.7.2Access Control List

One way in which protection isprovided or a common approach which is followed for a protection is an Access Control List (ACL).Access Control List is used to provide access based on the identity of a user. To base on the type of a user the access can be provide.

With each file or with each directory an Access Control List (ACL) is being associated that will contain information about what are all the types of access, which is allowed for all types of users. In short, this Access Control List contains information about the user name and types of access allowed for each user. When a user requests access to a particular file or a directory, the operating system checks the access control list associated with that file or a directory. 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.

Based on this, mostly operating systems provides three types of classification of users for every file or a directory. Users are classified into three different types:

• Owner. Owner is the user who created the file.

• Group. A set of users who are sharing the file and need similar access is a group, or work group.

• Universe. All other users other than the Owner and group users comes under universe.

7.7.3 Protection Bits

Protection bits are also used to define type of access associated with each file. Each field is a collection of bits, and each bit either allow or prevents the access associated with it. For example, the UNIX system three protection bits such as –rwx, where r stands for readaccess, w stands for write access and x stands for executes access. If that particular bit is

set that means the user is allowed to do the operation on that particular file. And if that particular bit is not set that means the user is not allowed to do the operation on the particular file. Separate –rwx field is kept for each type of user such as file owner, file's group and for all other users. That means separate 9 bits will be set for the three categories of the users.

7.8 Free Space Management

There are different ways to implement a free space. That are as follows:

- Bit map or Bit vector
- Linked List
- Grouping
- Counting

7.8.1 Bit map or Bit vector

Bit map is also called as bit vector. This is very simple method used for free space management. In this implementation of a free space, each block is represented by one bit. If the block have a free space or used space it is shown by a bit. If the block is free, the bit is 1 and if the block is allocated, the bit is 0.

For example: Consider a disk where the blocks 2,3,4,6,7,9,11,12,13,18,20 are free and the remaining blocks are allocated. The free space bit vector would be as follows:

00111011010111000010 1

7.8.2 Linked List

Linked list is another approach of free-space management used to link all the free blocks together. It keeps a pointer to the first free block. This first free block keeps a pointer to the next free block, and so on.

Consider the example, where the blocks 2,3,4,6,7,9,10 ,13,18,19 are free and the remaining blocks are allocated as shown in figure:

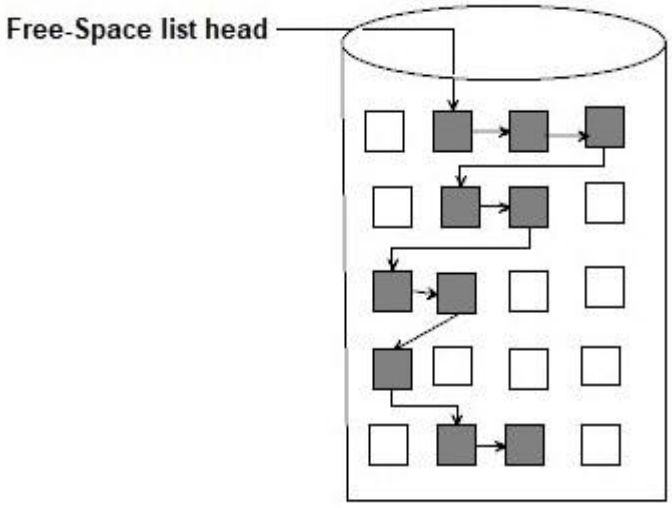


Fig 1.8.2 Linked List

In this example, it keeps a pointer to block 2 which is the first free block. Block 2 contain a pointer to block 3, which would contain a pointer to block 4, which would contain a pointer to block 6, and so on. In this method there is a need to read each block to traverse the list.

7.8.3 Grouping

Grouping approach stores the n free block's addresses in the first free block. From this blocks, the first n-1 blocks are actually free and the last block is used to store the addresses of another n free blocks, and so on. Using grouping, the addresses of a large number of free blocks can be found easily.

7.8.4 Counting

In a situation where some contiguous blocks may be allocated or freed simultaneously. Thus instead of keeping a list of n free disk addresses, we can keep the address of the first free block 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.

7.9 Summary

The one of the major functionality of an operating system is a file system. Operating system contains a module file system which manages the files. It manages the file

that how to store the data and how to fetch the data from a file. It is used to control each and every activities regarding files which store on secondary memory such as hard disk. File is basically a form of data that computer understands or computer can interpret.Different system calls used to create, read, write, reposition, delete, truncate files.To identify name and location of a file, file attributes are used.

Access methods are used to provide authorization for reading information from a file and writing information to a file. Sequential access, Direct access and Indexed access types are used to provide the access to a file.

Files must be stored in the hard disk and space is utilized effectively and the files can be accessed quickly using different allocation methods.

A Directory is the collection of the correlated files on the disk. In other words, a directory is used to store file and folder. In a directory, we can store the complete file attributes or some attributes of the file. A directory consist of various files. With the help of the directory, we can maintain the information related to the files.

When information is stored in a computing system, we would like to stay it safe from physical damage (the issue of reliability) and improper access (the issue of protection).Protection are often provided in several ways.

7.10 Bibliography

1) Silberschatz, Galvin, Gagne, Operating System Concepts, Ninth edition, U.K: John Wiley and Sons Inc., 2000.
2) Andrew S Tanenbaum, Modern Operating Systems, Prentice Hall India, 1992.

7.11 Unit End Exercises

1) What is a file? Explain different file attributes.
2) Explain different file operation.
3) What are the different types of Access methods? Explain in detail.
4) Explain different types of Allocation Methods.
5) What is Directory system? Explain Single-level and Two-level Directory structure.
6) What is Directory system? Explain Tree-structured and Acyclic Graph directory structure.
7) Write a note on File protection.
8) Describe free space management in detail.

Unit 7 - Chapter 8

Protection

Unit Structure

8.0 Objectives

After studying this unit, you will be able to:

- Know system protection.
- Describe various goals of protection.
- Discuss the principles of protection in a modern computer system.
- Explain how protection domains combined with an access matrix.
- Explain access matrix and its implementation.
- Define access control.
- Describe capability-based system.

8.1 Introduction

- The processes in an operating system must be protected from one another's activities. To provide such protection, we can use various mechanisms to ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory segments, CPU, and other resources of a system.
- **Protection** refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

    - The OS ensures that all access to system resources is controlled.
    - The OS ensures that external I/O devices are protected from invalid access attempts.
    - The OS provides authentication features for each user by means of passwords.

8.2 Goals 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 authorized and unauthorized usage.
- Role of **protection** in a computer system is to provide a **mechanism** for the enforcement of the **policies** governing resource use.
- Mechanisms determine how something will be done and policies decide what will be done.

8.3 Principles of Protection

- Guiding principle for protection is the **Principle of Least Privilege**. It dictates that programs, 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.
- Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs. An operator who needs to mount tapes and backup files on the system has access to just those commands and files needed to accomplish the job. Some systems implement **role-based access control (RBAC)** to provide this functionality.

8.4 Domain of Protection

- A computer system is a collection of processes and objects such as **hardware objects** like CPU, memory segments, printers, disks, and tape drives and **software objects** like files, programs, and semaphores.
- 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 its task. This is referred as **Need-to-Know** principle. It limits the amount of damage caused by faulty process.

1.4.1 Domain Structure

- 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 an operation on an object is called **Access Right.**

- A **domain** is a collection of access rights. It is denoted by **ordered pair - <object-name, right-set>**. **For example,** if domain D has the access right <file F, {read, write}>, then a process executing in domain D can both read and write file F and it cannot perform any other operation on that object.
- Domains do not need to be disjoint; they may **share access rights**. **For example**, in **below figure** we have **three domains: D_1, D_2, and D_3**. The access right < 0_4, {print}> is shared by D_2 and D_3, implying that a process executing in either of these two domains can print object 0_4.

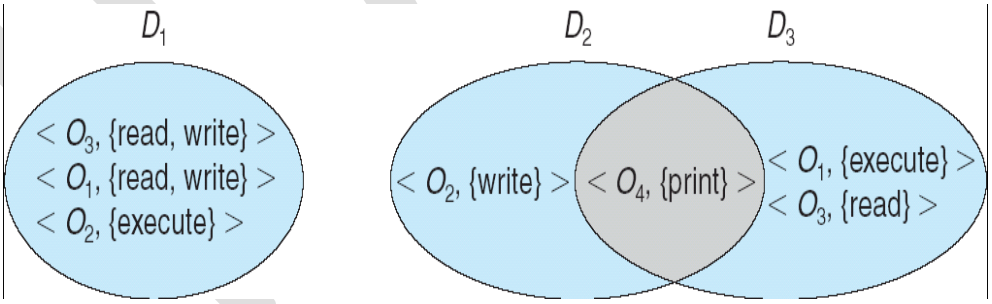


figure.(1)

- **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:** In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.

- **Each procedure may be a domain:** In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

- **An example - UNIX**

  - In UNIX Operating system, domain is related with the user. S**witching the domain** corresponds to changing the user identification temporarily.
  - Owner **identification and a domain bit (known as the setuid bit)** are associated with each file. When the **setuid bit is on**, and a user executes that file, the user ID is set to that of the owner of the file, but when the bit is **off,** the user ID does not change.
  - **Other methods** are used to change domains in operating systems in which user IDs are used for domain definition, because almost all systems need to provide such a mechanism.
  - An alternative to this method used in other operating systems is to place **privileged programs** in a special directory. The operating system would be designed to **change the user lD** of any program run from this directory, either to the equivalent of root or to the user lD of the owner of the directory.
  - Even more restrictive, and thus more protective, are systems that simply do not allow a change of user ID. In these instances, special techniques must be used to allow users access to privileged facilities. For instance, a **daemon process** may be started at boot time and run as a special user ID.
  - In any of these systems, great care must be taken in writing privileged programs.

- **An example – MULTICS**

  - Protections of domains are organized hierarchically into a **ring structure**. Rings are numbered from 0 to ring N-1. Each ring is a single domain as shown in **figure.(2)**

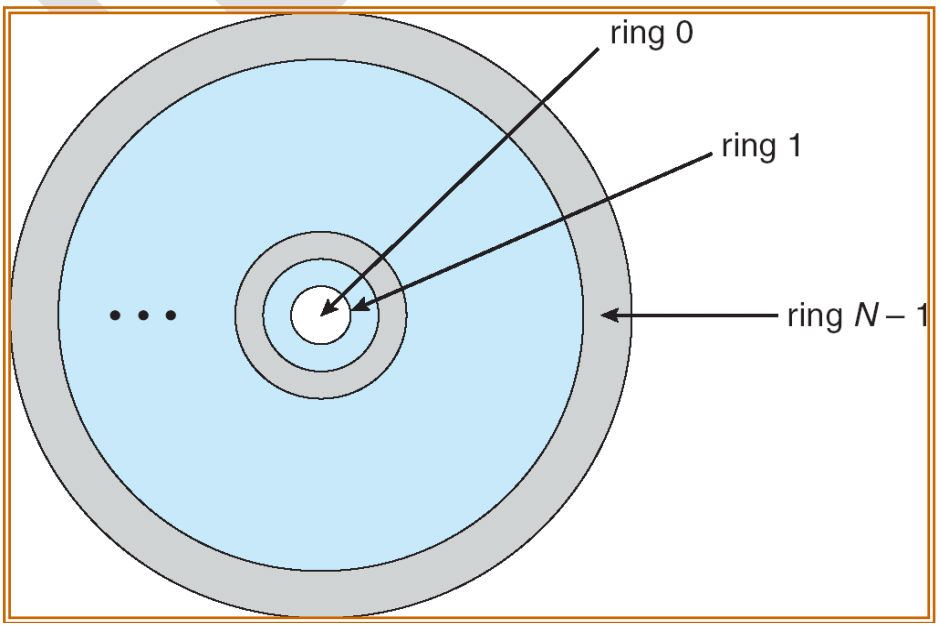


figure.(2)

- 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 the process executing in domain D_i. **Ring 0 has full privileges.**
- MULTICS has a **segmented address space**; each segment is a file, and each segment is associated with one of the rings. A segment description includes an entry that identifies the ring number and **three access bits** to control reading, writing, and execution.
- A current-ring-number counter is associated with each process, identifying the ring in which the process is executing currently. When a process is executing in ring i, it cannot access a segment associated with ring j (j <i). It can access a segment associated with ring k (k>= i). The type of access is restricted according to the access bits associated with that segment.
- **Domain switching** in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring. This switch must be done in a controlled manner; otherwise, a process could start executing in ring 0, and no protection would be provided.

- To allow **controlled domain switching,** we modify the ring field of the segment descriptor to include the following:
  - **Access bracket**. A pair of integers, bl and b2, such that bl <= b2.
  - **Limit.** An integer b3 such that b3 >b2.
  - **List of gates.** Identifies the **entry points** (or **gates)** at which the segments may be called.
- If a process executing in ring i calls a procedure (or segment) with access bracket (bl,b2), then the call is allowed if **bl <= i <= b2,** and the current ring number of the process remains i. Otherwise, a **trap** to the operating system occurs, and the situation is **handled as follows**:
  - If i < bl, then the call is allowed to occur, because we have a transfer to a ring (or domain) with fewer privileges. If parameters are passed that refer to segments in a lower ring then these segments must be copied into an area that can be accessed by the called procedure.
  - If i > b2, then the call is allowed to occur only if b3 is greater than or equal to i and the call has been directed to one of the designated entry points in the list of gates. This scheme allows processes with limited access rights to call procedures in lower rings that have more access rights, but only in a carefully controlled manner.

- The main **disadvantage** of the ring structure is that it does not allow us to enforce the **need-to-know principle.** The MULTICS protection system is generally more **complex and less efficient**.

8.5 Access Matrix

- The model of protection can be viewed abstractly as a matrix, called an **access matrix.**
- The **rows** of the access matrix represent **domains**, and the **columns** represent **objects**. 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.

- **For Example,** consider the access matrix shown in below **figure (3)**
- 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 D_1 can read file F_1 and file F_3.
  - Process in domain D_2 can only use printer.
  - Process in domain D_3 can read file F_2 and execute file F_3.
  - Process in domain D_4 can read and write file F_1 and file F_3.

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

figure.(3)

- Access matrix scheme provides us with the mechanism for specifying a variety of policies. We must ensure that a process executing in domain D_i, can access only those objects specified in row, and then only as allowed by the access-matrix entries. 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)**. The below **figure (4)** shows the access matrix with domains as objects. Process can change domain as follows,
  - Process in domain D_2 can switch to domain D_3 and domain D_4.
  - Process in domain D_4 can switch to domain D_1.
  - Process in domain D_1 can switch to domain D_2.

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 (4)

- Access matrix is inefficient for storage of access rights in computer system because they tend to be large and sparse.
- Allowing controlled change in the contents of the access-matrix entries requires **three additional operations: copy, owner, and control.**
- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an **asterisk (*)** appended to the access right.
- The **copy** right allows the access right to be copied only within the column for which the right is defined.
- **For example,** as shown in below **figure(5) - figure (a)**, a process executing in domain D_2 can copy the read operation into any entry associated with file F_2. Hence, the access matrix of **figure (a)** can be modified to the access matrix shown in **figure (b).**

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

(a)

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

(b)

figure(5)

- This scheme has **two variants:**
  - A right is copied from access(i, j) to access(k, j); it is then removed from access(i, j). This action is a transfer of a right, rather than a copy.

- Propagation of the copy right may be limited. That is, when the right R* is copied from **access(i,j) to access(k,j), only the right R (not R*)** is created. A process executing in domain D_k cannot further copy the right R.

- A system may select only one of these three copy rights, or it may provide all three by identifying them as separate rights: copy, transfer, and limited copy.
- The **owner right controls these operations.** If access(i, j) includes the owner right, then a process executing in domain D_i can add and remove any right in any entry in column j.

- **For example,** as shown in below **figure(6) - figure (a)** domain D_1 is the **owner of F_1** and thus can add and delete any valid right in column F_1. Similarly, domain D_2 is the owner of F_2 and F_3 and thus can add and remove any valid right within these two columns. Thus, the access matrix of **figure (a)** can be modified to the access matrix as shown in **figure (b).**

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

(a)

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

(b)

figure(6)

- The copy and owner rights allow a process to **change the entries** in a column.
- A mechanism is also needed to change the entries in a row. The **control** right is applicable only to domain objects. If access(i, j) includes the **control** right, then a process executing in domain D_i can remove any access right from row j.
- **For example,** in **figure (7)** we include the control right in access (D_2, D_4). Then, a process executing in domain D_2 could modify domain D_4, as shown in below **figure.**
- The **problem** of guaranteeing that no information initially held in an object can migrate outside of its execution environment is called the **confinement problem**. This problem is in **general unsolvable.**

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

figure (7)

8.6 Implementing Access matrix

- It is implemented in several ways. Methods for implementing access matrix are,
  A. Global table.
  B. Access lists for objects.
  C. Capability list for domain.
  D. A lock key mechanism.

A] Global Table

- It is 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 M** is executed on an object O_j within domain D_i, the global table is searched for a triple- $< D_i, O_j, R_k >$ with M € R_k. If the above triple is found, then operation is allowed to continue. If suppose triple is not found then an exception error condition occurs.
- **Limitations of Global table are,** Global table is large and it cannot be kept in memory and additional Input/ Output is required.

B] Access list for objects

- Each column in the access matrix can be implemented as an access list for one object. The empty entries can be discarded.
- The resulting list for each object consists of **ordered pairs <domain, rights-set>,** which define all domains with a **nonempty** set of access rights for that object.
- This approach can be extended easily to define a list plus a default set of access rights. When an operation M on an object O_i is attempted in domain D_i, we search the access list for object O_i, looking for an entry $< D_i, R_k >$ with M €

R_k. If the entry is found, we allow the operation; if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs.

C] Capability list for domains

- 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.**
- Process executes operation M by specifying the capability (or pointer) for object O_j 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.

D] A Lock –Key Mechanism

- The lock key scheme is a compromise between access 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 of methods**

        - 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.

8.7 Access Control

- **Role-based Access control (RBAC)** 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 below **figure (8)**

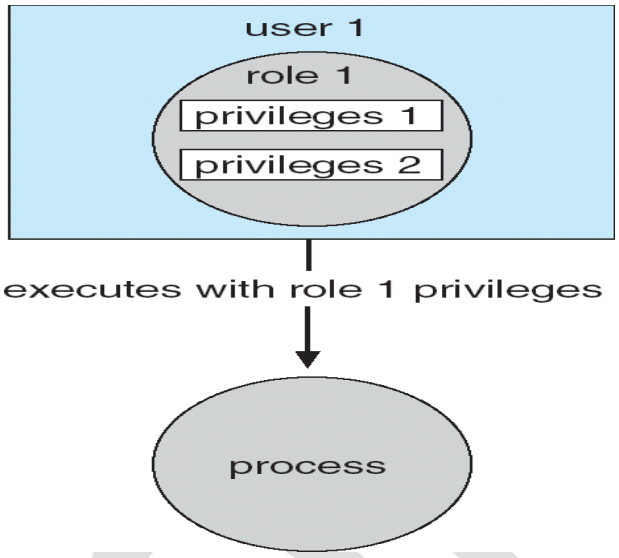


figure (8)

8.8 Revocation Of Access Rights

- Revocation of access rights to objects in shared environment is possible. Various questions about revocation may arise as follows,
  - **Immediate versus delayed.** Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?
  - **Selective versus general.** When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a **select group of users whose access rights should be revoked?**
  - **Partial versus total.** Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
  - **Temporary versus permanent.** Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?
- Revocation is **easy for access list and complex for capabilities list.** The access list
  is searched for any access rights to be revoked, and they are deleted from the list.
- **Schemes** that implement revocation for capabilities include the following:
  - **Reacquisition.** Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability.
  - **Back-pointers.** A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, change the capabilities as necessary. This scheme was adopted in the MULTICS system. It is quite general, but its implementation is costly.
  - **Indirection.** The capabilities point indirectly to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to

point to an illegal table entry. Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object. The object for a capability and its table entry must match. This scheme was adopted in the CAL system. It does not allow selective revocation.

- **Keys.** A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability. A **master key** is associated with each object; it can be defined or replaced with the **set-key** operation. When a capability is created, the current value of the master key is associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised. If we associate a list of keys with each object, then selective revocation can be implemented. Finally, we can group all keys into one **global table** of keys. A capability is valid only if its key matches some key in the global table. In key-based schemes, the operations of defining keys, inserting them into lists, and deleting them from lists should not be available to all users.

8.9 Capability Based Systems

Capability based protection systems are of two types,

- **An example - Hydra**
  - Hydra provides a fixed set of possible access rights that are known to and interpreted by the system. These rights include such basic forms of access as the right to read, write or execute a memory segment.
  - Operations an object are defined procedurally. The procedures that implement such operations are themselves a form of object and they are accessed indirectly by capabilities. When the definition of an object is made known to hydra, the names of operations on the type become **auxiliary right.**
  - Hydra also provides **rights amplification.** This scheme allows certification of a procedure as trust worthy to act on an formal parameter of a specified type, on behalf of any process that holds a right to execute the procedure.
- **An example - Cambridge Cap System**
  - CAP system is simpler and superficially less powerful than that of hydra. CAP has two kinds of capabilities-
    - **Data capability** can be used to provide access to objects, but the only rights provided are the standard read, write and execute of the individual storage segments associated with the object. Data capabilities are interpreted by microcode in the CAP machine.
    - **Software capability** is protected, but not interpreted by the CAP microcode. It is interpreted by a protected procedure, which may be written by an application programmer as part of a subsystem. The interpretation of a software capability is left completely to the subsystem, through the protected procedures it contains. This scheme allows a variety of protection policies to

8.10 Summery

Computer system contains many objects, and they need to be protected from misuse. Objects may be hardware (such as memory, CPU time, and I/O devices) or software (such as files, programs, and semaphores). An access right is permission to perform an operation on an object. A domain is a set of access rights. Processes execute in domains and may use any of the access rights in the domain to access and manipulate objects. During its lifetime, a process may be either bound to a protection domain or allowed to switch from one domain to another.

The access matrix is a general model of protection that provides a mechanism for protection without imposing a particular protection policy on the system or its users. The separation of policy and mechanism is an important design policy.

Real systems are much more limited than the general model and tend to provide protection for files.

8.11 Model Question

1. What do you mean by system protection?
2. Describe various goals of security and protection.
3. Explain access matrix. Also describe its implementation.
4. What do you mean by access control?
5. Describe various techniques of access controls.
6. Explain Domain Structure?

8.12 List of References

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/14_Protection.html

https://www.geeksforgeeks.org/system-protection-in-operating-system/

https://www.faceprep.in/operating-systems/operating-systems--security-and-protection/

http://www.uobabylon.edu.iq/download/M.S%202013-2014/Operating_System_Concepts,_8th_Edition%5BA4%5D.pdf

Unit 7 - Chapter 9
Security

Unit Structure

9.0 Objectives

After studying this unit, you will be able to:

- Define system security
- Explain Program, system and network threats
- Describe user authentication
- Explain One-Time passwords

9.1 Introduction

- Protection is strictly internal problem: how do we provide controlled access to program and data stored in computer system.

- **Security on other hand requires not only an adequate protection system but also consideration of the external environment within which the system operates.**
- Computer system must be guarded against unauthorised access, malicious destruction or alteration, and accidental introduction of inconsistency.

9.2 Goals of Security:

In general there are five goals of security:

Integrity: Data integrity is maintained. It is ensured that data is of high quality, correct, consistent and accessible. Verifying data integrity consists in determining if the data were changed during transmission.

Confidentiality: It is ensured that only the authorized parties have access to the resources of the system.

Availability: The goal of availability is to ensure that authorized parties can have the access to a service or resource.

Non-repudiation: The non-repudiation is the guarantee that an operation cannot be denied at a later date.

Authentication: The goal of authentication is to confirm a user's identity that means access control grants access to resources only to authorized individuals or parties.

9.3 The Security Problem

- We say that system is secure if its resources are used and accessed as intended under all circumstances.
- Unfortunately total security cannot be achieved, nonetheless , we must have mechanisms to make security breaches a rare occurrences, rather than the norm.
- Security violations of the system can be categorised as intentional or accidental.
- Intruders (crackers) are those who attempt to breach security
- **Threat** is potential security violation.
- **Attack** is an attempt to breach security.
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

- Some of the most common types of ***violations*** include:
    - ***Breach of Confidentiality*** - Theft of private or confidential information, such as credit -card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.

    - ***Breach of Integrity*** - Unauthorized ***modification*** of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.

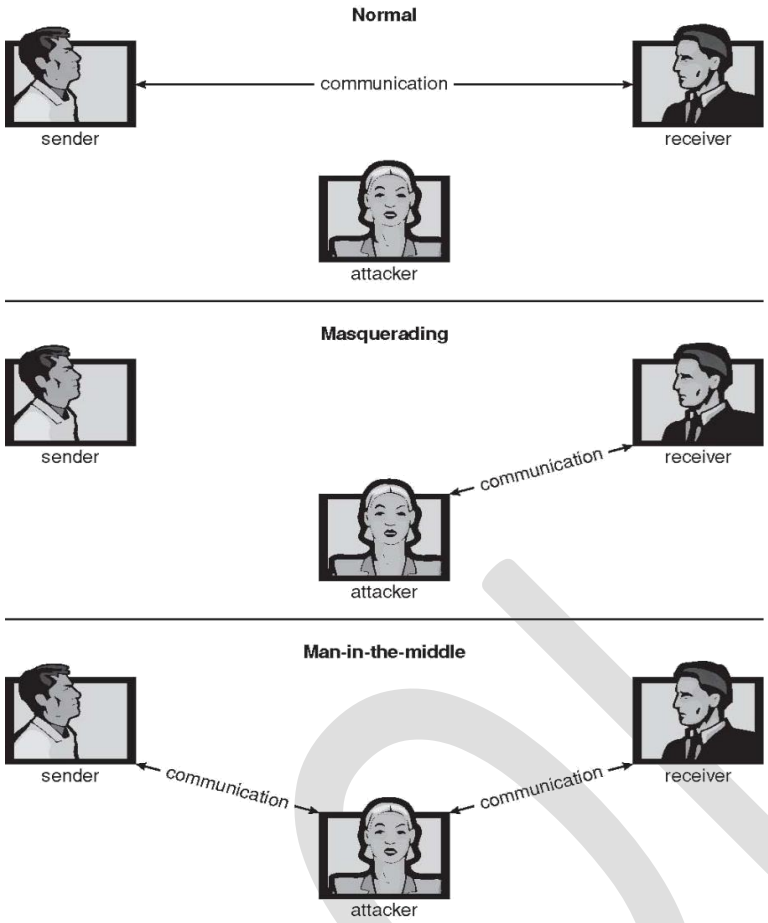
    - ***Breach of Availability*** - Unauthorized ***destruction*** of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.

    - ***Theft of Service*** - Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.

    - ***Denial of Service, DOS*** - Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.

Attackers use several standard methods in their attempts to breach security.

- One common attack is ***masquerading,*** in which the attacker pretends to be a trusted third party.
- A variation of this is the ***man-in-the-middle,*** in which the attacker masquerades as both ends of the conversation to two targets. In this attacker sits in the data flow of a communication.



Figure(1)

There are four levels at which a system must be protected:

1. **Physical -** The easiest way to steal data is to pocket the backup tapes. Also, access to the root console will often give the user special privileges, such as rebooting the system as root from removable media. Even general access to terminals in a computer room offers some opportunities for an attacker, although today's modern high-speed networking environment provides more and more opportunities for remote attacks.

2. **Human -** There is some concern that the humans who are allowed access to a system be trustworthy, and that they cannot be coerced into breaching security. However more and more attacks today are made via ***social engineering,*** which basically means fooling trustworthy people into accidentally breaching security.

- **Phishing** involves sending an innocent-looking e-mail or web site designed to fool people into revealing confidential information. E.g. spam e-mails pretending to be from e-Bay, PayPal, or any of a number of banks or credit-card companies.
- **Dumpster Diving** involves searching the trash or other locations for passwords that are written down. (Note: Passwords that are too hard to remember, or which must be changed frequently are more likely to be written down somewhere close to the user's station.)
- **Password Cracking** involves divining users passwords, either by watching them type in their passwords, knowing something about them like their pet's names, or simply trying all words in common dictionaries. (Note: "Good" passwords should involve a minimum number of characters, include non-alphabetical characters, and not appear in any dictionary (in any language), and should be changed frequently. Note also that it is proper etiquette to look away from the keyboard while someone else is entering their password.)

3. **Operating System -** The OS must protect itself from security breaches, such as runaway processes (denial of service), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.
4. **Network -** As network communications become ever more important and pervasive in modern computing environments, it becomes ever more important to protect this area of the system. (Both protecting the network itself from attack, and protecting the local system from attacks coming in through the network.) This is a growing area of concern as wireless communications and portable devices become more and more prevalent.

9.4 Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic.

So, unauthorized access can be prevented by this process. But there are some people called hackers who are always trying to break the authentication mechanism. Now there are different types of authentication methods available as discussed as follows:

9.4.1. Authentication Using Passwords

- Passwords are the most common form of user authentication. If the user is in possession of the correct password, then they are considered to have identified themselves.

- In theory separate passwords could be implemented for separate activities, such as reading this file, writing that file, etc. In practice most systems use one password to confirm user identity, and then authorization is based upon that identification.

9.4.1.1 Better password Security with strong password

A cracker always tries to guess passwords one at a time. So educating users about the need for strong passwords is very essential. Computer help can also be used for this purpose. Some computers have a program that generates some random words which does not have any meaning, such as ttdhdy, jjdghr etc. that can be used as passwords. The program that users call to install or change their password can also give a warning when a poor password is chosen. For example: some warnings are like the followings

1. Passwords should be a minimum of seven characters.
2. Passwords should contain both upper and lower case letters.
3. Passwords should contain at least one digit or special character.
4. Passwords should not be dictionary words, people's names, etc.

Some operating systems require users to change their passwords regularly, to limit the damage done if a password leaks out. But if users have to change their password too often then it may be possible that they start picking weak ones.

9.4.1.2 One-Time Password

One-time password is a password authentication approach where the user gets a book containing a list of passwords. Each login uses the next password in the list. If an intruder ever discovers a password then also it will not help him to break the system, since next time a different password must be used. It is suggested that the user try to avoid losing the password book.

9.4.1.3 Challenge-Response Authentication

In this authentication scheme, each new user provide a long list of questions and answers that are then stored on the server securely in encrypted form. At login, the

server asks one of them at random and checks the answer. To make this scheme practical, many question-answer pairs would be needed.

9.4.2 Authentication Using a Physical Object

The second method for authenticating users is to check for some physical object they have. Nowadays, the physical object used is often a plastic card that is inserted into a reader associated with the terminal or computer. Normally, the user must not only insert the card, but also type in a password, to prevent someone from using a lost or stolen card. For example: using a bank's ATM (Automated Teller Machine) starts out with the user logging in to the bank's computer via a remote terminal (the ATM machine) using a plastic card and a password.

The big advantage of smart cards over, say, credit or debit cards, is that they do not need an online connection to a bank.

9.4.3 Authentication Using Biometrics

- Biometrics involve a physical characteristic of the user that is not easily forged or duplicated and not likely to be identical between multiple users.
- Fingerprint scanners are getting faster, more accurate, and more economical.
- Palm readers can check thermal properties, finger length, etc.
- Retinal scanners examine the back of the users' eyes.
- Voiceprint analyzers distinguish particular voices but it difficulties may arise in the event of colds, injuries, or other physiological changes.

9.5 One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time passwords are implemented in various ways.

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.

- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.

- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

- denial of service attacks browser's content settings.

9.6 Threats

Threats can be classified into the following two categories:

1. **Program Threats:**
   A program written by a cracker to hijack the security or to change the behaviour of a normal process.
2. **System and Network Threats:**
   These threats involve the abuse of system services. They strive to create a situation in which operating-system resources and user files are misused. They are also used as a medium to launch program threats.

9.6.1 Program Threats

- There are many common threats to modern systems. Only a few are discussed here.

1) Trojan Horse

- A ***Trojan Horse*** is a program that secretly performs some maliciousness in addition to its visible actions.
- Some Trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with ***viruses,***(see below.)
- One dangerous opening for Trojan horses is long search paths, and in particular paths which include the current directory (".") as part of the path. If a dangerous program having the same name as a legitimate program (or a common mis-spelling, such as "sl" instead of "ls") is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake.
- Another classic Trojan Horse is a login emulator, which records a users account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again (with a proper login prompt), logs in successfully, and doesn't realize that their information has been stolen.

- ***Spyware*** is a version of a Trojan Horse that is often included in "free" software downloaded off the Internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. (This is an example of ***covert channels,*** in which surreptitious communications occur.) Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user.

2) Trap Door

- A ***Trap Door*** is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system.
- Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door.
- A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

3) Logic Bomb

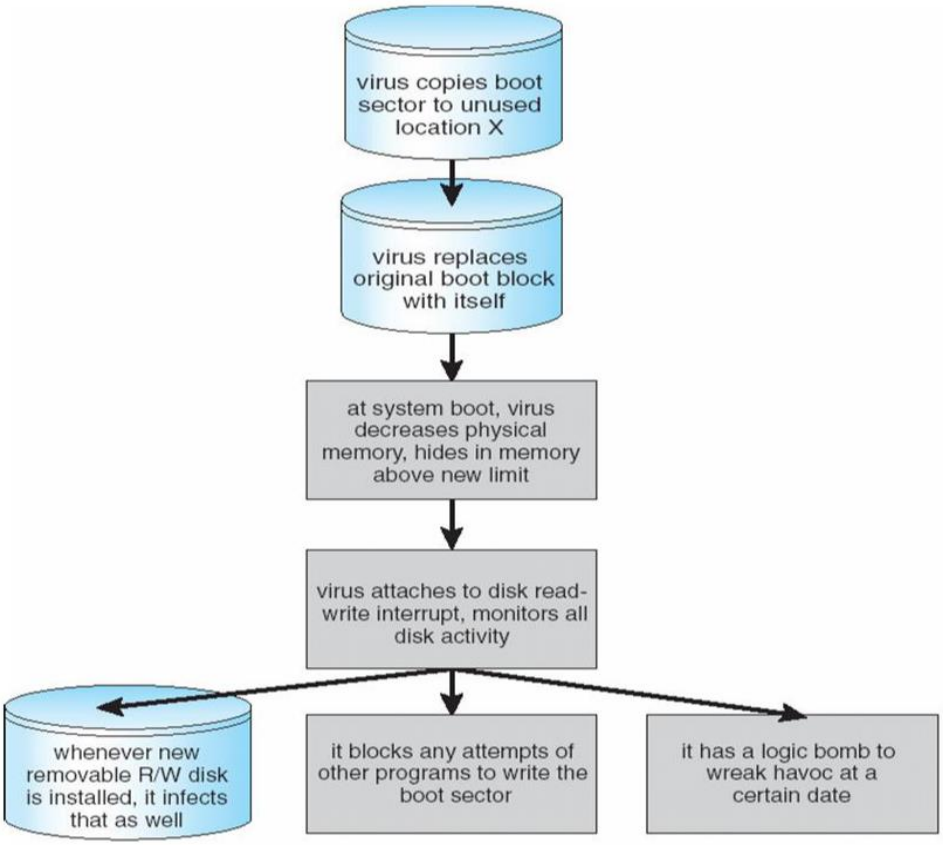
- A ***Logic Bomb*** is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.
- A classic example is the ***Dead-Man Switch***, which is designed to check whether a certain person (e.g. the author) is logging in every day, and if they don't log in for a long time (presumably because they've been fired), then the logic bomb goes off and either opens up security holes or causes other problems.

4) Stack and Buffer Overflow

- This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow. Consider what happens in the following code, for example, if argv[1] exceeds 256 characters:
  - The strcpy command will overflow the buffer, overwriting adjacent areas of memory.
  - (The problem could be avoided using str***n***cpy, with a limit of 255 characters copied plus room for the null byte.)

5) Viruses

- A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.
- Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures (such as the boot block.)
- Viruses are delivered to systems in a ***virus dropper,*** usually some form of a Trojan Horse, and usually via e-mail or unsafe downloads.
- Viruses take many forms (see below.) Figure shows typical operation of a boot sector virus:



Figure(2)

Some of the forms of viruses include:

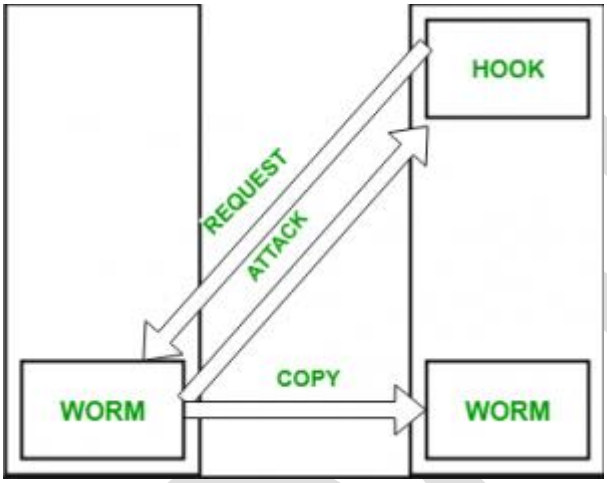
- **File -** A file virus attaches itself to an executable file, causing it to run the virus code first and then jump to the start of the original program. These viruses are termed ***parasitic,*** because they do not leave any new files on the system, and the original program is still fully functional.
- **Boot -** A boot virus occupies the boot sector, and runs before the OS is loaded. These are also known as ***memory viruses***, because in operation they reside in memory, and do not appear in the file system.
- **Macro -** These viruses exist as a macro (script) that are run automatically by certain macro-capable programs such as MS Word or Excel. These viruses can exist in word processing documents or spreadsheet files.
- **Source code** viruses look for source code and infect it in order to spread.
- **Polymorphic** viruses change every time they spread - Not their underlying functionality, but just their ***signature,*** by which virus checkers recognize them.
- **Encrypted** viruses travel in encrypted form to escape detection. In practice they are self-decrypting, which then allows them to infect other files.
- **Stealth** viruses try to avoid detection by modifying parts of the system that could be used to detect it. For example the read( ) system call could be modified so that if an infected file is read the infected part gets skipped and the reader would see the original unadulterated file.
- **Tunneling** viruses attempt to avoid detection by inserting themselves into the interrupt handler chain, or into device drivers.
- **Multipartite** viruses attack multiple parts of the system, such as files, boot sector, and memory.

9.6.2 Types of System and Network Threats –

Aside from the program threats, various system threats are also endangering the security of our system:

1. **Worm:**
   An infection program which spreads through networks. Unlike a virus, they target mainly LANs. A computer affected by a worm attacks the target system and writes a small program “hook” on it. This hook is further used to copy the worm to the target computer. This process repeats recursively, and soon enough all the systems of the LAN are affected. It uses the spawn mechanism to duplicate itself. The worm spawns copies of itself, using up a majority of system resources and also locking out all other processes.

   The basic functionality of a the worm can be represented as:



2. **Port Scanning**
   Port Scanning is technically not an attack, but rather a search for vulnerabilities to attack. The basic idea is to systematically attempt to connect to every known (or common or possible) network port on some remote machine, and to attempt to make contact. Once it is determined that a particular computer is listening to a particular port, then the next step is to determine what daemon is listening, and whether or not it is a version containing a known security flaw that can be exploited.

   Because port scanning is easily detected and traced, it is usually launched from ***zombie systems,*** i.e. previously hacked systems that are being used without the knowledge or permission of their rightful owner. For this reason it is important to protect "innocuous" systems and accounts as well as those that contain sensitive information or special privileges.

3. **Denial of Service:**
   Such attacks aren’t aimed for the purpose of collecting information or destroying system files. Rather, they are used for disrupting the legitimate use of a system or facility.
   These attacks are generally network based. They fall into two categories:
   – Attacks in this first category use so many system resources that no useful work can be performed.

   For example, downloading a file from a website that proceeds to use all available CPU time.

– Attacks in the second category involves disrupting the network of the facility. These attacks are a result of the abuse of some fundamental TCP/IP principles. fundamental functionality of TCP/IP.

9.7 Summery

Protection is an internal problem. Security, in contrast, must consider both the computer system and the environment people, buildings, businesses, valuable objects, and threats within which the system is used.

The security threat is a type of action that may harm the system.

There are four requirements of computer and network security given as follows: confidentiality, integrity, availability and authenticity.

The different types of security threats are interruption, accidental error, hardware or software errors, unauthorized access, malicious software, the race condition attack, fraudulent misuse.

The goals of security are data integrity, confidentiality, availability**,** non-repudiation, and authentication.

User authentication is a process performed by the operating system to identify the authorized users.

The different types of user authentication process are authentication using passwords, authentication using a physical object and authentication using biometrics.

9.8 Model Question

Q. 1 Explain authentication process?

Q. 2 Explain various types of system threats?

Q. 3 Write a note on Viruses

Q. 4 Explain types of Program threats?

Q. 5 Explain different types of security threats and goals.

Q. 6 What is user authentication? Explain different types of user authentication scheme.

9.9 List of References

https://www.tutorialspoint.com/operating_system/os_security.htm

https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/15_Security.html

http://www.uobabylon.edu.iq/download/M.S%202013-2014/Operating_System_Concepts,_8th_Edition%5BA4%5D.pdf

Unit 8 - CHAPTER 10

CASE STUDY

10.0 OBJECTIVES

At the end of this unit, the student will be able to

- Describe the concept of LINUX OS
- Illustrate the working of Window OS
- Highlight the concept of Android OS and IOS

10.1 CASE STUDY1- INTRODUCTION TO LINUX OS

1. Linux History

1.Linux is a modern, free operating system based on UNIX standards.

2. First developed as a small but self-contained kernel in 1991 by Linus Torvaids, with the major design goal of UNIX.

3.Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the internet.

4.It has been designed to run efficiently and reliably on common pc hardware, but also runs on a runs variety of other platforms.

5.The core Linux OS kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible OS free from proprietary code.

6. Linux System has many, varying Linux distributions including the kernel, applications, and management tools.

2. Linux Kernel

1. Version 0.01(May 1991) had no networking, ran only on 80386-compatible intel processors and on PC hardware, had extremely limited device drive support and supported only the Minix file system.

2.Linux 1.0 released in march 1994 included new features like

1. Support for Unix's standard TCP/IP networking protocols.

2.BSD compatible socket interface for networking programming

3.Device-Driver support for running IP over an Ethernet

4.It consist of enhanced file system without the limitation of the original Minixfile system.

5. It supported a range of SCSI controllers for high-performance disk access.

3. The developers extended the virtual memory subsystem o support paging to swap files and memory mapping of arbitrary files implemented in 1.0

4. A range of extra hardware support was included in this release, had grown to include floppy-disk and CD-ROM devices as well as sound cards, a range of mice and international keyboard.

5.System V unix-style interprocess communication (IPC) including shared memory, semaphores and message queues was implemented.

6.Kernel with an minor version number such as 1.1 or .5 are development kernelsand even numbered minor-version numbers are stable production kernels.

7.In March 1995, the 1.2 kernel was released with added feature like to support for 80386 CPU's virtual mode- to allow emulation of the DOS operating system for PC computers.

8.The 1.2 kernel was the final PC-Only Linux kernel. Alpha and MIPS CPUs but full integration of these other architectures did not begin until after the 1.2 stable kernel was released.

9. The Linux 1.3 developed with a great deal of new functionality added to the kernel.

10.Later on released as version 2.0, this version support for multiple architectures, including 64 bit native alpha port and symmetric multiprocessing port.

11. The memory management code was substantially improved to provide a unified cache for file-system data independent of the caching of block devices.

12. Other major improvementsincludedtheadditionofinternalkernel threads, a mechanism exposing dependencies between loadable modules, support for the automatic loading of modules on demand, file-system quotas, and POSIX-compatible real-time process-schedulingclasses.

13.Othermajorimprovementsincludedtheadditionofinternalkernel threads, a mechanism exposing dependencies between loadable modules, support for the automatic loading of modules on demand, file-system quotas, and POSIX-compatible real-time process-schedulingclasses.

14.The process scheduler was modified in version 2.6, providing an efficient O(1) scheduling algorithm. In addition, the 2.6 kernel was preemptive, allowing a process to be preempted evenwhile running in kernel mode.

15. The major version bump from 2 to 3 occurred to commemorate the 21st anniversary of Linux , with supporting additional features like virtualization support, a new page write back facility, improvements to the memory-management system, and completely fair scheduler(CFS).

3.The Linux System

1.Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X window system and the free software foundation's GNU project.

2.The main system libraries of Linux were originated by the GNU project, but the Linux community greatly improved the libraries by addressing omissions, inefficiencies, and bugs. Other components, such as the GNU C compiler (gcc), were already of sufficiently high quality to be used directly in Linux.

3.A small number of public Internet file-transfer-protocol (FTP) archive sites act as de facto standard repositories for these components. The File System hierarchy Standard document is also maintained by the Linux community as a means of ensuring compatibility across the various system components.

4. Application User's Interface

1. Interface between the kernel and user

2. Allow user to make commands to the system

3.Divided in to text based and graphical based

5.File Management

1. Control the creation, removal of files and provide directory maintenance.

2.For a multiuser system, every user should have its own right to access files and directories

6. Process Management

1. For a multitask system, multiple programs can be executed simultaneously in the system.

2. When a program starts to execute, it becomes a process

3.The same program executing at two different times will become two different processes.

4.Kernel manages processes in terms of creating, suspending, and terminating them

5.A process is protected from other processes and can communicate with the others.

7.Memory Management

1.Memory in a computer is divided in to main memory (RAM) and a secondary storage (usually refer to hard disk)

2.Memory is small in capacity but fast in speed and hard disk is vice versa.

3.Data that are not currently used should be saved to hard disk first, while data that are urgently needed should be retrieved and stored in RAM.

4. The mechanism is referred as memory management.

8 Device Drivers

1. Interfaces between the kernel and the BIOS

2. Different device has different driver

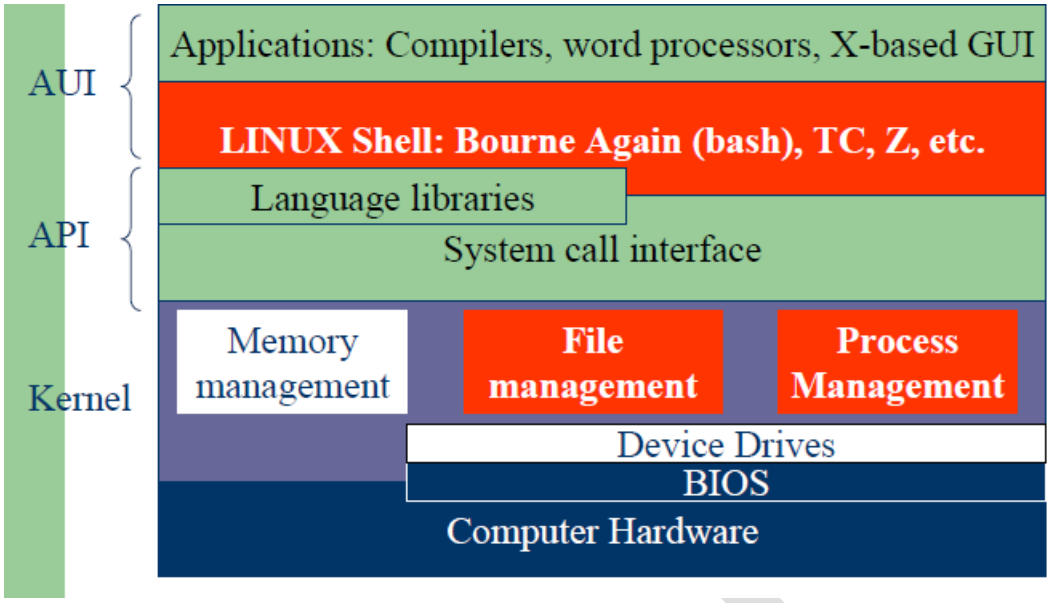


Fig 1 LINUX System Architecture

9. LINUX Shell

1. Shell interprets the command and request service from kernel.
2. Similar to DOS but DOS has only one set of the interface while Linux can select different shell like Bourne Again Shell(Bash), TC shell(Tcsh), Zshell(Zsh).
3. Different shell has similar but different functionality, whereas bash is the default shell of Linux . GUI of Linux works well on the shell.

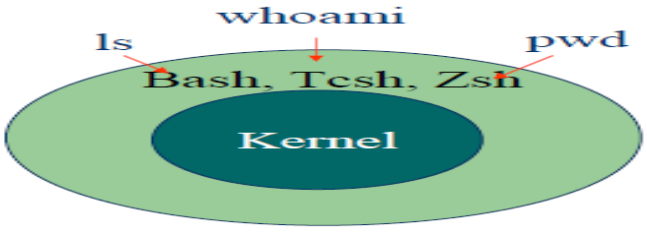


Fig 2 Shells of LINUX OS

4. Frequently used commands, available in most shells,

1. ls- to show(list) the names of the files in the current directory

For eg To used Ls command, open the copy.sh terminal in web browser, if unix OS is not installed in system, the output as

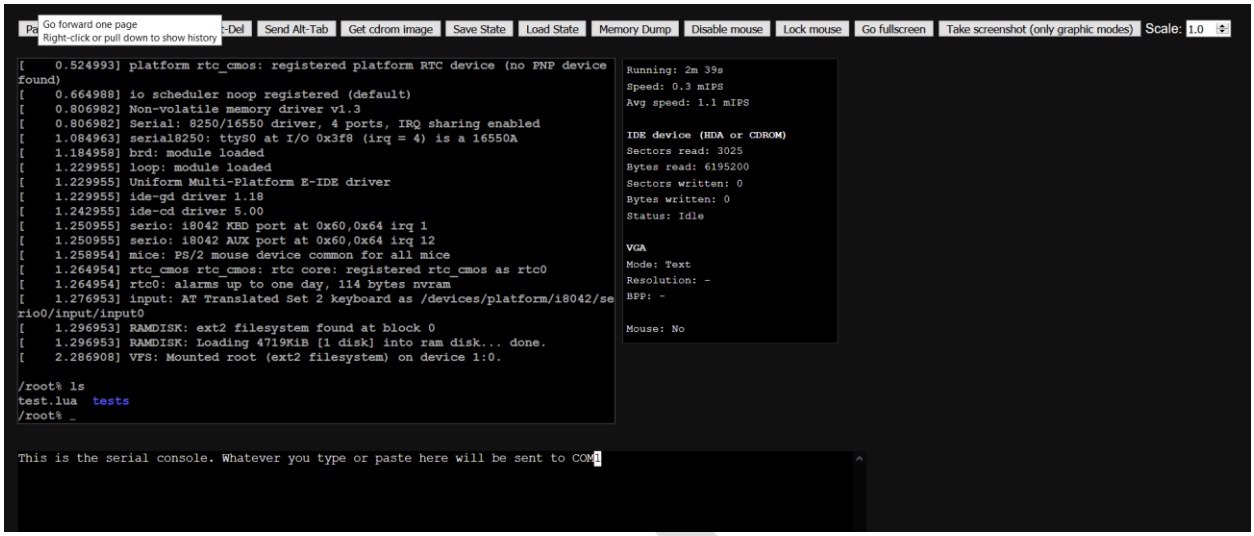


Fig 3 Ls command

2.cd: change directory, eg cd / change to the root directory

Fig 4 cd command

3.Pwd: show the name of the present working directory

Fig 5 Pwd command

4.Whoami: to show the username of the current user

Fig 6 Whoami command execution

5. cp: copy one file to another, eg cp abc.txt xyz.txt copy abc,txt to xyz.txt

6. rm:remove a file

7. man: ask for the manual(or help) of a command eg man cd ask for the manual of the command cd

10.2 CASE STUDY 2-INTRODUCTION TO WINDOW OS

1.About Windows Operating System

1.It is developed by microsoft corporation to run personal computers(Pcs)

2.It is the first GUI based systems for IBM compatible PCs

3. The window OS soon dominated the PC market. Approximately 90 percent of Pcs run some version of windows.

4. The first version of windows, released in 1985, was simply a GUI offered as an extension of microsoft's existing disk operating system or MS-DOS

5. Based in part on licensed concepts that Apple Inc, had used for its Macintosh system software, Windows for the first time allowed DOS users to visually navigate a virtual desktop, opening graphical "windows" displaying the contents of electronic folders and files with the click of a mouse button, rather than typing commands and directory paths at a text prompt.

6. Subsequent versions introduced greater functionality, including native Windows File manager, program Manager, and Print Manager programs and a more dynamic interface.

7.Microsoft also developed specialized windows packages, including the networkable windows for work groups and high powered Windows NT, aimed at businesses.

8.The 1995 consumer release Windows 95 fully integrated windows and DOS and offered built-in internet support, including the world wide web browser internet explorer.

9.With the 2001 release of windows XP, microsoft united its various Windows packages under a single banner, offering multiple editions for consumers, businesses, multimedia developers and others.

10.Windows XP abandoned the long-used windows 95 kernel for a more powerful code base and offered a more practical interface and improved application and memory management.

11.The highly successful XP standard was succeeded in late 2006 by window vista, which experienced a troubled roll out and met with considerable marketplace resistance, quickly acquiring a reputation for being a large, slow and resource consuming system.

12. Responding to vista's disappointing adoption rate, microsoft developed windows 7 and OS whose interface was similar to that of vista but was met with enthusiasm for its noticeable speed improvement and its modest system requirements.

2. History of Windows

1. Microsoft's windows operating system was first introduced in 1985.

2. Windows 1 / MS-DOS

1.The original windows 1 was released in november 1985 and was microsoft's first true attempt at a graphical user interface in 16-bit.

2. Development was spearheaded by microsoft founder bill gates and ran on top of MS-DOS which relied on command-line input.

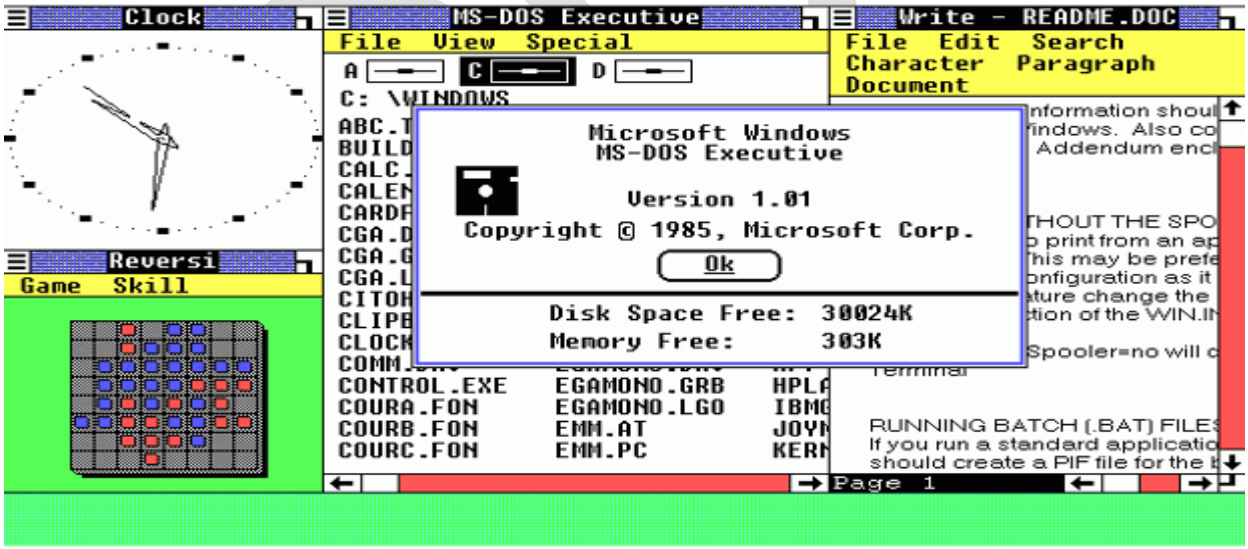


Fig 7 MS-DOS

2. Window 2

1. Two years after the release of windows 1, microsoft's windows 2 replaced it in december 1987.

2. The big innovation for windows 2 was that windows could overlap each other and also introduced the ability to minimize or maximize windows instead of ionizing or zooming.

3. The control panel where various system settings and configuration options were collected together in one place was introduced in windows 2 and survives to this day.

4. Microsoft word and excel also made their first appearances running on windows.

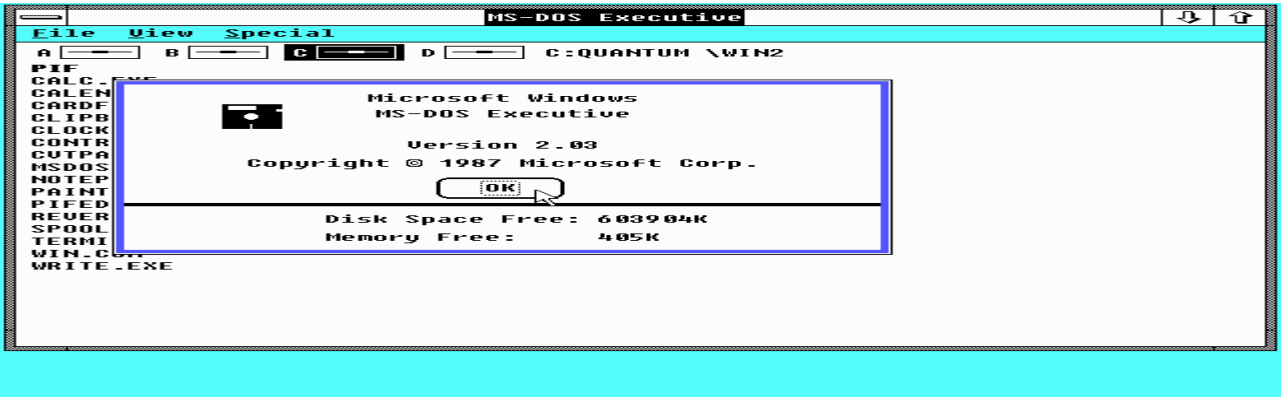


Fig 8 Window version 2.0

3. Window version 3 & 3.1

1. The first windows that required a hard drive launched in 1990. Windows 3 was the first version to see more widespread success and be considered a challenger to Apple's macintosh and the commodore Amiga GUI coming preinstalled on computers from PC compatible manufacturers zenith data systems.

2. Windows 3 introduced the ability to run MS-DOS programs in windows, which brought multitasking to legacy programs and supported 256 colors bringing a more modern, colorful look to the interface.

3. Windows 3.1 released in 1992 is notable because it introduced true type fonts making a viable publishing platform for the first time

4. Minesweeper also made its first appearance.

5. Windows 3.1 required 1MB of RAM to run and allowed supported MS-DOS programs to be controlled with a mouse for the first time.

6. Windows 3.1 was also the first windows to be distributed on CD-ROM, although once installed on a hard drive it only took up 10 to 15 MB (a CD can typically store up to 700 MB)

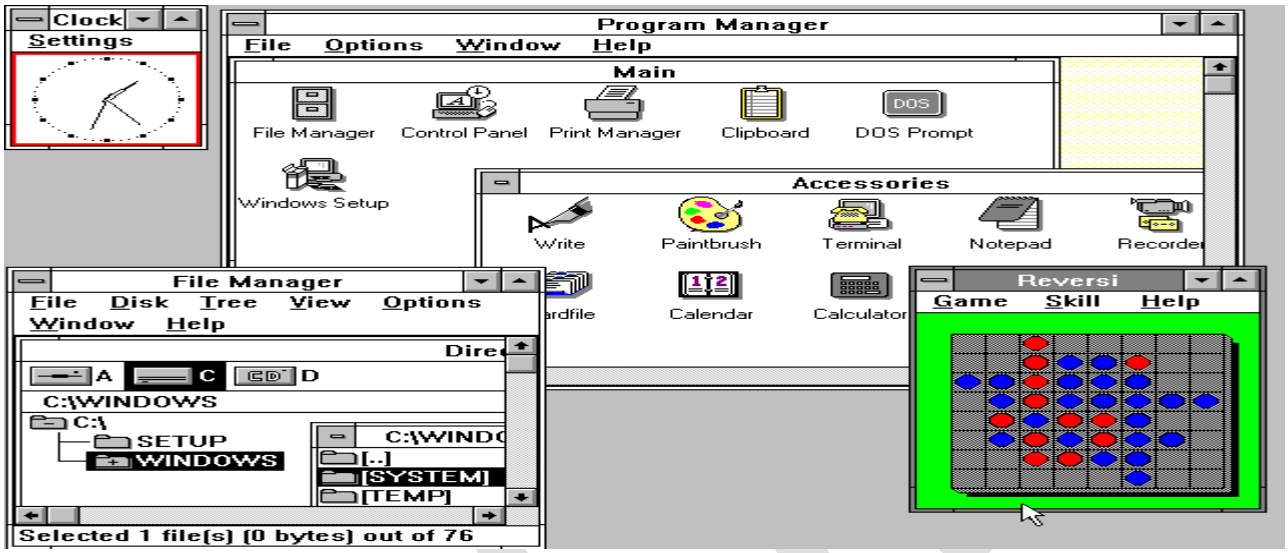


Fig 9 Windows 3

4. Windows 95

    1. As the name implies, windows 95 arrived in August 1995 and with it brought the first ever start button and start menu

5. Windows 98

1. Released in june 1998, windows 98 built on windows 95 and brought with IE 4 outlook express, windows address book, microsoft chat and net show player which was replaced by windows media player 6.2 in windows 98 second edition in 1999.

6. Windows ME

    1. Considered a low point in the windows series by many at least until they say windows vista- windows millennium edition was the last windows to be based on MS-DOS and the last in the windows 9x line. Released in september 2000, it was the consumer-aimed OS twined with windows 2000 aimed at the enterprise market. It introduced some important concepts to consumers, including more automated system recovery tools.

7. Windows 2000

1. The enterprise twin of ME windows 2000 was released in february 2000 and was based on microsoft's business-oriented system windows NT and later became the basis for Windows XP.

8. Windows XP

1. Arguably one of the best windows versions, Windows XP was released in october 2001 and brought microsoft's enterprise line and consumer line of OS under one roof.

2. It was based on Windows NT like windows 2000, but brought the consumer friendly elements from Windows ME. The start menu and task bar got a visual overhaul bringing the familiar green start button, blue task bar and vista wallpaper along with various shadow and other visual effects.

9. Windows 7

1. Windows 7 was first released in October 2009.

2. It was interested to fix all the problem and criticism faced by vista, with slight tweaks to its appearance and a concentration on user-friendly features and less dialogue box overload.

10. Windows 8

1. Released in october 2012, windows 8 was microsoft's most radical overhaul of the windows interface, ditching the start button and start menu in favour of a more touch-friendly start screen.

2. The new tiled interface saw program icons and live tiles which displayed at one glance information associated with "widgets" replace the lists of programs and icons.

11. Windows 10

1. Announced on 30 september 2014, windows 10 has only been released as a test version for keen users to try.

2. The "technical preview" is very much still a work in progress.

3. Windows 10 represents another step in Microsoft's U-turn bringing back the start menu and more balance to traditional desktop computer users.

10.3 CASE STUDY 3- INTRODUCTION TO ANDROID OS

1. Android is a mobile operating system based on a modified version of the linux kernel and other open source software, designed primarily for touch screen mobile devices such as smart phones and tablets.

2. The Android OS was originally created by Android Inc, which was bought by Google in 2005. Google teamed up with other companies to form the open handset alliance (OHA), which has become responsible for the continued development of the Android OS.

3.Inside the Linux kernel are found drivers for the display, camera, flash memory, keypad, WIFI and audio.

4. The Linux kernel serves as an abstraction between the hardware and the rest of the software on the phone. It also takes care of core system services like security, memory management, process management and network stack.

5. Android Inc was founded in Palo Alto, California, United states by Andy Rubin, Rich Miner, Nick Sears and Chris White in October 2003.

6.Google acquired Android Inc in August 2005

7.The open Handset Alliance, consortium of several companies was formed on 5th Nov 2007.

8.Android beta SDK released on 12th NOV 2007.

9. Features of Android OS is as follows:

1. Integrated browser

2. Based on the open source WebKit engine

3.Optimized 2D and 3D graphics

4.Multimedia and GSM connectivity

5. Has Bluetooth

6.SQLLITE, GPS etc

10 Version of Android

1. Android 1.5 Android Cup Cake
2. Android 1.6 Donut
3. 2.0-Eclair
4. 2.2- Froyo
5. 2.3-Gingerbread
6. 3.0-Honey Comb
7. 4.0- Ice Cream Sandwich
8. 4.1 to 4.3.1- Jelly Bean
9. 4.4 to 4.4.4-Kitkat
10. 5.0 to 5.1.1-Lollipop
11. 6.0 to 6.0.1-Marshmallow
12. 7.0 to 7.1- Nougat
13. 8.0 to 8.1-Oreo
14. 9.0 Pie etc

11.Goals of Android

1. A number of key design goals for the Android platform evolved during its development

1.1 Provide a complete open-source platform for mobile devices. The open-source part of Android is a bottom-to-top operating system stack, including a variety of applications, that can ship as a complete product.

1.2 Strongly support proprietary third-party applications with a robust and stable API. As previously discussed, it is challenging to maintain a platform that is both open-source and also stable enough for third-party applications.

1.3 Allow all third-party applications, including those from Google, to compete on a level playing field. The Android open-source code is designed to be neutral as much as possible to the higher level system features built on top of it, from access to cloud services (such as data

sync or cloud-to-device messaging API's), to libraries (such as Google's mapping library) and rich services like application stores.

1.4 Provide an application security model in which users do not have to deploy trust third-party applications. The OS must protect the user from misbehavior of applications, not only buggy applications that can cause it.

12. Android Architecture

1.Below figure explains about the different layers of Android architecture which works in assembly to carry out every function properly.

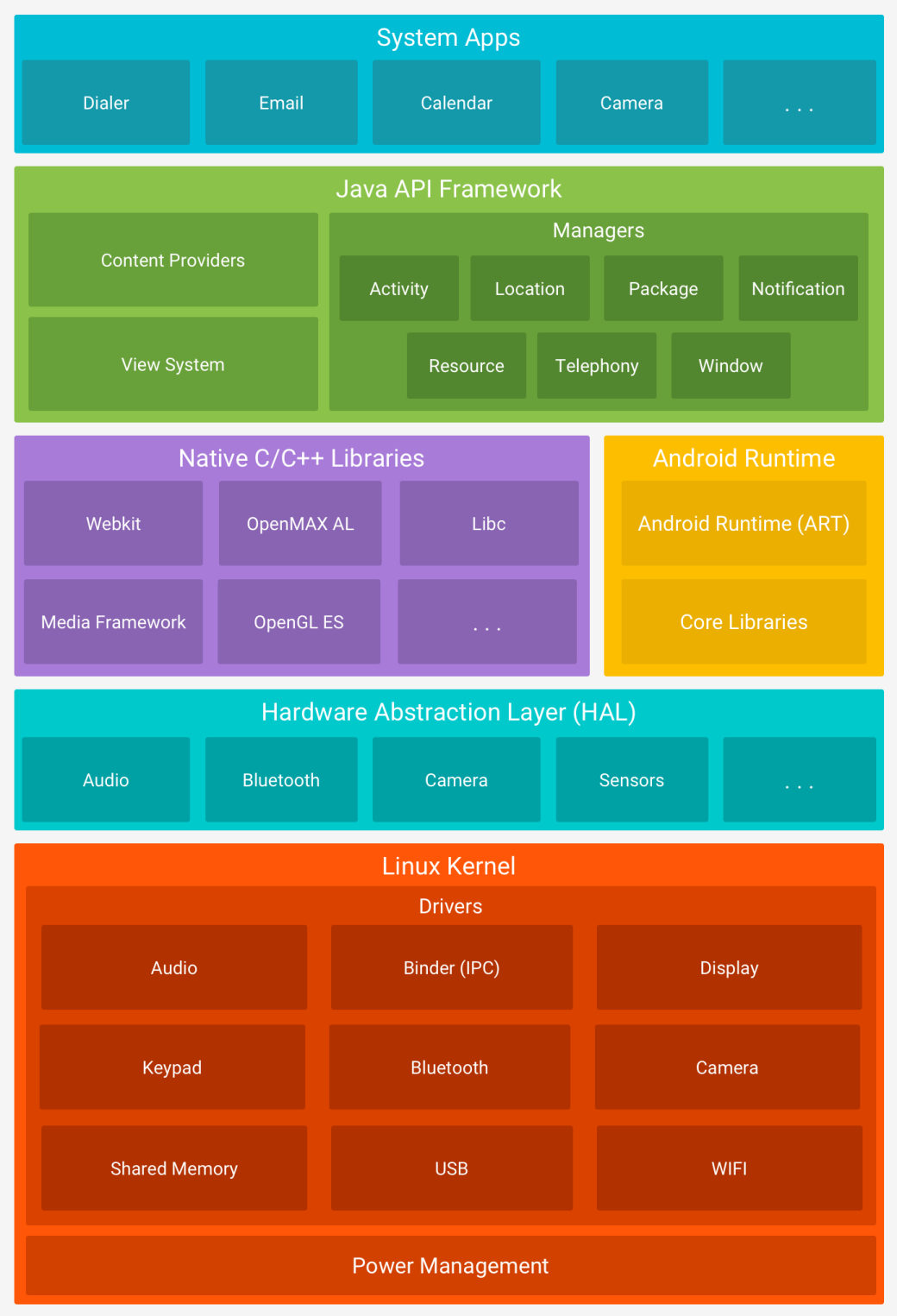


Fig 10 Android Architecture

10.4 CASE STUDY 4- INTRODUCTION TO IOS

1. IOS History

1. IOS is a mobile OS that is developed and distributed to run ipad,iphone and ipod touch devices.

2. It was first introduced in 2007 at Mac world conference and expo, January 2007.

3. Software development Kit was released in next year march 2008. The IOS catalyzed a transition of traditional mobile industry in to value network industry(Kenney and Pon 2011).

4. IOS created an ecosystem, which was originally under operations for ipod products where application developers and customers voluntarily transact with each other and bring in more value to platform.

5. Even after the market share was surpassed by Android, still 54.5 % of mobile internet connection is through IOS devices, while Android is 34.6% and Jave ME is 4.3 %

2. IOS Architecture

1. IOS architecture is written in Objective-C language and comprisedof four layers, Cocoa touch, Media, Core service, and Core OS. System interfaces are provided by framework, which contains libraries and resources such as header and image.

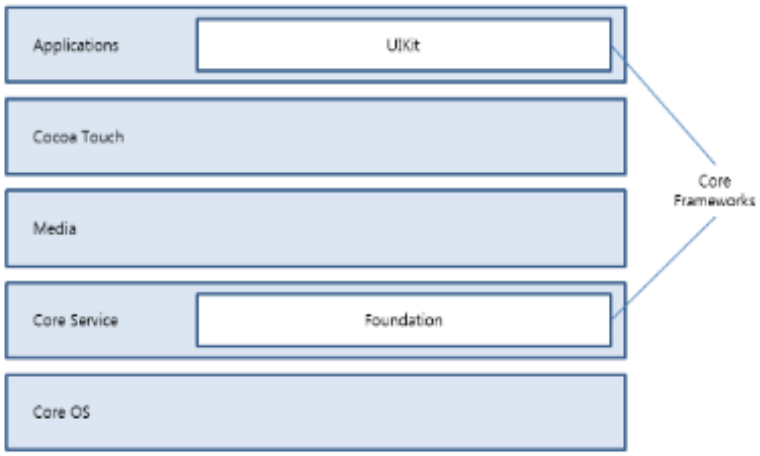


Fig 11 IOS Architecture

2.Cocoa touch layer-The Cocoa touch layer is a basic infrastructure to run applications and provide key technologies such as multitasking, touch input, and push notifications. Main frameworksof Cocoa touch layer contain Address Book UI, Event Kit UI (Calendar), Game Kit (Game center), iAd, Map Kit, Message UI, and UIKit. As a main framework, UIKit provides main user interactions such as event handling, windows and controls for touch UI.

3. Media Layer- The media layer contains technologies to implement multimedia functions, such as graphics, Audio, Video, and Airplay. It contains frameworks for audio/video codecs and OpenGL features.

4. Core Services-The core services layer provides fundamental system services for applications. Core services layer contains frameworks such as location, file sharing, in app purchase,account management, telephony.Core foundation framework and foundation framework, which provide basic data management and service features, are contained in core services layer.

5. Core OS Layer-The core OS layersits directly on top of the device hardware andprovides low level features mostly used by upper layer frameworks. To connect the iOS device with an external accessory or to deal with security features, core OS layer framework need to be used. The core OS layer contains TCP/IP networking protocol, Bluetooth framework, 64bit support mode, and kernel.

3.IOS application development

1.To develop applications for iOS,developers need integrated development environment (IDE) called Xcode, which runs on Mac OS X platform, and software development kit (SKD), that provides APIs in the form of libraries. IOS applications are written in Objective-C as well. IOS SDK was first released in February 2008.

2.To publish and upload applications on the devices, developers need to pay $99.00 per year.

Miscellaneous Questions

Q1. Describe in brief about the Linux Operating System

Q2. Describe the process management in Windows Operating System

Q3. Justify with your answer which Android version is better?

Q4. Give a Glimpse of GUI of IOS?