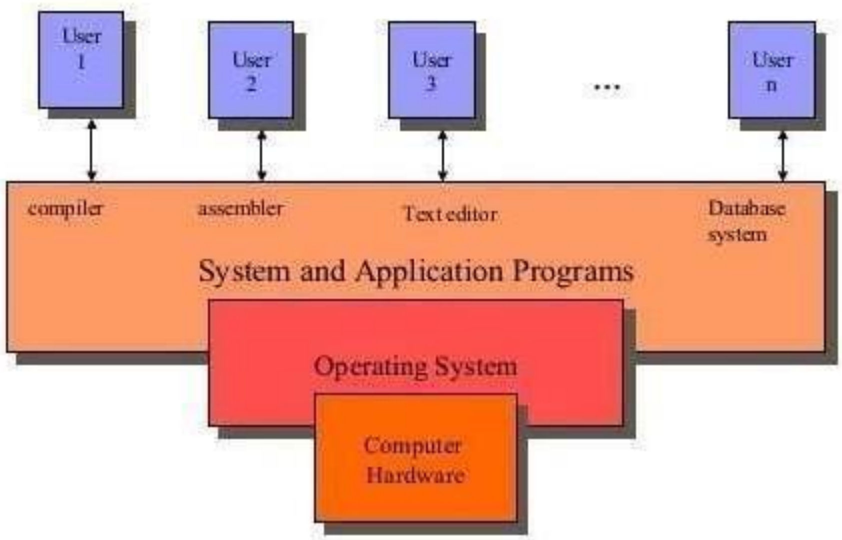
UNIT–1

Operating System Introduction-Structure-Simple batch, Multiprogramming, Time-shared, Personal Computer, Parallel, Distributed System, Real time System, System Components, OS services, System calls.

Process-Process concepts and scheduling, Operations on process, Cooperating process, Threads

Introduction of Operating System

- An OS act as an interface between user and system hardware.
- Computer consists of the hardware, Operating System, system programs, application programs.
- The hardware consists of memory, CPU, ALU, I/O device, storage device and peripheral device.
- System program consists of compilers, loaders, editors, OS etc.
- Application program consists of database programs, business programs.
- Every computer must have an OS to run other programs.
- The OS controls & coordinates the use of the hard ware among the various system programs and application programs for various tasks.
- Its imply provides an environment with in which other programs can do useful work.



your roots to success...

OPERATINGSYSTEM

Definition

- In the 1960's one might have defined OS as **"The software that controls the hardware"**.
- Operating System performs all the basic tasks like managing files,processes,

And memory. Thus operating system acts as the manager of all the resources, i.e. **Resource manager**.

- Operating system becomes an interface between the user and the machine. It is one of the most required software that is present in the device.
- Operating System is a type of software that works as an interface between the system program and the hardware.

Concept of OS

- The OS is a set of special programs that run on a computer system that allow it to work properly.
- It performs basic task as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling a peripheral device.
- The OS must support the following tasks. They are,
  - Provides the facilities to create, modification of program and data file 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 memory for execution.

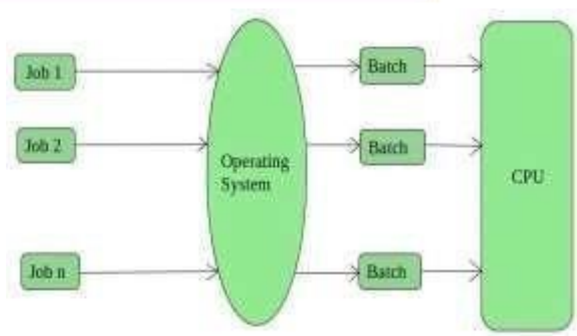
Types of Operating Systems

There are several types of Operating Systems which are mentioned below.

- Batch Operating System
- Multi-Programming System
- Time-Sharing Operating System
- Personal Computers
- Parallel Operating System
- Distributed Operating System
- Real-Time Operating System

1. Batch Operating System

This type of operating system does not interact with the computer directly. There is an operator which takes similar job shaving the same requirement and groups them into batches. It is the responsibility of the operator to sort jobs with similar needs.



Advantages

- Processors of the batch systems know how long the job would be when itis in the queue.
- Multiple users can share the batch systems.
- The idle time for the batch system is very less.
- It is easy to manage large work repeatedly in batch systems.

Disadvantages

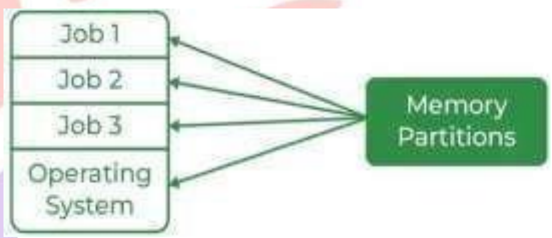
- The computer operators should be well known with batch systems.
- Batch systems are hard to debug.
- It is sometimes costly.
- The other jobs will have to wait for an unknown time if any job fails.
- It is very difficult to guess or know the time required for any job to complete.

Examples

Payroll Systems, Bank Statements, etc.

2. Multi-Programming Operating System

Multi programming Operating Systems can be simply illustrated as more than one program is present in the main memory and any one of them can be kept in execution. This is basically used for better Execution of resources.



Advantages of Multi-Programming Operating System

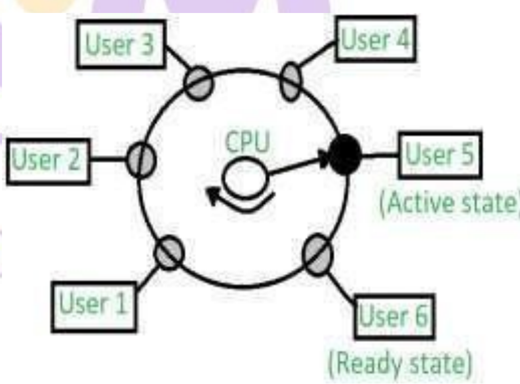
- Multi Programming increases the Through put of the System.
- It helps in reducing there sponse time.

Disadvantages of Multi-Programming Operating System

- There is not any facility for user interaction of system resources with the system.

3. Time-Sharing Operating Systems

Each task is given some time to execute so that all the tasks work smoothly. Each user gets the time of the CPU as they use a single system. These systems are also known as Multitasking Systems. The task can be from a single user or different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to the next task.



Advantages

- Each task gets an equal opportunity.
- Fewer chances of duplication of software.
- CPU idle time can be reduced.
- **Resource Sharing:** Time-sharing systems allow multiple users to share hardware resources such as the CPU, memory, and peripherals, reducing the cost of hardware and increasing efficiency.
- **Improved Productivity:** Time-sharing allows users to work concurrently, thereby reducing the waiting time for their turn to use the computer. This increased productivity translates to more work getting done in less time.
- **Improved User Experience:** Time-sharing provides an interactive environment that allows users to communicate with the computer in real time, providing a better user experience than batch processing.

Disadvantages

- Reliability problem.
- One must have to take care of the security and integrity of user programs and data.
- Data communication problem.
- **High Overhead:** Time-sharing systems have a higher overhead than other operating systems due to the need for scheduling, context switching, and other overheads that come with supporting multiple users.
- **Complexity:** Time-sharing systems are complex and require advanced software to manage multiple users simultaneously. This complexity increases the chance of bugs and errors.
- **Security Risks:** With multiple users sharing resources, the risk of security breaches increases. Time-sharing systems require careful management of user access, authentication, and authorization to ensure the security of data and software.
- - sharing operating system that allows multiple users to access a Windows server remotely. Users can run their own applications and access shared resources, such as printers and network storage, in real-time.

4. Personal Computer

A personal computer (PC) is a microcomputer designed for use by one person at a time.

Prior to the PC, computers were designed for -- and only affordable for – companies that attached terminals for multiple users to a single large mainframe computer whose resources were shared among all users. By the1980s, technological advances made it feasible to build a small computer that an individual could own and use as a word processor and for other computing functions.

your roots to success...

Whether they are home computers or business ones, PCs can be used to store, retrieve and process data of all kinds. A PC runs firmware that supports an operating system (OS), which supports a spectrum of other software. This software lets consumers and business users perform a range of general-purpose tasks, such as the following:

- word processing
- spreadsheets
- email
- instant messaging
- accounting
- database management
- internet access
- listening to music
- network-attached storage
- graphic design
- music composition
- video gaming
- software development
- network reconnaissance
- multimedia a servers
- wireless network access hot spots
- video conferencing

Types

Personal computers fall into various categories, such as the following:

- **Desktop computers** usually have a tower, monitor, keyboard and mouse.
- **Tablets** are mobile devices with a touch screen display.
- **Smart phones** are phones with computing capabilities.
- **Wearables** are devices users wear, such as smartwatches and various types of smart clothing.
- **Laptop computers** are portable personal computers that usually come with an attached keyboard and trackpad.
- **Note book computers** are light weight laptops.
- **Handheld computers** include advanced calculators and various gaming devices.

5. Parallel Operating System

Parallel Systems are designed to speed up the execution of programs by dividing the programs into multiple fragments and processing these fragments at the same time.

your roots to success...

Advantages

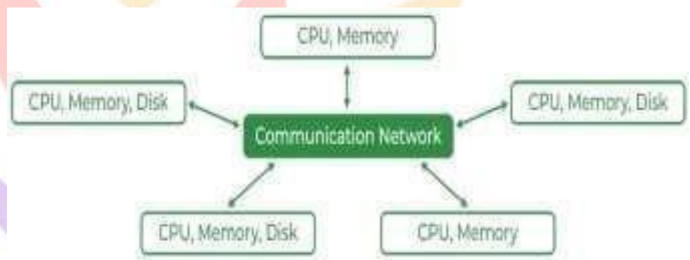
- **High Performance:** Parallel systems can execute computationally intensive tasks more quickly compared to single processor systems.
- **Cost Effective:** Parallel systems can be more cost-effective compared to distributed systems, as they do not require additional hardware for communication.

Disadvantages

- **Limited Scalability:** Parallel systems have limited scalability as the number of processors or cores in a single computer is finite.
- **Complexity:** Parallel systems are more complex to program and debug compared to single processor systems.
- **Synchronization Overhead:** Synchronization between processors in a parallel system can add overhead and impact performance.

6. Distributed Operating System

These types of operating systems are a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, at agreat pace.



Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as loosely coupled systems or distributed systems. These systems' processors differ in size and function.

The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network i.e., remote access is enabled within the devices connected in that network.

Types of Distributed Systems

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows −

Client/Server Systems

In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

Peer to Peer Systems

The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

Advantages

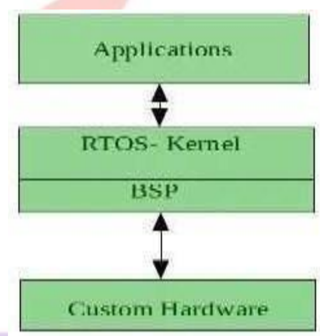
- Failure of one will not affect the other network communication, as all systems are independent of each other.
- Electronic mail increases the data exchangespeed.
- Sinceresourcesarebeingshared,computationishighlyfastanddurable.
- Loadonhostcomputerreduces.
- Thesesystemsareeasilyscalableasmany systemscanbeeasily addedtothe network.
- Delay indataprocessingreduces.

Disadvantages

- Failureofthemainnetworkwillstoptheentirecommunication.
- Toestablishdistributedsystemsthelanguageisusednotwell-definedyet.
- These types of systems are not readily available astheyarevery expensive.Notonly thattheunderlyingsoftwareishighly complexandnot understood well yet.

Example:LOCUS

7. Real-TimeOperatingSystem



These types of OSs serve real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

Real-time systems are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

Types:

1. **HardReal-TimeSystems**

   Hard Real-Time OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable.These systemsarebuilt for saving life like automatic parachutes or airbags whicharerequired to be readily available in case of an accident. Virtual memory is rarely found in these systems.

2. **SoftReal-TimeSystems**

   TheseOSsareforapplicationswheretime-constraintislessstrict.

Advantages

- **MaximumConsumption:** Maximumutilizationofdevicesandsystems,thus more output from all the resources.
- **TaskShifting:**Thetimeassignedforshiftingtasksinthesesystemsis very less. For example, in older systems, it takes about 10 microsecondsin shifting from one task to another, and in the latest systems, it takes 3 microseconds.

- **Focus on Application:** Focus on running applications and less importance on applications that are in the queue.
- Real-time **operating system in** the **embedded system:** Since the size of programs is small, RTOS can also be used in embedded systems like intransport and others.
- **ErrorFree:**Thesetypesofsystemsareerror-free.
- **Memory Allocation:** Memory allocation is best managed in these typesof systems.

Disadvantages

- **Limited Tasks:** Very few tasks run at the same time and theirconcentration is very less on a few applications to avoid errors.
- **Useheavysystemresources:**Sometimesthesystemresourcesarenotsogood and they are expensive as well.
- **ComplexAlgorithms:** Thealgorithmsareverycomplexanddifficultforthe designer to write on.
- **Devicedriverandinterruptsignals:**Itneedsspecificdevicedriversand interrupts signal to respond earliest to interrupts.
- **ThreadPriority:**Itisnotgoodtosetthreadpriorityasthesesystemsare very less prone to switching tasks.

Examples

Scientificexperiments,medicalimaging systems, industrial control systems, weapon systems,robots,airtrafficcontrolsystems,etc.

OperatingSystemServices

- **User Interface -** User interface is essential and all operating systems provide it. Users either interface with the operating system through command-lineinterface (CUI) or graphical userinterface(GUI). Command interpreter executes next user-specifiedcommand.AGUI offers the user a mouse-based window and menu system as an interface.
- **Program execution -** The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** -A running program may require I/O, which may involve afile or an I/O device.

- **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create anddelete them, search them, list file Information, permission management.
- **Communications**–Processesmayexchangeinformation,onthesame

NRCM
your roots to success...

computer or between computers over a network. Communications may be via shared memory or through message passing (packets moved by the OS)

- **Error detection** – OS needs to be constantly aware of possible errors mayoccur in the CPU and memory hardware, in I/O devices, in user program. For each type of error, OS should take the appropriate action to ensure correct and consistent computing. Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

AnothersetofOSfunctionsexistsforensuringtheefficientoperationofthe system itself via resource sharing

- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them. Many types of resources such as CPU cycles, main memory, and file storage may have special allocation code, others such as I/O devices may havegeneral request andrelease code.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other. **Protection** involves ensuring that all access to system resources is controlled. **Security**of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts. If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

SystemCalls

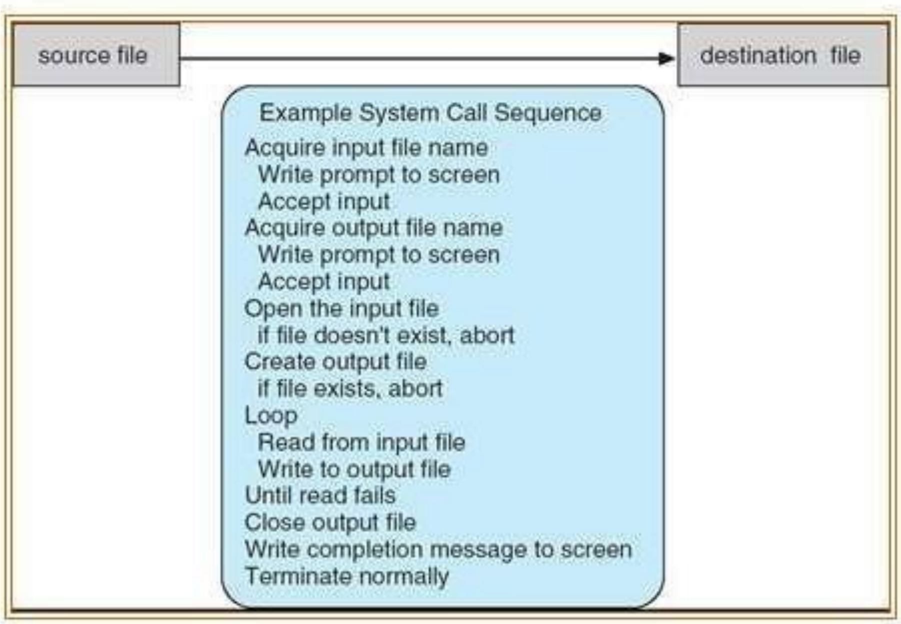
- Asystemcallis awayforauserprogramtointerfacewiththeoperating system. The program requests several services, and the OS responds by invoking aseries of system calls to satisfy the request.
- Asystemcallcanbewritteninassemblylanguageorahigh-levellanguagelike **C,C++**or**Pascal**.
- Systemcallsarepredefinedfunctionsthattheoperatingsystemmaydirectly invoke if a high-level language is used.
- A system call is a method for a computer program to request a service fromthe kernel of the operating system on which it is running.
- Asystemcallis amethodof interactingwiththeoperatingsystemviaprograms.
- Asystemcallisarequestfromcomputersoftwaretoanoperatingsystem's kernel.

- A simple system call may take few nanoseconds to provide the result, like retrieving the system date and time. A more complicated system call, such as connecting to a network device, may take a few seconds. Most operating systems launch a distinct kernel thread for each system call to avoidbottlenecks. Modern operating systems are multi-threaded, which means they can handle various system calls at the same time.
- The **Application Program Interface (API)** connects theoperating system's functions to user programs. It acts as a link between the operating system and a process, allowing user-level programs to request operating system services. The kernel system can only be accessedusing system calls. System calls arerequired for any programs that use resources.

- Whencomputersoftwareneedstoaccesstheoperatingsystem'skernel,it makes a system call. The system call uses an API to expose the operating system's services touserprograms. It is theonlymethod to access the kernel system. All programs or processes that require resources for execution must use system calls, as they serve as an interface between the operating system and user programs.

ExampleofSystemCalls

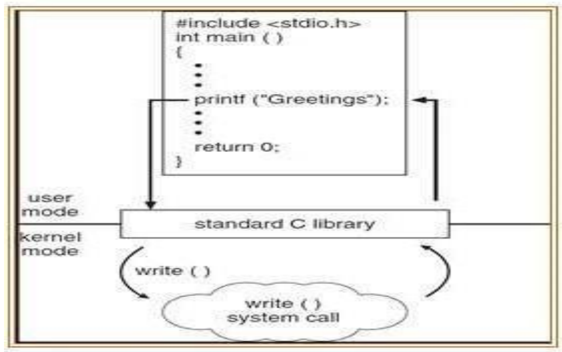
Systemcallsequencetocopythecontentsofone filetoanotherfile



your roots to success...

StandardCLibraryExample

Cprograminvokingprintf()librarycall,whichcall write()systemcall



Therearevarioussituationswherewemustrequiresystemcallsintheoperating system. Following of the situations are as follows:

1. Itismustrequirewhenafilesystem wantstocreateordeleteafile.
2. Networkconnectionsrequirethesystemcallstosendingandreceivingdata packets.
3. Ifyouwanttoreadorwriteafile,youneedtosystemcalls.
4. Ifyouwanttoaccesshardwaredevices,includingaprinter,scanner,youneed a system call.
5. Systemcallsareusedtocreateandmanagenew processes.

TypesofSystemCalls

Therearecommonlyfive typesof systemcalls.Theseare asfollows:

1. **ProcessControl**
2. **FileManagement**
3. **DeviceManagement**
4. **InformationMaintenance**
5. **Communicatio**

nProcess Control

Process control is the system call that is used to direct the processes. Some process control examples include creating, load, abort, end, execute, process,terminate the process, etc.

FileManagement

Filemanagementisasystemcallthatisusedtohandlethefiles.Somefile management examples include creating files, delete files, open, close, read, write, etc.

DeviceManagement

Devicemanagementisasystemcallthatisusedtodealwithdevices.Some examplesofdevicemanagementincluderead,device,write,getdeviceattributes, releasedevice,etc.

InformationMaintenance

Information maintenance is a system call that is used to maintain information. Therearesomeexamplesofinformationmaintenance,includinggettingsystemdata,set time or date, get time or date, set system data, etc.

Communication

Communicationisasystemcallthatisusedforcommunication.Thereare some examples of communication, including create, delete communication connections,send, receive messages, etc.

ExamplesofWindowsandUnixsystemcalls

Process	Windows	Unix
ProcessControl	CreateProcess() ExitProcess() WaitForSingleObject()	Fork() Exit() Wait()
FileManipulation	CreateFile()ReadFile() WriteFile() CloseHandle()	Open() Read() Write() Close()
DeviceManagement	SetConsoleMode() ReadConsole()WriteConsole()	Ioctl() Read() Write()
InformationMaintenance	GetCurrentProcessID() SetTimer() Sleep()	Getpid() Alarm() Sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	Pipe() Shmget() Mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorgroup()	Chmod() Umask() Chown()

open()

The**open()**systemcallallowsyoutoaccessafileonafilesystem.It allocatesresources to the fileand provides ahandlethat the process mayreferto. Many processescanopenafileatonceorbyasingleprocessonly.It'sallbasedonthefile systemand structure.

read()

It is used to obtain data from a file on the file system. It accepts threearguments in general:

- Afiledescriptor.
- Abuffertostoreread data.
- Thenumberofbytes to readfromthefile.

Thefiledescriptorofthefiletobereadcouldbeusedtoidentifyitandopenit using **open()** before reading.

wait()

Insomesystems,aprocessmayhavetowaitforanotherprocessto completeitsexecutionbeforeproceeding.Whenaparentprocessmakesachild process, theparent process execution is suspended until the child process is finished.The**wait()**systemcallisusedtosuspendtheparentprocess.Oncethe childprocesshas completed its execution, control is returned to the parent process.

write()

Itisusedtowritedatafromauserbuffertoadevicelikeafile.Thissystemcall is one wayfor a program to generate data. It takes three arguments in general:

- Afiledescriptor.
- Apointertothebufferin whichdataissaved.
- Thenumber ofbytesto bewrittenfromthebuffer.

fork()

Processesgenerateclonesofthemselvesusingthe **fork()**systemcall.Itis one of the most common ways to create processes in operating systems. When a parent process spawns a child process, execution of the parent process is interrupted until the child process completes. Once the child processhas completedits execution, control is returned to the parent process.

close()

Itisusedtoendfile systemaccess.Whenthissystemcallisinvoked, itsignifies thattheprogramnolongerrequiresthefile,andthebuffersareflushed,thefileinformation is altered, and the file resources are de-allocated as a result.

exec()

Whenanexecutablefilereplacesanearlierexecutablefileinanalready executing process,thissystemfunctionisinvoked.Asanewprocessisnotbuilt,the oldprocessidentificationstays,butthenewprocessreplacesdata,stack,data,head,etc.

exit()

The**exit()**isasystemcallthatisusedtoendprogramexecution.Thiscall indicatesthatthethreadexecutioniscomplete,whichisespeciallyusefulinmultithreaded environments. The operating system reclaims resources spent by the processfollowing the use of the **exit()** system function.

SystemcomponentsinOS:-

An operating system is a large and complex system that can only be created by partitioning intosmall parts. These pieces should be a well-defined part of the system, carefully defining inputs,outputs,andfunctions.

Although Windows, Mac, UNIX, Linux, and other OS do not have the same structure, mostoperating systems share similar OS system components, such as file,memory, process, I/Odevicemanagement.

Thecomponentsofanoperatingsystemplayakeyroletomakeavarietyofcomputer systempartsworktogether.There arethefollowingcomponentsofanoperatingsystem,suchas:

1. ProcessManagement
2. FileManagement
3. NetworkManagement
4. MainMemoryManagement
5. SecondaryStorageManagement
6. I/ODeviceManagement
7. SecurityManagement
8. CommandInterpreterSystem

Operatingsystem componentshelp you get thecorrect computingbydetectingCPU and memoryhardware errors.

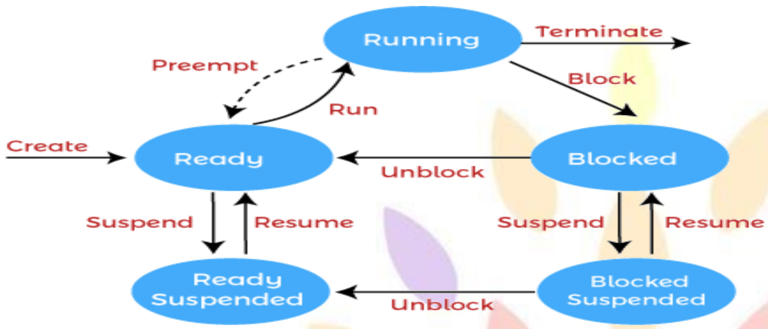
1. ProcessManagement

The process management component is a procedure for managing many processes running simultaneously on the operating system. Every running softwareapplication program has one or more processes associated with them.

For example, when you use a search engine like Chrome, there is a process running for thatbrowserprogram.

Process management keeps processes running efficiently. It also uses memory allocated to themandshuttingthemdownwhenneeded.

The execution of a process must be sequential so, at least one instruction should be executed onbehalfoftheprocess.



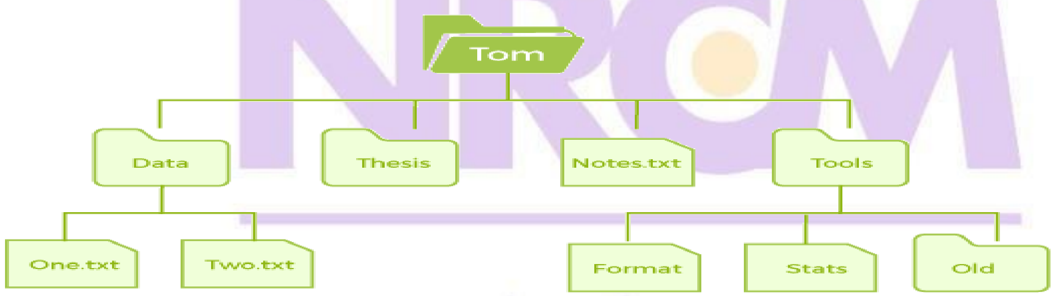
Functionsofprocessmanagement

Herearethefollowingfunctionsofprocessmanagementintheoperatingsystem,suchas:

- Processcreationanddeletion.
- Suspensionandresumption.
- Synchronizationprocess
- Communicationprocess

2. FileManagement

A file is a set of related information defined by its creator. It commonly represents programs(bothsourceandobjectforms)anddata. Datafiles canbealphabetic, numeric, oralphanumeric.



Functionoffilemanagement

Theoperatingsystemhasthefollowingimportantactivitiesinconnectionwithfilemanagement:

- Fileanddirectorycreationanddeletion.
- Formanipulatingfilesanddirectories.
- Mappingfilesontosecondarystorage.

- Backupfilesonstablestoragemedia.

3. NetworkManagement

Networkmanagementis the process of administeringandmanagingcomputernetworks. Itincludesperformancemanagement, provisioning of networks, fault analysis, and maintaining thequalityofservice.

A distributed system is a collection of computers or processors thatnever share theirmemoryand clock. In this type of system, all the processors have their local memory, and the processorscommunicate with each other using differentcommunication cables, such as fibre optics ortelephonelines.

Thecomputersinthenetworkareconnectedthroughacommunicationnetwork, which canconfigureinmanydifferentways.Thenetworkcanfullyorpartiallyconnectinnetwork management,whichhelpsusersdesignroutingandconnectionstrategiesthatovercomeco nnectionandsecurityissues.

FunctionsofNetworkmanagement

Networkmanagementprovidesthefollowingfunctions,suchas:

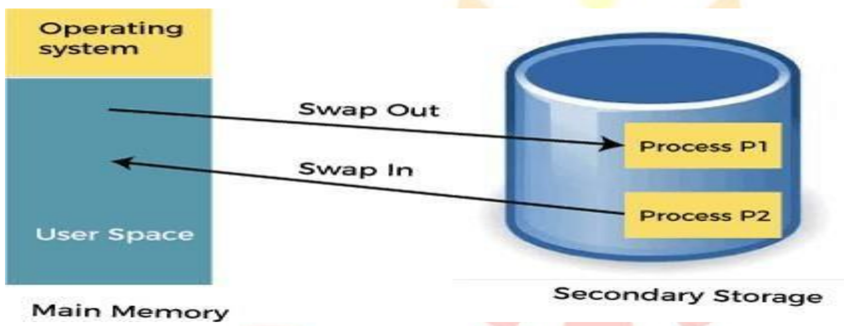
- Distributed systems help you to various computing resources in size and function. Theymayinvolveminicomputers,microprocessors,andmanygeneral-purposecomputersystems.
- A distributed system also offers the user access to the various resources the networkshares.
- It helps to access shared resources that helpcomputation to speed up oroffers dataavailabilityandreliability.

4. MainMemorymanagement

Mainmemoryisalargearrayofstorageorbytes,whichhasanaddress.Thememorymanage mentprocessisconductedbyusingasequenceofreadsorwritesofspecificmemoryaddresses.

It should be mapped to absolute addresses and loaded inside the memoryto execute a program.The selectionofamemorymanagementmethoddependsonseveralfactors.

However, it is mainly based on the hardware design of the system. Each algorithm requirescorresponding hardware support. Main memory offers fast storage that can be accessed directlyby the CPU. It is costly and hence has a lower storage capacity. However, for a program to beexecuted,itmustbeinthemainmemory.



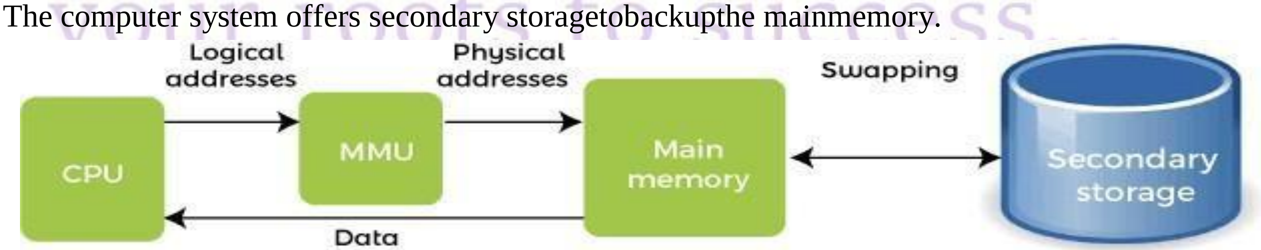
FunctionsofMemorymanagement

AnOperatingSystemperformsthefollowingfunctionsforMemoryManagementintheop eratingsystem:

- Ithelpsyoutokeeptrackofprimarymemory.
- Determinewhatpartofitareinusebywhom,whatpartisnotinuse.
- Inamultiprogrammingsystem,theOSdecideswhichprocesswillgetmemoryand howmuch.
- Allocatesthememorywhenaprocessrequests.
- Italsode-allocatesthememorywhenaprocessnolongerrequiresorhasbeenterminated.

5. Secondary-StorageManagement

The most important task of a computer system is to execute programs. These programs help youto access the data from the main memory during execution. This memory of the computer is verysmall to store all data and programs permanently. The computer system offers secondary storagetobackupthe mainmemory.



Todaymoderncomputersuseharddrives/SSDastheprimarystorageofbothprogramsand data.However,thesecondarystoragemanagementalsoworkswith storage devices, such as USB flashdrivesandCD/DVDdrives.Programslikeassemblersandcompilersarestoredon thediskuntilitisloadedintomemory,andthenusethediskisusedasasourceanddestinationf orprocessing.

FunctionsofSecondarystoragemanagement

Herearesomemajorfunctionsofsecondarystoragemanagementintheoperatingsystem:

- Storageallocation
- Freespacemanagement
- Diskscheduling
-

6. I/ODeviceManagement

One of the important use of an operating system that helps to hide the variations of specifichardware devicesfromtheuser.

FunctionsofI/Omanagement

TheI/Omanagementsystemoffersthefollowingfunctions,suchas:

- Itoffersabuffercachingsystem
- Itprovidesgeneraldevicedrivercode
- Itprovidesdriversforparticularhardwaredevices.
- I/Ohelpsyoutoknowtheindividualitiesofaspecificdevice.

7. SecurityManagement

The various processes in an operating system need to be secured from other activities. Therefore,various mechanisms can ensure those processes that want to operate files, memory CPU, andotherhardwareresourcesshouldhaveproperauthorizationfromtheoperatingsystem.

Securityreferstoamechanismforcontrollingtheaccessofprograms,processes,or users to theresourcesdefinedbycomputercontrolstobeimposed,togetherwithsomemeansofenforcement.

For example, memory addressing hardware helps to confirm that a process can be executedwithin its own address space. The time ensures that no process has control of the CPU withoutrenouncing it. Lastly, no process is allowed to do its own I/O to protect, which helps you to keepthe integrityofthevarious peripheraldevices.

Security can improve reliability by detecting latent errors at the interfaces between componentsubsystems.Early detection of interface errors can prevent thefoulness of ahealthysubsystembya malfunctioningsubsystem.Anunprotectedresource cannot misuse by an unauthorized orincompetentuser.

9. CommandInterpreterSystem

One of the most important components of an operating system is its command interpreter.Thecommandinterpreteristheprimaryinterfacebetweentheuserandtherest ofthesystem.

Many commands are given to the operating system by control statements. A program that readsand interprets control statements is automatically executed when a new job is started in a batchsystemorauserlogsintoatime-sharedsystem.Thisprogramisvariouslycalled.

- Thecontrolcardinterpreter,
- Thecommand-lineinterpreter,
- Theshell(inUNIX),andsoon.

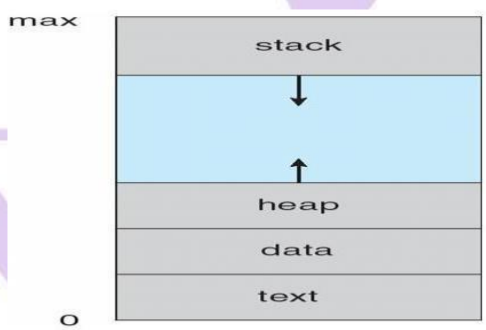
Its function is quite simple, get the next command statement, and execute it. The commandstatements deal with process management, I/O handling, secondarystorage management, mainmemorymanagement,file systemaccess,protection,andnetworking.

PROCESS: A process can be thought of as a program in execution. A process is the unit of workin most systems.

A process will need certain resources—such as CPU time, memory, files, and I/O devices to accomplish its task. These resources are allocated to the process either when it is created or while it is executing.

StructureofaProcessinMemory

- Aprocessismorethantheprogramcode,whichissometimesknownasthe**text section**.
- Italsoincludesthecurrentactivity,asrepresentedbythevalueof the**programcounter** andthecontentsoftheprocessor'sregisters.
- A process generally also includes the process **stack**, which contains temporary data (suchasfunction parameters, return addresses, and local variables).
- A**datasection**,whichcontainsglobalvariables.
- Aprocessmayalsoincludea**heap**,whichismemorythatisdynamicallyallocated during process run time.



WhenaProgrambecomes Process?

A program is a ***passive*** entity, such as a file containing a list of instructions stored on disk (Often called as **executable file**). In contrast, a process is an ***active*** entity,with a program counter specifying the nextinstruction to execute and a set of associated resources. Aprogrambecomes a process when an executable file is loaded into memory.

Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog.exe or a.out).

Iftwoprocessesareassociatedwiththesameprogram,aretheysameordifferent?(Or) Explain if you run same program twice, what section would be shared in memory?

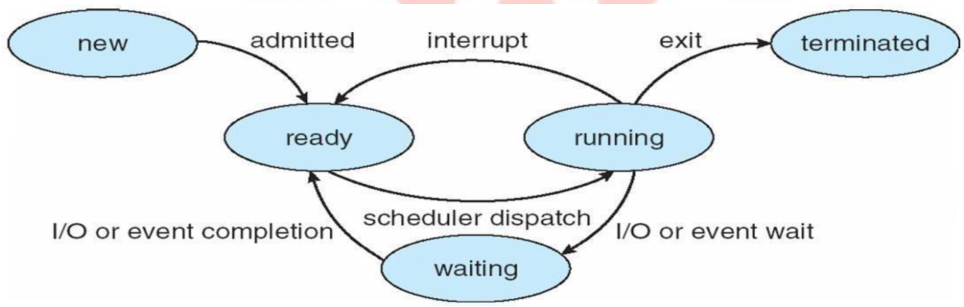
Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. For instance, several users may berunning different copies of the mail program, or the same usermay invoke many copies of the web browser program. Each of these is a separate process; and although the text sections are equivalent, the data, heap, and stack sections vary. It is also common to have a process that spawns many processes as it runs.

1. ProcessState

As a process executes, it changes **state**. The state ofa process is defined in part bythecurrent activity of that process.

Aprocessmaybeinone ofthefollowing states:

- **New:**Theprocess isbeingcreated.
- **Running:**Instructionsarebeingexecuted.
- **Waiting**:Theprocessiswaitingforsomeeventtooccur(suchasanI/Ocompletionor reception of a signal).
- **Ready:**Theprocessiswaitingtobeassigned toaprocessor.
- **Terminated**:Theprocesshasfinishedexecution.

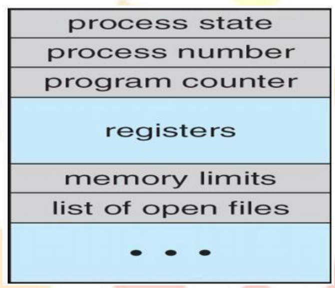


2. ProcessControlBlock

Each process is represented in the operatingsystem bya **Process Control Block (PCB)** or **Task Control Block.** It contains many pieces of information associated with a specific process, including these:

- **Processstate:**Thestatemaybe new,ready,running, andwaiting, halted,andsoon.
- **Programcounter**. The counterindicatestheaddress ofthenext instruction tobeexecutedfor 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. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
- **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

- **Memory-management information**. This information may include such items as the valueof the base and limit registers and the page tables, or the segment tables, dependingon the memory system used by the operating system.
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.



ProcessScheduling

Theobjectiveofmultiprogrammingistohavesomeprocessrunningatalltimes,to maximize CPU utilization.

TheobjectiveoftimesharingistoswitchtheCPUamongprocessessofrequentlythat users can interact with each program while it is running.

To meet these objectives, the **process scheduler** selects an available process (possiblyfrom aset of several available processes) for program execution on the CPU.

1. SchedulingQueues

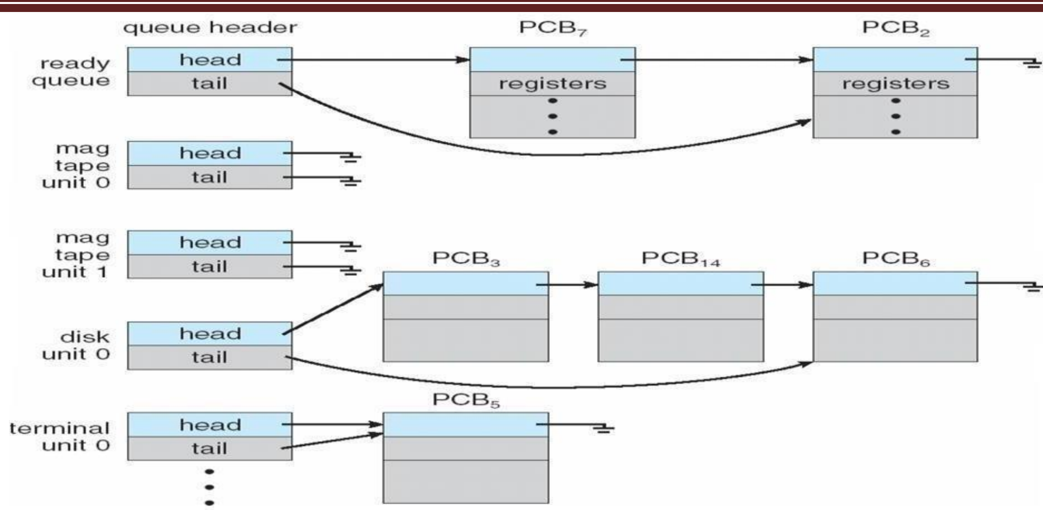
Thefollowingarethedifferentqueuesavailable,

a. Job Queue

- Asprocessesenterthesystem,theyareputintoa**jobqueue**,whichconsistsofall processes in the system.

b. Ready Queue

- The processes that are residingin main memoryand are readyand waitingto execute are kept on a list called the **ready queue**.
- This queue is generally stored as a linked list. A ready-queue header containspointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.



c. **DeviceQueue**

- ThelistofprocesseswaitingforaparticularI/Odeviceiscalled a**devicequeue**.
- Eachdevicehasitsowndevice queue.

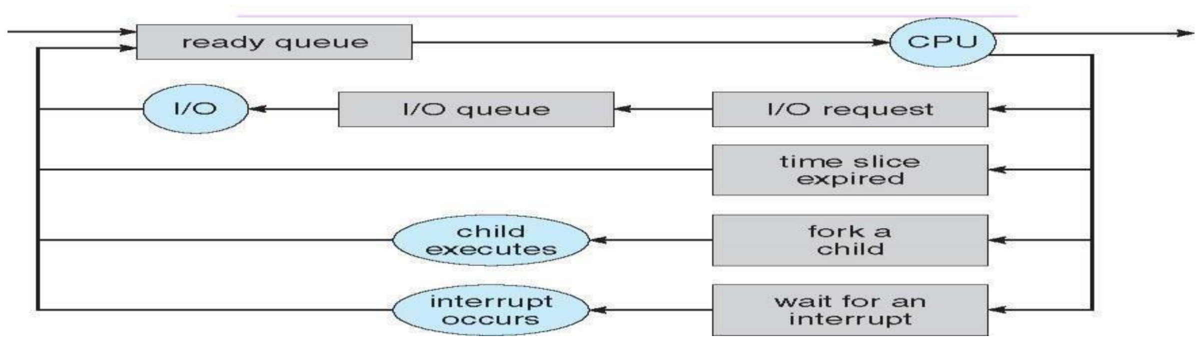
Queuing-diagramrepresentationofprocessscheduling

Acommonrepresentationofprocessschedulingisa**queuingdiagram**.Eachrectangular boxrepresentsaqueue. Twotypesofqueuesare present: theready queueanda set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

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

- TheprocesscouldissueanI/OrequestandthenbeplacedinanI/O queue.
- Theprocesscould createanewchildprocessandwaitforthechild'stermination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, andbeput back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



2. **Schedulers**

Definition: A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out bythe appropriate **scheduler**.

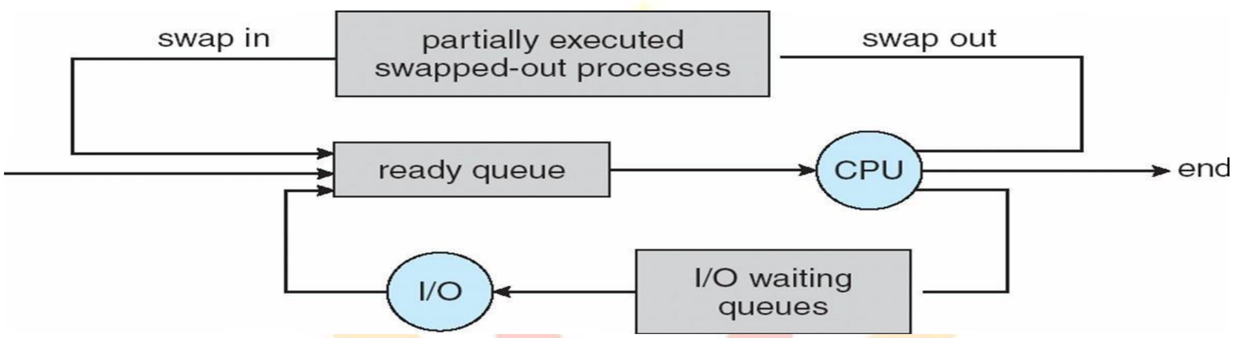
TypesofSchedulers

a. **Long-TermScheduler**or**JobScheduler**
   - Often, in a batch system, more processes are submitted than can be executedimmediately. These processes are spooled toa mass-storage device (typically a disk), where they are kept for later execution.
   - The **long-term scheduler**, or **job scheduler**, selects processes from this pool and loads them into memory for execution.
   - The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next.
   - The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).
   - Ifthedegreeofmultiprogrammingisstable,thentheaveragerateofprocesscreationmust be equal to the average departure rate of processes leaving the system. Thus, the long- term scheduler may need to be invoked only when a process leaves the system.
   - Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.It is important that the long-term scheduler select a good ***process mix*** of I/O-bound and CPU-bound processes.
   - Onsomesystems,thelong-termschedulermaybeabsentorminimal.

b. **Short-TermScheduler**,Or**CPUScheduler**
   - The**short-termscheduler**,or **CPU scheduler**,selectsfromamongthe processes that are readyto execute and allocates the CPU to one of them.
   - Theshort-termschedulermustselectanewprocessfortheCPUfrequently.
   - AprocessmayexecuteforonlyafewmillisecondsbeforewaitingforanI/Orequest. Often, the short-term scheduler executes at least once every 100 milliseconds.
   - Becauseoftheshorttimebetweenexecutions,the short-termschedulermustbefast.

c. **Medium-TermScheduler**
   - Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.
   - The key idea behind a medium-term scheduler is that sometimes it can be advantageousto remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.
   - Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called **swapping**.

- The process is swapped out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiringmemory to be freed up.



3. Context Switch

Definition: Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**.

Whenacontextswitchoccurs,thekernelsavesthecontextoftheoldprocessinitsPCB and loads the saved context of the new process scheduled to run.

Overhead: Context-switch time is pure overhead, because the system does no useful workwhile switching.

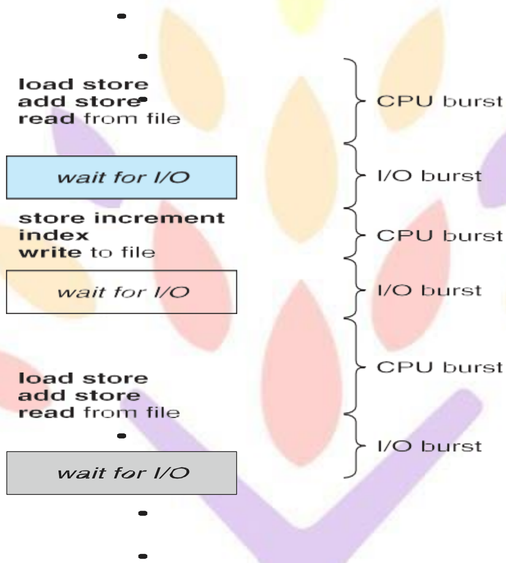
Switching Speed: Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). A typical speed is a fewmilliseconds.

Hardware Support: Context-switch times are highly dependent on hardware support. A context switchhere simply requires changing the pointer to the current register set. Of course,if there are moreactiveprocessesthanthereareregistersets,thesystemresortstocopying registerdatatoand from memory, as before. Also, the more complex the operating system, the greater the amount of work that must be done during a context switch

your roots to success...

4. CPU–I/OBurstCycle

The success of CPU scheduling depends on an observed property ofprocesses: process execution consists of a **cycle** of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a **CPU burst**. That is followed byan**I/O burst**, which is followed byanother CPU burst, then another I/O burst, and so on. Eventually, the finalCPU burst ends with a system request to terminate execution.



DefinitionofNonPreemptiveScheduling

Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method was used by Microsoft Windows 3.x.

DefinitionofPreemptiveScheduling

Under this, a running process may be replaced by higher priority process at any time. Used from Windows 95 to till now. Incurs the cost associated with access to shared data. It also affects the design of OS.

Dispatcher

Another component involved in the CPU-scheduling function is the **dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the short- term scheduler. This function involves the following:

- Switchingcontext
- Switchingtousermode
- Jumpingtotheproperlocationintheuser programtorestartthatprogram

Thedispatchershouldbeasfastaspossible,sinceitisinvokedduringevery process switch.

DispatchLatency:Thetimeittakesforthedispatchertostop oneprocessandstartanotherrunning is known as the **dispatch latency**.

Operationsonprocesses(OR)Systemcallinterfaceforprocessmanagement-fork,exit,wait, waitpid,exec

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination.

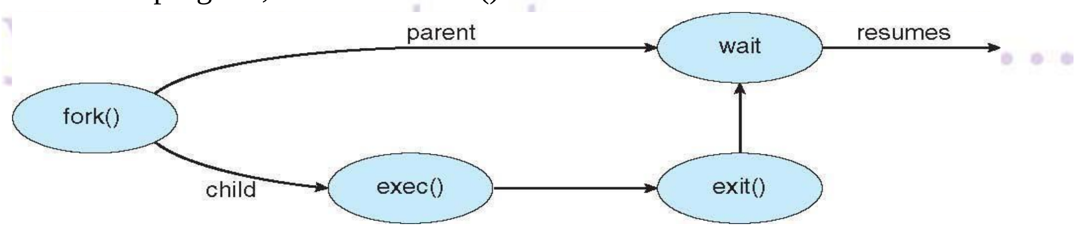
a. ProcessCreation

During the course of execution, a process may create several new processes. The creating process is called a parent process, and the new processes are called the children of that process. Each of these new processes may in turn create other processes, forming a **tree** of processes.

SystemCalls

- **fork()**
  - Most operating systems (including UNIX, Linux, and Windows) identify processes according to a unique **process identifier** (or **pid**), which is typically an integer number.
  - Anewprocess is created bythefork ()system call. Thenewprocess consists ofa copyof the address space of the original process.
  - This mechanism allows the parent process to communicate easily with its child process. Both processes (the parent and the child) continue execution at the instruction after the fork (), with one difference: thereturn code for thefork () is zero for the new (child) process, whereas the (nonzero) process identifier of the childis returned to the parent.
- **exec()**
  - After a fork () system call, one of the two processes typically uses the exec () systemcallto replace the process's memoryspace with a new program.
  - The exec () system call loads a binary file into memory and starts its execution. In this manner, the two processes are able to communicate and then go their separate ways.

- **wait()**
  - The parent can then create more children; or, if ithas nothing else to do while the child runs, it can issue a wait () system call to move itself off the ready queue until the termination of the child. Because the call to exec () overlays the process's address space with a new program, the call to exec () does not return control unless an error occurs.



b. ProcessTermination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit () system call. At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call). All the resources of the process—including physical and virtual memory, open files and I/O buffers—are deallocated by the operating system.

Termination can occur in other circumstances as well. A process can cause thetermination of another process via an appropriate system call (for example, TerminateProcess() in Windows). Usually, such a system call can be invoked only by the parent of the process that is to be terminated. Otherwise, users could arbitrarily kill each other's jobs.

COOPERATINGPROCESSES

- TheconcurrentprocessexecutingintheOSmaybeeitherindependent process or cooperating process.
- Independentprocess**cannot**affectorbeaffectedbytheexecutionofanother process.
- Cooperatingprocess**can**affectorbeaffectedbytheexecutionofanotherprocess.

Advantagesofprocesscooperation

1. **Informationsharing:**severalusersmaybeinterestinthesamepieceof information.
2. **Computationspeed-up:**Ifwewantaparticulartasktorunfaster,wemust break it into subtasks and run in parallel.
3. **Modularity:**Constructing thesystem inmodularfashion,dividing thesystem functions into separate process.
4. **Convenience:**Userwillhavemanytaskstoworkinparallel(Editing,compiling, printing).

Processescancommunicatewitheachotherthroughboth:

- SharedMemory
- Messagepassing

The following figure shows a basic structure of communicationbetweenprocesses via the shared memory method and via the message passing method.

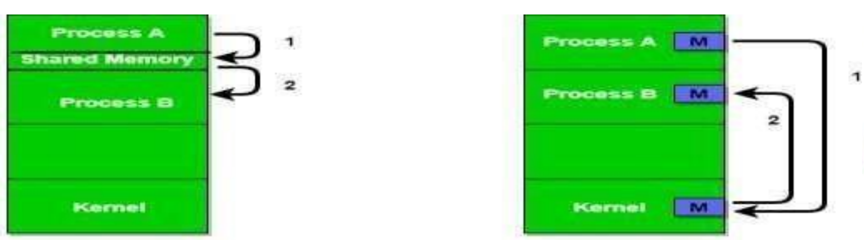


Figure 1 - Shared Memory and Message Passing

(i) SharedMemory

Communication between processes using shared memory requires processestosharesomevariable, and itcompletelydependson howtheprogrammerwill implement it.

One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and theyshare some resources or use some information from another process. Process1 generates informationaboutcertain computationsorresourcesbeingused and keepsitasarecord in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly.

Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Ex:Producer-Consumerproblem

A producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.

A producer can produce one item while the consumer is consuming another item. The Producer and Consumer must be synchronized. The consumer does not tryto consume an item, the consumer must wait until an item is produced.

Unbounded-Buffer

- nopractical limit onthesizeof thebuffer.
- Producercan produceanynumberofitems.
- Consumermayhave towait

Bounded-Buffer

- assumesthatthereisa fixedbuffersize.

Bounded-Buffer–Shared-Memory Solution:

Shared data

```
#define BUFFER_SIZE 10
Typedef struct
{
.. .
}item;
itembuffer[BUFFER_SIZE];
int in = 0;
intout=0;
```

Bounded-Buffer–Producer Process:

```
itemnextProduced;
while (1)
{
 while(((in+1)% BUFFER_SIZE)==out);                    /*donothing
 */buffer[in]= nextProduced;
 in=(in+1)%BUFFER_SIZE;
}
```

Bounded-Buffer–ConsumerProcess:

```
itemnextConsumed;
while (1)
{
 while(in==out);                                       /*donothing
 */next Consumed = buffer[out];
 out=(out+1)%BUFFER_SIZE;
}
```

(ii) MessagingPassingMethod

In this method, processes communicate with eachother without usinganykind of sharedmemory. Iftwoprocesseswanttocommunicatewitheachother, theyproceed as follows

:

- Establish a communication link (if a link alreadyexists,noneedto establish it again.)
- Startexchangingmessagesusingbasicprimitives.
- Themessagesizecanbeoffixedsizeorofvariablesize.Ifitisoffixed size,itiseasy foranOSdesignerbutcomplicatedforaprogrammerandifit is of variable size then it is easy for a programmer but complicated for the OS designer.
- Cooperating process to communicate with each other via an inter process communication (IPC).
- IPC provides a Mechanism to allow processes to communicate and tosynchronize their actions.
- IfPandQwanttocommunicate,acommunicationlinkexistsbetweenthemand

exchangemessages viasend/receive.OSprovidesthis facility.

- IPCfacilityprovidestwooperations:

  Send (*message*)–messagesizefixedorvariable.

  Receive(*message*)

- Implementationofcommunicationlink byfollowing.
  - physical(e.g.,sharedmemory,hardwarebus)
  - logical(e.g.,logicalproperties)

Methodsforlogicalimplementationof a link

i. Directcommunication.
ii. Indirectcommunication.

<u>DirectCommunication</u>

- Eachprocessesmust nameeachotherexplicitly:
  - Send(*P*,*message*) –sendamessagetoprocessP.
  - Receive(*Q*,*message*)–receiveamessagefromprocessQ.
- Linksareestablishedautomatically.
- Alinkis associatedwith exactlyonepairofcommunicatingprocesses.
- Betweeneach pair thereexistsexactlyonelink.
- Thelink maybeunidirectional, but is usuallybi-directional.
- Thisexhibitsboth symmetryand asymmetryin addressing

  Symmetry:

  Boththesenderandthereceiverprocessesmustnametheotherto communicate.

  Asymmetry:

  Onlysendernamestherecipient,therecipientisnotrequiredtonamethesender. The send and receive primitives are as follows.

  - Send(P,message)–sendamessagetoprocess P.
  - Receive(id,message)–receiveamessagefromanyprocess.

Disadvantageofdirectcommunication

Changinga name oftheprocesscreatesproblems.

<u>IndirectCommunication</u>

- Themessagesaresentandreceivedfrommailboxes(alsoreferredtoas ports).
- Amailbox isanobject
- Processcanplace messages.
- Processcanremovemessages.

your roots to success...

- Twoprocesses cancommunicate onlyiftheyhaveasharedmailbox.
- Primitivesaredefinedas:
  send(*A*,*message*)–sendamessagetomailboxA
  receive(*A*,*message*)–receiveamessagefrommailbox A.
- Amailboxmaybeowned either byaprocess orbythe OS.
- If the mailbox is owned by a process, then we distinguish b/w the owner (who can only receive msg through this mailbox) and the user (who can only send msg to the mailbox).
- Amailboxmaybe owned bytheOSis independentand provide amechanism,
  - createa mailbox
  - receivemessagesthroughmailbox
  - destroyamailbox.

Mailboxsharingproblem

TheprocessesP1,*P2*,and*P3*allsharemailboxA.*ProcessesP1*,sends;*P2* and*P3*receivethemessagefromA.Who gets amessage?

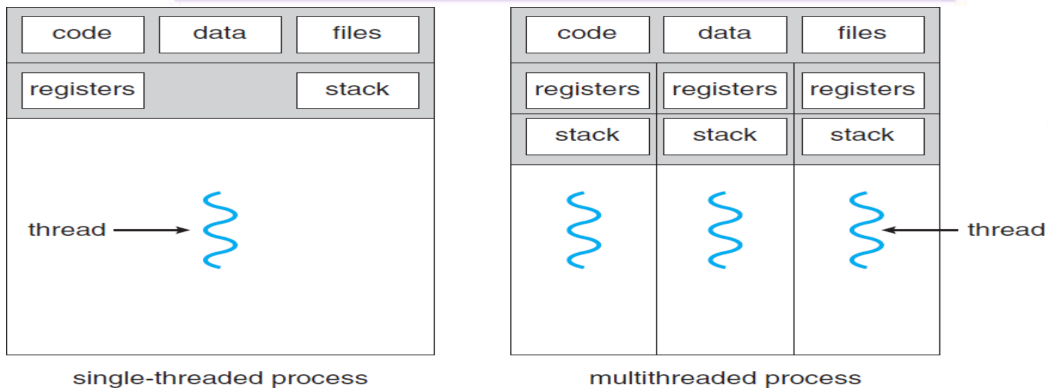
Solutions

- Allowalinktobeassociatedwith at most two processes.
- Allowonlyoneprocessat atimetoexecuteareceive operation.
- Allow the system to select arbitrarily the receiver. The system may identifythe receiver to the sender.

DefiningThread

AthreadisaLightweightprocess.Threadisaflowofcontrolexecutionofthe program.
Threadisa basicunit of CPU utilization; it comprisesa thread ID, a program counter, a register set,and astack. 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.



SingleThread

- Aprocessisaprogramthatperformsasingle**thread**ofexecution.
- For example, when a process is running a word-processor program, a single thread of instructions is being executed.

- Thissinglethreadofcontrolallowstheprocesstoperformonlyonetaskat atime.The usercannot simultaneously type in characters and run the spell checker within the same process, for example.Multi Thread
  - Most modern operating systems have extended the process concept to allow a process tohave multiple threads of execution and thus to perform more than one task at a time.
  - This feature is especially beneficial on multicore systems, where multiple threads can runinparallel.
  - Onasystemthatsupportsthreads,thePCBisexpandedtoincludeinformationforeach thread. Other changes throughout the system are also needed to support threads.

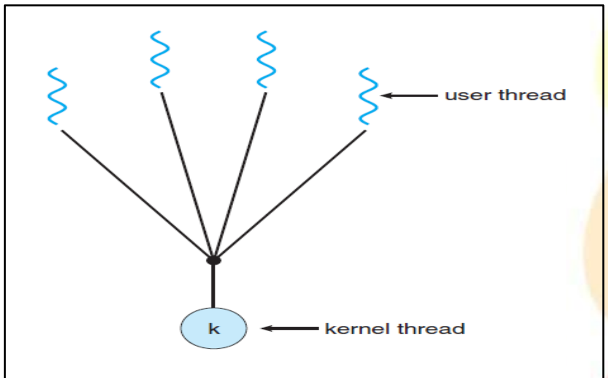
MultithreadingModels

Support for threads may be provided either at the user level, for **user threads**, or by the kernel, for **kernel threads**. User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system. Virtually all contemporary operating systems—including Windows, Linux, Mac OS X, and Solaris support kernel threads.

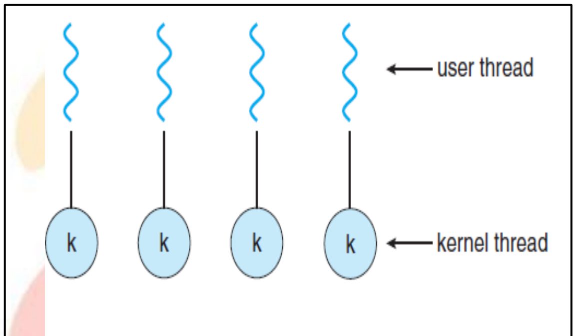
Ultimately, a relationship must exist between user threads and kernel threads. The following are the three common ways of establishing such a relationship: the many-to-one model, the one-to-one model, and the many-to many models.

1. **Many-to-OneModel**
   - Themany-to-onemodelmapsmanyuser-levelthreadstoonekernelthread.
2. **One-to-OneModel**
   - Theone-to-onemodelmapseachuserthread toakernelthread.
3. **Many-to-ManyModel**
   - Itmultiplexesmanyuser-level threads toasmalleror equal numberof kernelthreads.

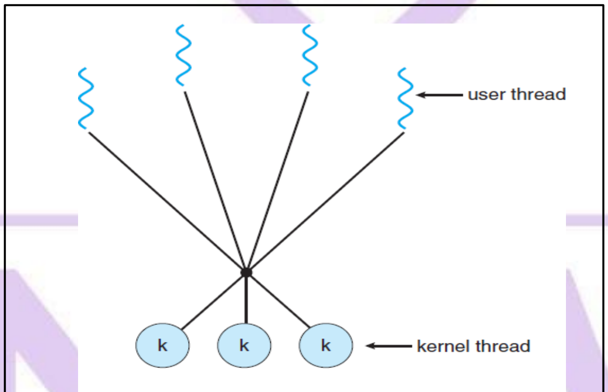
your roots to success...



Many-to-one model.



One-to-one model.



Many-to-many model.

your roots to success...

UNIT-2

CPU scheduling- Scheduling criteria, Scheduling Algorithms, Multiple Processor Scheduling, System call interface for process management-fork, exit, wait, waitpid, exec.

Deadlocks- system Model, Deadlock Characterization, Methods for handling deadlocks, Deadlock Prevention, Deadlock Avoidance, deadlock Detection, and recovery from deadlock

CPUscheduling

CPUschedulingistheprocessofdecidingwhichprocesswillownthe CPU to use while another process is suspended. The main function of the CPU schedulingistoensurethatwhenevertheCPUremainsidle,theOShasatleast selected one of the processes available in the ready-to-use line.

In Multiprogramming, if the long-term scheduler selects multiple I / O binding processesthenmostofthetime,theCPUremainsanidle.Thefunctionofan effective program is to improve resource utilization.

If most operating systems change their status from performance to waiting then there may always be a chance of failure in the system. So in ordertominimize this excess, the OS needs to schedule tasks in order to make full use of the CPU and avoid the possibility of deadlock.

ObjectivesofProcessSchedulingAlgorithm

- UtilizationofCPUatmaximumlevel.KeepCPUasbusyaspossible.
- AllocationofCPUshouldbefair.
- ThroughputshouldbeMaximum.i.e.Numberofprocessesthatcomplete their execution per time unit should be maximized.
- Minimumturnaroundtime,i.e.timetakenbyaprocesstofinishexecution should be the least.
- Thereshouldbea minimumwaitingtime andtheprocessshouldnotstarve in the ready queue.
- Minimum responsetime. Itmeansthatthetimewhenaprocessproducesthe first response should be as less as possible.

Terminologies

- **ArrivalTime:**Timeatwhichtheprocessarrivesinthereadyqueue.
- **CompletionTime:**Timeatwhichprocesscompletesitsexecution.
- **BurstTime:**TimerequiredbyaprocessforCPUexecution.
- **TurnAroundTime:**TimeDifferencebetweencompletiontimeandarrival time.

  TurnAroundTime=CompletionTime–ArrivalTime
- **WaitingTime(W.T):**TimeDifferencebetweenturnaroundtimeandburst

time.

WaitingTime=TurnAroundTime–BurstTime

THESCHEDULINGCRITERIA

CPUutilization:

Themain purposeofanyCPUalgorithm isto keep theCPUas busyas possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.

Throughput:

The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.

TurnroundTime:

For a particular process, the important conditions are how long it takes to performthatprocess.Thetimeelapsedfromthetimeofprocessdelivery tothe timeofcompletionisknownastheconversiontime.Conversiontimeisthe amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

WaitingTime:

The Scheduling algorithm does not affect the time required to completethe process once it has started performing. It only affects the waiting time oftheprocess i.e. the time spent in the waiting process in the ready queue.

ResponseTime:

In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

TypesofCPUSchedulingAlgorithms

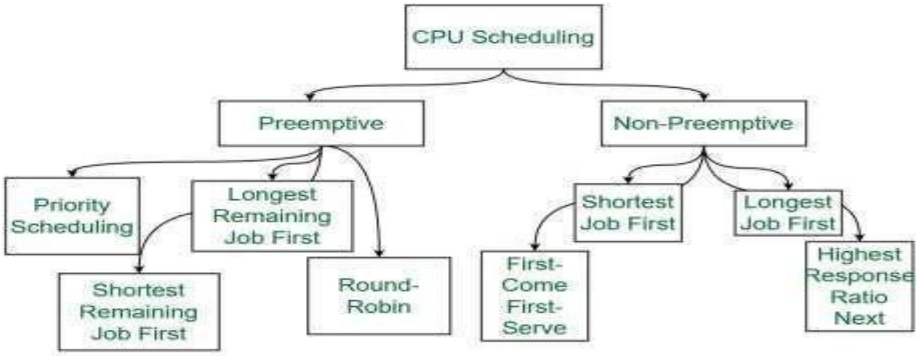
Therearemainly twotypesofschedulingmethods:

PreemptiveScheduling:

Preemptiveschedulingisusedwhenaprocessswitchesfromrunningstateto ready state or from the waiting state to the ready state.

Non-PreemptiveScheduling:

Non-Preemptiveschedulingisusedwhenaprocessterminates,orwhena process switches from running state to waiting state.



1. FirstComeFirstServeScheduling:

FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithmstatesthattheprocessthat requeststheCPUfirstisallocatedtheCPUfirstandisimplementedbyusingFIFO queue.

Characteristics:

- FCFS supports non-preemptive and preemptive CPU schedulingalgorithms.
- TasksarealwaysexecutedonaFirst-come,First-serveconcept.
- FCFSiseasytoimplementanduse.
- Thisalgorithmisnotmuchefficientinperformance,andthewaittimeisquite high.

Advantages:

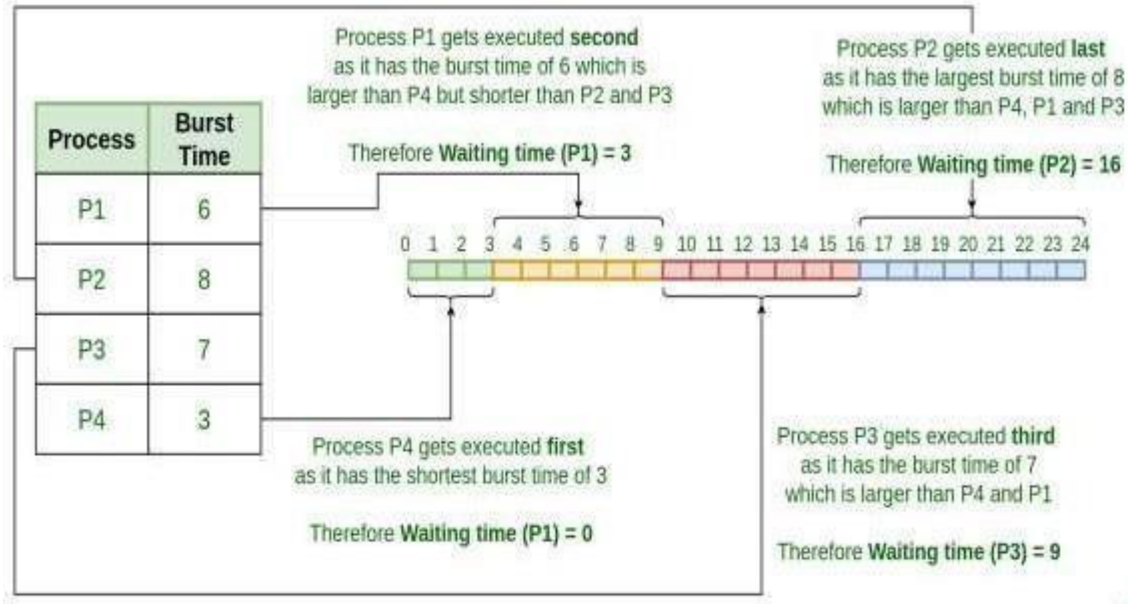
- Easy toimplement
- Firstcome,firstservemethod

Disadvantages:

- FCFSsuffersfrom**Convoyeffect**.
- Theaveragewaitingtimeismuchhigherthantheotheralgorithms.
- FCFSisvery simpleandeasy toimplementandhencenotmuchefficient.

2. ShortestJobFirst (SJF)Scheduling:

Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method mayormay notbepreemptive.Significantlyreducestheaveragewaitingtime forother processes waiting to be executed. The full form of SJF is Shortest Job First.



Characteristics:

- Shortest Job first has the advantage of having a minimum average waitingtime among all operating system scheduling algorithms.
- Itisassociatedwitheachtaskasaunitoftimetocomplete.
- Itmaycausestarvationifshorterprocesseskeepcoming.Thisproblemcan be solved using the concept of ageing.

Advantages:

- AsSJFreducestheaveragewaitingtimethus,itisbetterthanthefirst come first serve scheduling algorithm.
- SJFisgenerallyusedforlongtermscheduling

Disadvantages:

- OneofthedemeritSJFhasisstarvation.
- Manytimesitbecomescomplicatedtopredictthelengthoftheupcoming CPU request

3. LongestJobFirst(LJF)Scheduling:

This is just opposite of shortest job first (SJF), as the name suggests this algorithmisbaseduponthefactthattheprocess withthelargestbursttime is processed first. Longest Job First is non-preemptive in nature.

Characteristics:

- Among all the processes waiting in a waiting queue, CPU is alwaysassigned to the process having largest burst time.
- Iftwoprocesseshavethesamebursttimethenthetieisbrokenusing FCFSi.e. the process that arrived first is processed first.

- LJFCPUSchedulingcanbeofbothpreemptiveandnon-preemptivetypes.

Advantages:

- Noothertaskcanscheduleuntilthelongestjoborprocessexecutes completely.
- Allthejobsorprocessesfinishatthesametimeapproximately.

Disadvantages:

- Generally, the LJF algorithm gives a very high average waitingtime and average turn-around time for a given set of processes.
- Thismayleadtoconvoyeffect.

4. PriorityScheduling:

Preemptive Priority CPU Scheduling Algorithm is apre-emptivemethod of CPU scheduling algorithm that works **based on the priority** of aprocess. In this algorithm, the editor sets the functions to be as important,meaning that the most importantprocessmustbedonefirst.Inthecaseofanyconflict, thatis,where there are more than one processor with equal value, thenthe most important CPU planning algorithm works on the basis of the FCFS **Characteristics:**

- Schedulestasksbasedonpriority.
- Whenthehigherpriorityworkarriveswhileataskwith lesspriorityis executed, the higher priority work takes the place of the less priority one and
- Thelatterissuspendeduntiltheexecutioniscomplete.
- Loweristhenumberassigned,higheristheprioritylevelofaprocess.

Advantages:

- TheaveragewaitingtimeislessthanFCFS
- Lesscomplex

Disadvantages:

- One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the Starvation Problem. This is the problem in which a process has to wait for a longer amount of time to get scheduled into the CPU. This condition is called the starvation problem.

5. RoundRobinScheduling:

Round Robin is a CPU scheduling algorithm where each process is cyclically assignedafixedtimeslot.ItisthepreemptiveversionofFirstcomeFirstServe CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

Characteristics:

- It'ssimple,easytouse,andstarvation-freeasallprocessesgetthebalanced CPU allocation.
- OneofthemostwidelyusedmethodsinCPUschedulingasacore.

- ItisconsideredpreemptiveastheprocessesaregiventotheCPUfora very limited time.

Advantages:

- Roundrobinseemstobefairaseveryprocessgetsanequalshareof CPU.
- Thenewly createdprocessisaddedtotheendofthereadyqueue.

6. ShortestRemainingTimeFirstScheduling(SRTF):

SRTF is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated tothejobclosesttocompletion. In SRTF the process with thesmallest amountoftime remaining until completion is selected to execute.

Characteristics:

- SRTF algorithm makes the processing of the jobs fasterthanSJF algorithm, given it's overhead charges are not counted.
- The context switch is done a lot more times in SRTF than in SJF and consumes the CPU's valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.

Advantages:

- InSRTFtheshortprocessesarehandledvery fast.
- Thesystemalsorequiresverylittleoverheadsinceitonlymakesadecision when a process completes or a new process is added.

Disadvantages:

- Liketheshortestjobfirst,italsohasthepotentialforprocessstarvation.
- Long processes maybe held off indefinitelyif short processes are continually added.

7. LongestRemainingTimeFirst:

Thelongestremainingtimefirst isapreemptiveversionofthelongestjob firstschedulingalgorithm.Thisschedulingalgorithmisusedbytheoperatingsystem to program incoming processes for use inasystematicway.This algorithm schedules those processes first which have the longest processing time remaining for completion.

Characteristics:

- Amongalltheprocesseswaitinginawaitingqueue,theCPUisalways assigned to the process having the largest burst time.
- Iftwoprocesseshavethesamebursttimethenthetieisbrokenusing FCFS i.e. the process that arrived first is processed first.
- LJFCPUSchedulingcanbeofbothpreemptiveandnon-preemptivetypes.

Advantages:

- Nootherprocesscanexecuteuntilthelongesttaskexecutescompletely.

- Allthejobsorprocessesfinishatthesametimeapproximately.

Disadvantages:

- Thisalgorithmgivesaveryhighaveragewaitingtimeandaverageturn- around time for a given set of processes.
- Thismayleadtoaconvoyeffect.

8. HighestResponseRatioNext:

Highest Response Ratio Next is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms. The name itself statesthatweneedtofindtheresponseratioofallavailableprocessesandselectthe one with the highest Response Ratio. A process once selected will run till completion.

Characteristics:

- The**criteria**for HRRN is**Response Ratio**and the **mode** is **NonPreemptive.**
- HRRNisconsideredasthemodificationofShortestJobFirst toreducethe problem of starvation.

- In comparison with SJF, during the HRRN scheduling algorithm, the CPUis allotted to the nextprocess which has the **highestresponseratio** and not tothe process having less burst time.

ResponseRatio=(W+S)/S

Here,**W**-Waitingtimeoftheprocess

S-Bursttimeoftheprocess.

Advantages:

- HRRNSchedulingalgorithmgenerallygives better performance thanthe shortest job first Scheduling.
- Thereisareductioninwaitingtimeforlongerjobsandalsoitencourages shorter jobs.

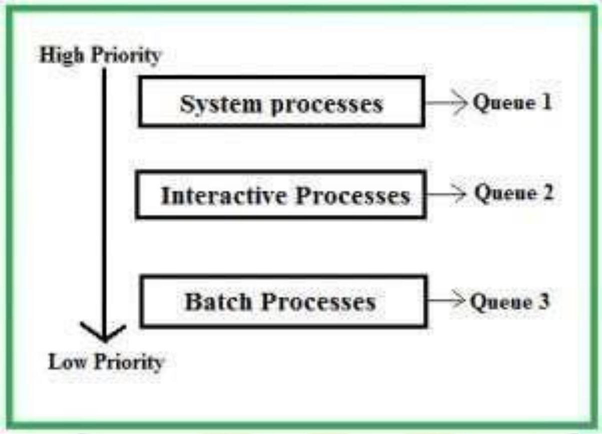
Disadvantages:

- The implementation of HRRN scheduling is not possible as it is notpossible to know the burst time of every job in advance.
- Inthisscheduling,theremayoccuranoverloadontheCPU.

your roots to success...

9. MultipleQueueScheduling:

Processes in the ready queue can be divided into different classes where each classhasitsownschedulingneeds.Forexample,acommondivisionis a **foreground (interactive)** process and a **background(batch)** process.These two classes have different scheduling needs. For this kind of situation **Multilevel Queue Scheduling** is used.



Thedescriptionoftheprocessesintheabovediagramisasfollows:

- **System Processes:** The CPU itself has its process to run, generally termedas System Process.
- **Interactive Processes:** An Interactive Process is a type of process inwhich there should be the same type of interaction.

- **Batch Processes:** Batch processing is generally a technique in the Operating system that collectsthe programs and data together in the form of a **batch** before the **processing** starts.

Advantages:

- Themainmeritofthemultilevelqueueisthatithasalowscheduling overhead.

Disadvantages:

- Starvationproblem
- Itisinflexibleinnature

10. MultilevelFeedbackQueueScheduling:

Multilevel Feedback Queue Scheduling (MLFQ) CPUSchedulingis like **Multilevel Queue Scheduling** but in this process can move between the queues. And thus, much more efficient than multilevel queue scheduling.

Characteristics:

- In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system, and processes are not allowedto move between queues.
- As the processes are permanently assigned to the queue, this setup has the advantage of low scheduling overhead,
- Butontheotherhanddisadvantageofbeinginflexible.

Advantages:

- Itismoreflexible

- Itallowsdifferentprocessestomovebetweendifferentqueues

Disadvantages:

- ItalsoproducesCPUoverheads
- Itisthemostcomplexalgorithm.

ComparisonbetweenvariousCPUSchedulingalgorithms

HereisabriefcomparisonbetweendifferentCPUschedulingalgorithms:

Algorithm	**Allocationis**	**Complexity**	**Average waiting time (AWT)**	**Preemption**	**Starvation**	**Performance**
FCFS	According to the arrival time of the processes,the CPUis allocated.	Simple and easy to implement	Large.	No	No	Slow
SJF	Basedonthelowest CPUbursttime (BT).	More complex thanFCFS	Smaller than FCFS	No	Yes	Good
SRTF	Same as SJF the allocationoftheCPU is based on thelowest CPUbursttime(BT). But it is preemptive.	More complex thanFCFS	Dependingon arrival time, process size	Yes	Yes	Good
RR	According to the order of the process arrives with fixed time quantum (TQ)	The complexity dependson TQ	Large than SJF and Priority scheduling.	Yes	No	Fair
Priority Pre-emptive	According to the priority. The bigger priority task executes first	Less complex	Smaller than FCFS	Yes	Yes	Well

your roots to success...

Priority non-preemptive	According to the priority with monitoring the new incoming higher priorityjobs	Lesscomplexthan Priority preemptive	Smaller than FCFS	No	Yes	Most beneficial withbatch systems

Algorithm	**Allocationis**	**Complexity**	**Average waiting time (AWT)**	**Preemption**	**Starvation**	**Performance**
MLQ	According to the process that residesin the bigger queue priority	More complex thanthe priority	Smaller than FCFS	No	Yes	Good
MLFQ	According to the process of a bigger priorityqueue.	Itisthemost Complex	Smaller than all scheduling	No	No	Good

Example1(FCFS)

1.Process ID	ProcessName	BurstTime(ms)
P1	A	6
P2	B	2
P3	C	1
P4	D	9
P5	E	8

GanttChart

P 1	P 2	P 3	P 4	P 5

0 6 8 9 18 26

ProcessID	**Arrival Time(ms)**	**Burst Time (ms)**	**Completion Time (ms)**	**Turn Around Time (ms)**	**Waiting Time (ns)**
P1	0	6	6	6	0
P2	2	2	8	8	6

P3	3	1	9	9	8
P4	4	9	18	18	9
P5	5	8	26	26	18

AverageTurnAroundTime=(6+8 +9+18 +26)/5=67 /5=13.4ms
AverageWaitingTime =(0 +6 +8+9 +18)/ 5 =41 / 5=8.2ms

Example2(FCFS)

ProcessID	ProcessName	Burst Time
P1	A	79
P2	B	2
P3	C	3
P4	D	1
P5	E	25
P6	F	3

ProcessId	**BurstTime (BT)**	**CompletionTime (CT)**	**Turn Around Time(TAT)**	**Waiting Time(WT)**
P1	79	79	79	0
P2	2	81	81	79
P3	3	84	84	81
P4	1	85	85	84
P5	25	110	110	85
P6	3	113	113	110

AvgWaitingTime=(0 +79+81 +84 +85+110)/6 =73.17ms
AvgTurnAroundTime= (79 +81+84+85+110+113)/ 6=92 ms

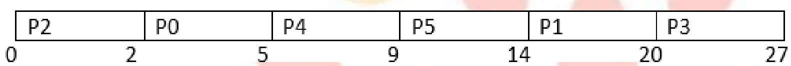
your roots to success...

Example3(SJF)

ProcessID	Arrival Time	BurstTime
P0	1	3
P1	2	6
P2	1	2
P3	3	7
P4	2	4
P5	5	5

NonPre-Emptive ShortestJobFirstCPUScheduling

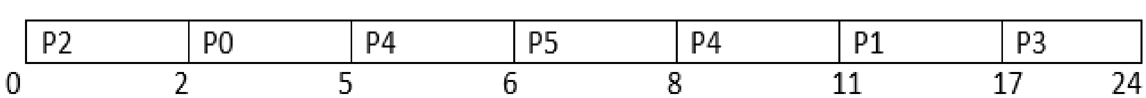
GanttChart:



ProcessID	**Arrival Time**	**Burst Time**	**Completion Time**	**TurnAround Time TAT=CT–AT**	**Waiting Time WT=CT–BT**
P0	1	3	5	4	1
P1	2	6	20	18	12
P2	0	2	2	2	0
P3	3	7	27	24	17
P4	2	4	9	7	4
P5	5	5	14	10	5

AverageWaitingTime=(1+12 +17 +0+5+4)/6=39/ 6=6.5 ms
AverageTurnAround Time=(4+18+2+24+7+10)/6=65/6=10.83ms Pre Emptive Shortest Job First CPU Scheduling

Ganttchart:



Process ID	Arrival Time	Burst Time	Completion Time	TurnAroundTime TAT=CT-AT	WaitingTime WT=CT–BT
P0	1	3	5	4	1
P1	2	6	17	15	9
P2	0	2	2	2	0
P3	3	7	24	21	14
P4	2	4	11	9	5
P5	6	2	8	2	0

AverageTurn AroundTime =(4 +15+2 +21+9+2)/ 6=53/ 6= 8.83 ms
AverageWaitingTime =(1+9+0 +14+5+0)/6 =29/ 6= 4.83 ms

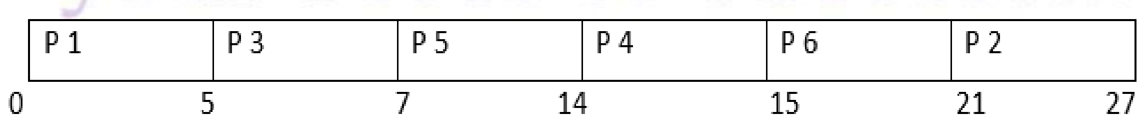
Example4(PRIORITY)

S.No	ProcessID	ArrivalTime	BurstTime	Priority
1	P1	0	5	5
2	P2	1	6	4
3	P3	2	2	0
4	P4	3	1	2
5	P5	4	7	1
6	P6	4	6	3

(5has theleastpriorityand 0has thehighest priority)

Solution:

Gantt Chart:



Process Id	Arrival Time	Burst Time	Priority	Completion Time	TurnAround Time TAT=CT-AT	Waiting Time WT=TAT-BT
P1	0	5	5	5	5	0
P2	1	6	4	27	26	20
P3	2	2	0	7	5	3
P4	3	1	2	15	12	11
P5	4	7	1	14	10	3
P6	4	6	3	21	17	11

AvgWaiting Time =(0+20+3 +11+3 +11)/ 6 =48 / 6= 8 ms
AvgTurnAroundTime=(5 +26+5 +11+ 10 +17)/ 6=74 / 6=12.33ms

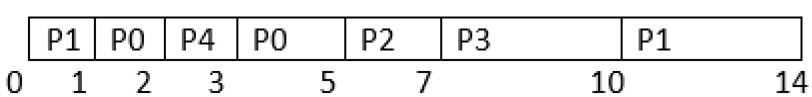
Example5(RoundRobin)

TimeQuantum=1 ms

ProcessID	Arrival Time	Burst Time
P0	1	3
P1	0	5
P2	3	2
P3	4	3
P4	2	1

Solution:

Ganttchart:



ProcessID	Arrival Time	Burst Time	Completion Time	Turn AroundTime	Waiting Time
P0	1	3	5	4	1
P1	0	5	14	14	9
P2	3	2	7	4	2
P3	4	3	10	6	3
P4	2	1	3	1	0

AvgTurnAroundTime=(4+14+4+6+1)/5=5.8 ms
AvgWaitingTime =(1+9+2+3+0)/5=3 ms

DEADLOCK

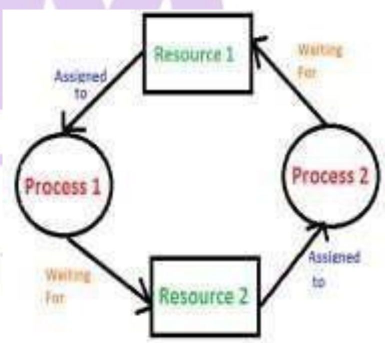
Aprocessinoperatingsystemusesresourcesinthefollowingway.

(i) Requestsaresource
(ii) Usetheresource
(iii) Releasestheresource

A ***deadlock*** is a situation where a set of processes are blocked because each processis holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they arein front of each other.

A similar situation occurs in operating systems when there are two or more processes that holdsome resources and wait for resources heldby other(s). For example, in the below diagram, Process1 is holding Resource1 and waiting for Rsource2 which is acquired by Process2, and Process2 is waiting for Resource1.



ExamplesofDeadlock

1. Thesystemhas2tapedrives.P1andP2eachholdonetapedriveandeachneeds

anotherone.

2. SemaphoresAandB,initializedto1,P0,andP1arein deadlock as follows:
   P0executeswait(A)andpreempts.P1 executes wait(B).
   NowP0andP1enterindeadlock.

P0	**P1**
wait(A);	wait(B)
wait(B);es ndthefo	wait(A) llowing

3. Assumethespaceisavailableforallocationof200Kbyt sequence of events occurs.

P0	P1
Request 80KB;	Request 70KB;
Request 60KB;	Request 80KB;

Systemmodel:

A system consists of a finite number of resources to be distributed amonganumberofcompetingprocesses. The resources are partitioned into several types, each consisting of some number ofidenticalinstances.Memoryspace,CPUcycles,files,I/Odevicesareexamplesofresourcetypes.Ifa systemhas2CPUs,thenthe resourcetypeCPU has 2instances.

Aprocessmustrequestaresourcebeforeusingitandmustreleasetheresourceafterusing it.Aprocessmayrequestasmanyresourcesasitrequirestocarryoutitstask.Thenumber ofresourcesasitrequirestocarryoutitstask.Thenumberofresourcesrequestedmaynot exceedthetotalnumberofresourcesavailableinthesystem.Aprocesscannotrequest3 printers if thesystemhas onlytwo.

Aprocessmayutilizearesourceinthefollowingsequence:

(I) **REQUEST**: The process requests the resource. If the request cannot be granted immediately(iftheresourceisbeingusedbyanotherprocess),thentherequestingprocessmustwaituntil itcan acquiretheresource.

(II) **USE**: The processcanoperateonthe resource.ifthe resourceisa printer,the processcanprintontheprinter.

(III) **RELEASE**:Theprocessreleasestheresource.

For each use of a kernel managed by a process the operating system checks that the process hasrequested and has been allocated the resource. A system table records whether eachresource isfree (or) allocated. For each resource that is allocated, the table also records the process to whichitisallocated.Ifaprocessrequestsaresourcethatiscurrentlyallocatedtoanother process, itcanbe addedtoa queueofprocesses waitingforthis resource.

Toillustrateadeadlockedstate,considerasystemwith3CDRWdrives.Eachof3 processesholdsoneoftheseCDRWdrives.Ifeachprocessnowrequestsanotherdrive,the3processeswill be in a deadlocked state. Each is waiting for the event "CDRW is released" which can becaused only by one of the other waiting processes. This example illustrates a deadlock involvingthe sameresourcetype.

Deadlocks may also involve different resource types. Consider a system with one printer and oneDVD drive. The process Pi is holding the DVD and process Pj is holding the printer. If PirequeststheprinterandPjrequests the DVDdrive,a deadlockoccurs.

NECESSARYCONDITIONSFORDEADLOCK

- ***MutualExclusion***
  Twoormoreresourcesarenon-shareable(Onlyoneprocesscanuse at a time)
- ***HoldandWait***
  Aprocessisholdingatleastoneresourceandwaitingforresources.
- ***NoPre-emption***
  A resource cannot be taken from a process unless the processreleases the resource.
- ***CircularWait***
  Asetofprocesseswaitingforeachotherincircularform.

ResourceAllocationGraph

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

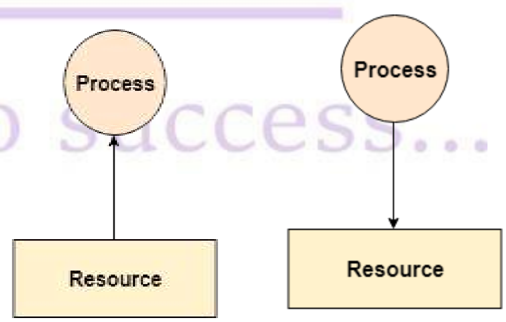
Italsocontainstheinformationaboutalltheinstancesofalltheresources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle.

Vertices are mainly of two types, Resource and Process. Each of them will be represented by a different shape. Circle represents process while rectanglerepresents resource. A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.

EdgesinRAGarealsooftwotypes, one represents **Assignment Edge** and other represents the wait of a process for a resource ie.**Request Edge**.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instanceto theresource andtheheadis attached to a process.



Resource is assigned to process

Process is requesting for a resource

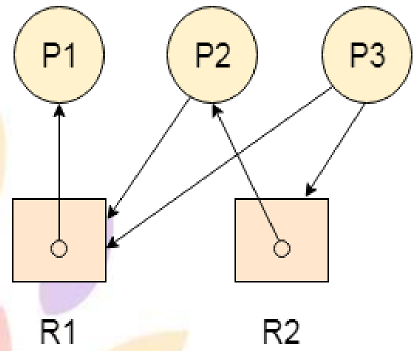
A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

Example

Consider 3 processes P1, P2 and P3 and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 iswaiting forR1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.



Using Resource Allocation Graph, it can be easily detected whether systemis in a Deadlock state or not. The rules are

Rule-01: InaResourceAllocationGraphwherealltheresourcesaresingleinstance,

- If acycleis beingformed,thensystemis inadeadlockstate.
- If nocycleisbeingformed,thensystemisnotinadeadlock state.

Rule-02: InaResourceAllocationGraphwherealltheresourcesare**NOT**single instance,

- Ifacycleisbeingformed, thensystemmaybe inadeadlockstate.
- **Banker's Algorithm**is applied to confirm whether system is in a deadlockstateor not.
- Ifnocycleisbeingformed,thensystemisnotinadeadlock state.
- Presenceofacycleisanecessarybutnotasufficientconditionforthe occurrence of deadlock.

METHODSFORHANDLINGDEADLOCK

Therearethreewaystohandledeadlock

1) **Deadlockpreventionoravoidance**

PREVENTION

Theideaistonotletthesystemintoadeadlockstate.Thissystemwillmake surethat abovementioned fourconditions will notarise. Thesetechniques areverycostly so we use this in cases where our priority is making a system deadlock-free.

One can zoom into each category individually, Prevention is done by negatingone of the four necessary conditions for deadlock.

Eliminatemutualexclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

SolveholdandWait

Allocate all required resources to the process before the start of its execution, thisway hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked tillit has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.

Allowpre-emption

Preempt resources from the process when resources are required by otherhigh-priorityprocesses.

CircularwaitSolution

Each resource will be assigned a numerical number. A process can request the resources to increase/decrease. order of numbering. For Example, if the P1 process is allocated R5 resources, now next time ifP1 asks for R4, R3 lesser than R5 such a request will not be granted, only a request for resources more than R5 will be granted.

AVOIDANCE

Avoidance is kind of futuristic. By using the strategy of "Avoidance", we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process.

ResourceAllocationGraph

The resource allocation graph (RAG) is used to visualize the system‟s current state as a graph. The Graph includes all processes, the resources that are assignedtothem, as well as the resources that each Process requests. Sometimes,if there are fewer processes, we can quickly spot a deadlock in the system by looking at the graph rather than the tables we use in Banker‟s algorithm.

Banker'sAlgorithm

Bankers‟s Algorithm is a resource allocation and deadlock avoidance algorithm whichtestalltherequestmadebyprocessesforresources,itchecksfor the safestate, and after granting a request system remains in the safe state it allows the request, and if there is no safestateit doesn‟t allow the request madeby the process.

Inprevention andavoidance,we get thecorrectness ofdatabutperformance decreases.

2) Deadlockdetectionandrecovery

IfDeadlockpreventionoravoidanceisnotappliedtothesoftwarethenwecan handle this by deadlock detection and recovery,which consist of two phases.

Inthefirstphase,weexaminethestateoftheprocessandcheckwhether there is a deadlock or not in the system.

If found deadlock in the first phase then we apply the algorithmfor recovery of the deadlock.

3) Deadlockignorance:

If a deadlock is very rare, then let it happen and reboot the system. This is the approachthatbothWindowsandUNIXtake.Weusetheostrichalgorithm for deadlock ignorance.

In Deadlock, ignorance performance is better than the abovetwo methods but not the correctness of data.

SAFESTATE

A safe state can be defined as a state in which there is no deadlock. It is achievable if:

- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- Alltherequestedresourcesareallocatedtotheprocess.

BANKER'SALGORITHM

Itisabankeralgorithmusedto **avoiddeadlock** and **allocate resources**safely to each process in the computer system. The '**S-State'**examines all possible testsor activities before decidingwhether the allocation should be allowed to each process. It also helps the operating system to successfully sharethe resources between all the processes.

The banker's algorithm is named because it checks whether a person shouldbe sanctioned a loan amount or not to help the bank system safely simulate allocation resources.

Suppose the number of account holders in a particular bank is 'n', and the total moneyin abank is 'T'. If an account holder applies foraloan; first, thebank subtracts the loan amount from full cash and then estimates the cash difference is greaterthanTto approvetheloanamount.Thesestepsaretakenbecauseifanotherpersonapplies for a loan or withdraws some amount from the bank, it helps the bank manage andoperate all things without any restriction in the functionalityof the banking system.

Similarly, it works in an **operating system**. When a new process is created in a computersystem,theprocessmustprovidealltypesofinformationtothe operating system like upcoming processes, requests for their resources, counting them, and delays.

Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also knownas**deadlock avoidancealgorithm**or**deadlockdetection** in theoperating system.

Whenworkingwithabanker'salgorithm,itrequeststoknowaboutthree things:

1. Howmucheachprocesscanrequestforeachresourceinthesystem.Itis denoted by the [**MAX**] request.
2. Howmucheachprocessiscurrentlyholdingeachresourceinasystem.Itis denoted by the [**ALLOCATED**] resource.
3. Itrepresentsthenumberofeachresourcecurrently availableinthesystem.It is denoted by the [**AVAILABLE**] resource.

Following are the important data structures terms applied in the banker's algorithm as follows:

Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.

1. **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.

2. **Max:** It is a [n x m] matrix that indicates each process P[i] can store themaximum number of resources R[j] (each type) in a system.

3. **Allocation:** It is a matrix of m x n orders that indicates the typeof resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j]in the system.

4. **Need:** It is an M x N matrix sequence representing thenumberof remaining resources for each process. When the Need[i] [j] = k,then process P[i] may require K more instances of resources type Rj to complete the assigned work.

   Need[i][j]=Max[i][j]-Allocation[i][j].

5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

TheBanker'sAlgorithmisthecombinationofthesafetyalgorithmandtheresource request algorithm to control the processes and avoid deadlock.

Safety Algorithm

Itisasafetyalgorithmusedtocheckwhetherornotasystemisinasafestate or follows the safe sequence in a banker's algorithm:

Step1:

There are two vectors**Wok**and**Finish**of length m and n in a safetyalgorithm. Initialize:

Work = Available

Finish[i]=false;forI=0,1,2, 3, 4… n -1.

your roots to success...

Step2:

Checktheavailabilitystatusforeachtypeofresources[i],suchas: Need[i] <= Work

Finish[i]==false

Iftheidoesnotexist,gotostep4. Step3:

Step4:

Work = Work +Allocation(i) //togetnewresourceallocation

Finish[i] = true

Gotostep2tocheckthestatusofresourceavailabilityforthenextprocess.If Finish[i] == true;

it means that the system is safe for all processes.

ResourceRequestAlgorithm

LetcreatearesourcerequestarrayR[i]foreachprocessP[i].

Step1:

Whenthenumberof**requestedresources** ofeachtypeislessthan the **Need** resources, go to step2 and if the condition fails, which means that the processP[i] exceeds its maximum claim for the resource. As the expressionsuggests:

IfRequest(i)<=Need,thengotostep2,Elseraiseanerror message.

Step2:

Andwhenthenumberofrequestedresourcesofeachtypeislessthanthe availableresourceforeachprocess,gotostep(3).Astheexpressionsuggests:If Request(i) <= Available, then go to step3.

ElseProcess P[i] mustwaitfortheresource.

Step3:

Whentherequestedresourceisallocatedtotheprocessbychangingstate: Available = Available – Request

Allocation(i)=Allocation(i)+Request(i) $Need_i$ =

$Need_i$- $Request_i$

When the resource allocation state is safe, its resources are allocated to theprocess P(i). And if the new state is unsafe, the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state.

Example:

Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation A	Allocation B	Allocation C	Max A	Max B	Max C	Available A	Available B	Available C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answerthefollowingquestionsusingthebanker'salgorithm:

1. Whatisthe referenceoftheneed matrix?
2. Determineifthesystemissafeor not.
3. Whatwillhappeniftheresourcerequest(1,0,2)forprocessP1canthe system accept this request immediately?
4. Whatwill happeniftheresourcerequest(3,3,0)forprocessP5?
5. Whatwill happeniftheresourcerequest(0,2,0)forprocessP1?

Ans.1:

Contextoftheneedmatrix isasNeed [i]=Max[i]-Allocation [i]

NeedforP1: (7,5,3)-(0,1,0)=7, 4,3
NeedforP2: (3,2, 2)-(2,0,0)=1, 2,2
NeedforP3: (9,0, 2)-(3,0,2)=6, 0,0
NeedforP4: (2,2, 2)-(2,1,1)=0, 1,1
NeedforP5: (4,3, 3)-(0,0,2)=4, 3,1

Process	Need A	Need B	Need C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Ans.2:ApplytheBanker'sAlgorithm:

AvailableResourcesofA,BandCare3, 3,and 2.

Nowwecheckifeachtypeofresourcerequestisavailableforeachprocess.

Step1:

ForProcessP1:

Need<=Available

7, 4, 3 <= 3, 3, 2 condition is **false**.

So,we examine another process, P2.

Step2:

ForProcessP2:

Need<=Available

1,2, 2<=3,3, 2condition**true**

New available = available + Allocation(3, 3, 2) + (2, 0, 0) => 5, 3, 2

Similarly,weexamineanotherprocessP3.

Step3:

ForProcessP3:

P3Need<= Available

6, 0, 0 < = 5, 3, 2 condition is **false**.

Similarly,we examine another process, P4.

Step4:

ForProcessP4:

P4Need<= Available

0,1, 1<=5,3, 2conditionis **true**

NewAvailableresource=Available+Allocation5,3, 2 + 2, 1, 1 => 7, 4, 3

Similarly,weexamineanotherprocessP5.

Step5:

ForProcessP5:

P5Need<= Available

4,3, 1<=7,4, 3conditionis **true**

New available resource= Available+Allocation7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now,weagainexamineeachtypeofresourcerequestforprocesses P1 and P3.

your roots to success...

Step6:

ForProcessP1:

P1Need<= Available

7,4, 3<=7,4, 5conditionis **true**

NewAvailableResource=Available+Allocation7,4, 5 + 0, 1, 0 => 7, 5, 5

So,weexamineanotherprocessP2.

Step7:

ForProcessP3:

P3Need<= Available

6,0, 0<=7,5, 5conditionistrue

NewAvailableResource=Available+Allocation7,5, 5 + 3, 0, 2 => 10, 5, 7

Hence,weexecutethebanker'salgorithmtofindthesafestateandthesafe sequence like P2, P4, P5, P1 and P3.

Ans.3:

ForgrantingtheRequest (1,0,2),firstwehavetocheck that **Request <= Available**, that is (1, 0, 2)<= (3, 3,2), Since the condition is true, the process P2mayget therequestimmediately.

AllocationforP2is(3,0,2)andnewAvailableis (2, 3, 0)

Contextoftheneedmatrixisasfollows: Need [i] = Max [i] - Allocation [i]

NeedforP1: (7,5,3)-(0,1,0)=7, 4,3
NeedforP2:(3,2, 2)-(3, 0,2)=0, 2,0
NeedforP3: (9,0, 2)-(3,0,2)=6, 0,0
NeedforP4: (2,2, 2)-(2,1,1)=0, 1,1
NeedforP5: (4,3, 3)-(0,0,2)=4, 3,1

Process	Need		
	A	B	C
P1	7	4	3
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

ApplytheBanker's Algorithm:

AvailableResourcesofA,BandCare 2, 3,and0.

Nowwecheckifeachtypeofresourcerequestisavailableforeachprocess.

your roots to success...

Step1:

ForProcessP1:

Need<=Available

7, 4, 3 <= 2, 3, 0 condition is **false**.

So,we examine another process, P2.

Step2:

ForProcessP2:

Need<=Available

1,2, 2<=2, 3, 0condition**true**

New available = available + Allocation(2, 3, 0) + (3, 0, 2) => 5, 3, 2

Similarly,weexamineanotherprocessP3.

Step3:

ForProcessP3:

P3Need<= Available

6, 0, 0 < = 5, 3, 2 condition is **false**.

Similarly,we examine another process, P4.

Step4:

ForProcessP4:

P4Need<= Available

0,1, 1<=5,3, 2conditionis **true**

NewAvailableresource=Available+Allocation5,3, 2 + 2, 1, 1 => 7, 4, 3

Similarly,weexamineanotherprocessP5.

Step5:

ForProcessP5:

P5Need<= Available

4,3, 1<=7,4, 3conditionis **true**

Newavailableresource=Available+Allocation7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now,weagainexamineforprocessesP1andP3.

Step6:

ForProcessP1:

P1Need<= Available

7,4, 3<=7,4, 5conditionis **true**

NewAvailableResource=Available+Allocation7,4, 5 + 0, 1, 0 => 7, 5, 5

So,weexamineanotherprocessP2.

Step7:

ForProcessP3:

P3Need<= Available

6,0, 0<=7,5, 5conditionistrue

NewAvailableResource=Available+Allocation7,5, 5 + 3, 0, 2 => 10, 5, 7

Hence,P2grantedimmediatelyandthesafesequencelikeP2,P4,P5,P1and P3.

Ans.4:

Forgrantingthe Request (3,3, 0) byP5, firstwehavetocheck that **Request<= Available**, thatis(3,3, 0)<=(2, 3,0),

Sincetheconditionisfalse.Sotherequestfor(3,3,0)byprocessP5 cannotbe granted.

Ans.5:

Forgrantingthe Request (0,2, 0) byP1, firstwehavetocheck that **Request<= Available**, thatis(0,2, 0)<=(2, 3,0),

Sincetheconditionistrue.Sotherequestfor(0,2,0)byprocessP1maybe granted.

AllocationforP1is(0,3,0)

Contextoftheneedmatrixisasfollows: Need [i] = Max [i] - Allocation [i]

NeedforP1: (7,5,3)-(0,3, 0)=7, 2, 3

Process	**Need**		
	A	**B**	**C**
P1	7	2	3
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

ApplytheBanker's Algorithm:

AvailableResourcesofA,BandCare2,1, and0.

ForProcess P1:7, 2,3<=2,1,0condition is**false**.
ForProcess P2:0, 2,0<=2,1,0condition is**false.**
ForProcess P3:6, 0,0<=2,1,0condition is**false**.
ForProcessP4:0,1,1<=2,1,0condition is **false.**
ForProcessP5:4,3,1 <=2,1, 0condition is **false.**

Hence,thestateisunsafe,P1cannotbegrantedimmediately.

DEADLOCKDETECTION

Ifasystemdoesnotemployeitheradeadlockpreventionordeadlockavoidance algorithm then a deadlock situation may occur. In this case-

- Applyanalgorithmtoexaminethesystem‟sstatetodeterminewhetherdeadlock has occurred.
- Applyanalgorithm torecoverfromthe deadlock.

Adeadlockdetection algorithm is atechniqueused byan operatingsystem to identify deadlocks in the system. This algorithm checks the status of processes andresources to determine whether any deadlock has occurred and takes appropriate actions to recover from the deadlock.

Thealgorithmemploysseveraltimesvaryingdata structures:

Available–Avectoroflengthmindicatesthenumberofavailableresourcesof each type.

Allocation – An n*m matrix defines the number of resources of each type currentlyallocated to a process. The column represents resource and rows represent a process. **Request**–Ann*mmatrixindicatesthecurrentrequestofeachprocess.If request[i][j] equals k then process $P_i$is requesting k more instances of resource type R_j.

The Bankers algorithm includes a **Safety Algorithm / Deadlock Detection Algorithm.** The algorithm for finding out whether a system is in a safe state canbe described as follows:

Stepsof Algorithm:

1. Let *Work* and *Finish* be vectors of length m and n respectively. Initialize *Work= Available*. For *i=0, 1, …., n-1*, if $Request_i$= 0, then *Finish[i]* = true; otherwise, *Finish[i]*= false.
2. Findanindexisuchthatboth
   a) Finish[i]==false
   b) $Request_i$*<=Work*
   Ifnosuch*i*existsgotostep4.
3. *Work=Work+*$Allocation_i$*Fin ish[i]= true*
   GotoStep 2.
4. If *Finish[i]==false* forsomei,0<=i<n,thenthesystemisinadeadlockedstate. Moreover, if *Finish[i]==false* the process $P_i$is deadlocked.

Forexample,

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

1. Inthis, Work =[0,0, 0]&
Finish=[false,false,false,false,false]

2. *i=0*isselectedasbothFinish[0]=falseand[0,0,0]<=[0,0,0].3.
Work=[0, 0, 0]+[0,1,0]=>[0, 1, 0]&
Finish=[true,false,false,false,false].

4. *i=2*isselectedasbothFinish[2]=falseand[0,0,0]<=[0,1,0].5.
Work=[0, 1, 0]+[3,0,3]=>[3, 1, 3]&
Finish=[true,false,true,false,false].

6. *i=1*isselectedasbothFinish[1]=falseand[2,0,2]<=[3,1,3].7.
Work=[3, 1, 3]+[2,0,0]=>[5, 1, 3]&
Finish=[true,true,true,false,false].

8. *i=3*isselectedasbothFinish[3]=falseand[1,0,0]<=[5,1,3].9.
Work=[5, 1, 3]+[2,1,1]=>[7, 2, 4]&
Finish=[true,true,true,true,false].

10. *i=4*isselectedasbothFinish[4]=falseand[0,0,2]<=[7,2,4].11.
Work=[7, 2, 4]+[0,0,2]=>[7, 2, 6]&
Finish=[true,true,true,true,true].

12. SinceFinishisavectorofalltrueitmeans**thereisnodeadlock**inthis example.

your roots to success...

There areseveralalgorithmsfor detectingdeadlocksinanoperating system,including:

1. Wait-ForGraph:

A graphical representation of the system‟s processes and resources. A directed edgeis created from a process to a resource if the process is waiting for that resource. A cycle in the graph indicates a deadlock.

2. Banker'sAlgorithm:

A resource allocation algorithm that ensures that the system is always in a safe state, where deadlocks cannot occur.

3. ResourceAllocation Graph:

A graphical representation of processes and resources, where a directed edge from a processto a resourcemeansthatthe processis currentlyholdingthat resource. Deadlocks can be detected by looking for cycles in the graph.

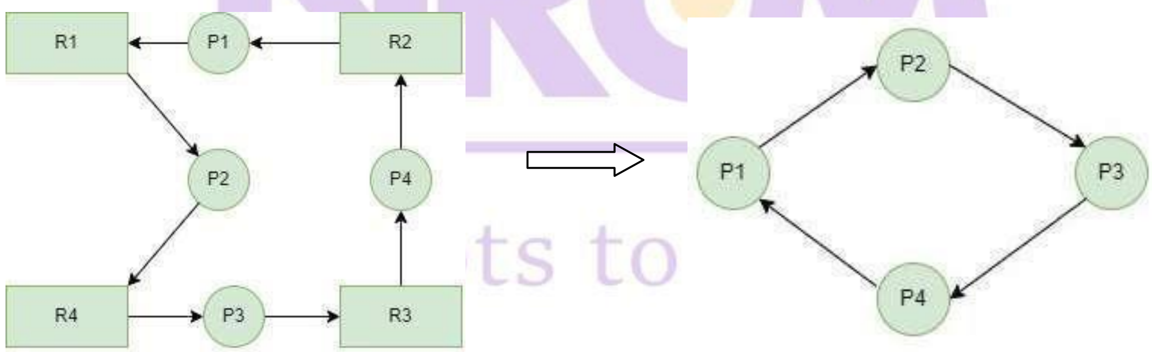
4. DetectionbySystemModeling:

A mathematical model of the system is created, and deadlocks can be detected by finding a state in the model where no process can continue to make progress.

5. Timestamping:

Each process is assigned a timestamp, and the system checks to see if any process is waiting for a resource that is held by a process with a lower timestamp.

These algorithms are used in different operating systems and systems with different resource allocation and synchronization requirements. The choice ofalgorithm depends on the specific requirements of the system and the trade-offs between performance, complexity and accuracy.



Resource Allocation Graph **Wait-For graph**

RECOVERYFROMDEADLOCK

The OS will use various recoverytechniques to restore the system if it encounters any deadlocks. When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover fromthat deadlock.

ApproachestoBreaking aDeadlock

(a) ProcessTermination

To eliminate the deadlock, we can simplykill one or more processes. For this, weuse two methods:

1. **AbortalltheDeadlocked Processes**:

   Abortingalltheprocesseswillcertainlybreakthe deadlockbut atagreat expense. The deadlocked processes may have been computed for a long time, and the result of those partial computations must be discarded and there is a probability ofrecalculating them later.

2. **Abortoneprocess atatimeuntil thedeadlock is eliminated**:

   Abort one deadlocked process at a time, until the deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because, after aborting each process, we have to run a deadlock detection algorithm to check whether any processes are still deadlocked.

(b) ResourcePreemption

To eliminate deadlocks using resource preemption, we preempt some resourcesfrom processes and give those resources to other processes. This method will raise three issues –

1. **Selectingavictim**:

   Wemust determinewhichresourcesand whichprocesses areto bepreemptedand also in order to minimize the cost.

2. **Rollback**:

   We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. Thatmeans aborting the processand restarting it.

3. **Starvation**:

   In a system, it may happen that the same process is always picked as avictim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that aprocess must be picked as a victim only a finite number of times.

UNIT–3

ProcessManagementandSynchronization- Thecriticalsectionproblems,Synchronizationhardware,Semaphore, and Classical problems of Synchronization, Critical region, Monitor.

Inter process communication Mechanism- IPC between process on a single computer system, IPC between process on different system,Using Pipes, FIFOs, Message Queue, Shared memory

SYNCHRONIZATION

Process Synchronization is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources inacontrolled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system.

Themainobjectiveofprocesssynchronizationistoensurethatmultiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access. To achieve this, various synchronization techniques such as semaphores, monitors and critical sections are used.

On the basis of synchronization, processes are categorized as one of the followingtwo types:

- **Independent Process**: The execution of one process does not affect the execution of other processes.
- **Cooperative Process**: A process that can affect or be affected by other processes executing in the system.

ProcesssynchronizationproblemarisesinthecaseofCooperativeprocesses also because resources are shared in Cooperative processes.

RaceCondition

A race condition is a condition when there are many processes and every process shares the data with each other and accessing the data concurrently and theoutput of execution depends on a particular sequence in which they share the data and access.

(OR)

When more than one process is executing the same code or accessing the same memoryor anyshared variable in that condition there is a possibilitythat theoutput or the value of the shared variable is wrong so for that all the processes doing the race to saythat my output is correct. This condition is known as **race condition**.

Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the accesstakes place.

Example:

Let‟s say there are two processes P1 and P2 which share common variable (shared=10), both processes are present in ready – queue and waiting for its turn tobe execute.

Process1	**Process2**
intX=shared	intY=shared
X++	Y--
sleep(1)	sleep(1)
shared= X	shared= Y

Suppose, Process P1 first come under execution,initializedasX=10andincrementitby 1 (ie.X=11), after then when CPU read line sleep(1), it switches from current process P1 to processP2 present in ready-queue. The processP1 goes in waiting state for 1 second.

Now CPU execute the Process P2, initialized Y=10 and decrement Yby 1(ie.Y=9), after then when CPU read sleep(1), the current process P2 goes in waiting state andCPU remains idle for sometime as there is no process in ready-queue.

After completion of 1 second of process P1 when it comes in ready-queue, CPU takes the process P1 under execution and execute the remaining line of code and shared=11.

After completion of 1 second of Process P2, when process P2 comes in ready-queue, CPU start executing the further remaining line of Process P2 and shared=9.

Note:

Weareassumingthefinalvalueofcommonvariable(shared)after execution of Process P1 and Process P2 is 10 (as Process P1 increment variable by1 and ProcessP2decrementvariableby1andfinallyitbecomesshared=10). But wearegetting undesired value due to lack of proper synchronization.

Actualmeaningofrace-condition

- If the order of execution of process (first P1 -> then P2) then we will get thevalue of common variable (shared) = 9.
- If the order of execution of process (first P2 -> then P1) then we will get thefinal value of common variable (shared) =11.

Basically, Here the (value1 = 9) and (value2=11) are racing , If we execute these two process in our computer system then sometime we will get 9 and sometime we will get 10 as final value of common variable(shared). This phenomenon is called **Race-Condition.**

CRITICALSECTIONPROBLEM

A critical section is acode segment that can be accessed byonlyone process at a time. The critical section contains shared variables that need to be synchronised to maintain the consistency of data variables. So the critical section problem means designing a way for cooperative processes to access sharedresources without creating data inconsistencies.

Intheentrysection, theprocessrequestsforentryinthe**Critical Section.** Anysolutiontothecriticalsection problemmustsatisfythreerequirements:

- **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executingintheirremaindersectioncanparticipateindecidingwhichwillenterin the critical section next, and the selection can"t be postponed indefinitely.
- **Bounded Waiting**: A bound must exist on the number of times that other processes are allowed to enter their critical sections after aprocess hasmade a request to enter its critical section and before that request is granted.

PETERSON'SSOLUTION

Peterson"sSolutionisaclassicalsoftware-basedsolutiontothecriticalsectionproblem. In Peterson"s solution, we have two shared variables:

- **boolean flag[i]**: Initialized to FALSE, initially no one is interested inentering the critical section
- **intturn**: Theprocesswhoseturnistoenterthecritical section.

```
//codeforproduceri
do
{
 flag[i]=true;turn
 =j;
 while(flag[j]==true&&turn==j);
          criticalsection
 flag[i]=false;
          remindersection
 }while(TRUE);
```

```
//codeforconsumerj
do
{
 flag[j]=true;turn
 =i;
 while(flag[i]==true&&turn==i);
          criticalsection
 flag[i]=false;
          remindersection
 }while(TRUE);
```

In the solution, i represents the Producer and j represents the Consumer. Initially, the flags are false. When a process wants to execute it‟s critical section,it sets its flag to true and turn into the index of the other process. This means that the process wants to execute but it will allowthe other process to run first. The process performs busy waiting until the other process has finishedit‟sown critical section. Afterthis, the current process enters its critical section and addsor removes a random number from the shared buffer.Aftercompletingthe critical section, it sets it‟s own flag to false, indicating it does not wish to execute anymore.

Peterson’sSolutionpreservesallthreeconditions:

- MutualExclusionisassuredasonlyoneprocesscanaccessthecriticalsection at any time.
- Progressisalsoassured,asaprocessoutsidethecriticalsectiondoesnotblock other processes from entering the critical section.
- BoundedWaitingispreservedaseveryprocessgets afairchance.

DisadvantagesofPeterson’sSolution

- Itinvolves busywaiting.
- Itislimitedto2processes.
- Peterson‟ssolutioncannotbeusedinmodernCPUarchitectures.

SynchronizationHardware

- ProblemsofCriticalSectionarealsosolvablebyhardware.
- Uniprocessor systems disables interrupts while a Process Pi isusing the CS but it is a great disadvantage in multiprocessorsystems

- Some systems provide a lock functionality where a Processacquires a lock while enteringthe CS and releases thelockafterleavingit. Thus another process tryingto enter CS cannotenteras the entry is locked. It can only do so if it is free byacquiringthelockitself

- Anotheradvancedapproachisthe***AtomicInstructions***(Non-Interruptibleinstructions).

MUTEXLOCKS

- Asthesynchronizationhardwaresolutionisnot easytoimplement from everyone, a strict software approachcalledMutex Locks was introduced. In this approach, intheentry section of code, a LOCK is acquired over thecriticalresourcesmodifiedand used insidecriticalsection,andintheexitsection thatLOCKisreleased. As the resource is locked while a process executes itscritical sectionhenceno otherprocess canaccess

SEMAPHORES

Semaphore is a Hardware Solution. This Hardware solution is written or given to critical section problem. The Semaphore is just a normal integer. The Semaphore cannot be negative. The least value for a Semaphore is zero (0). The Maximum value of a Semaphore can be anything. The Semaphores usually have two operations. The two operations have the capability to decide the values of the semaphores.

ThetwoSemaphoreOperationsare:

1. Wait()
2. Signal()

WaitSemaphoreOperation

The Wait operation works on the basis of Semaphore or Mutex Value.If the Semaphore value is greater than zero, then the Process can enter the Critical SectionArea.

IftheSemaphorevalueisequaltozerothentheProcesshasto wait.

IftheprocessexitstheCriticalSection,thenhavetoreducethevalueof Semaphore.

Definitionofwait()

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

SignalSemaphoreOperation

ThemostimportantpartisthatthisSignalOperationorVFunctionisexecuted only when the process comes out of the critical section. The value of semaphore cannotbe incremented before the exit of process from the critical section.

Definitionofsignal()

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

Therearetwotypesofsemaphores:

- **BinarySemaphores:**

  Theycan onlybeeither0 or1. Theyare also knownas mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of the mutex semaphore to 0 and some other process can enter its critical section.

- **Counting Semaphores:**

  They can have any value and are not restricted over acertain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the processhas an instance available. Then, the processcan enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of theresource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

CLASSICALPROBLEMSOFSYNCHRONIZATION

Thefollowingproblemsofsynchronizationareconsideredasclassicalproblems:

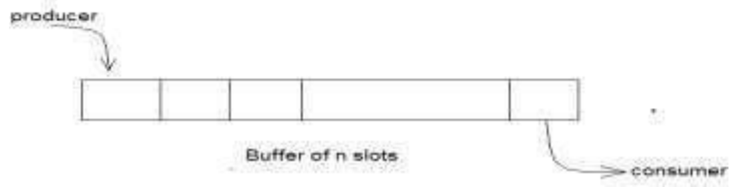
1. Bounded-buffer(orProducer-Consumer)Problem,

2. Dining-PhilosophersProblem,

3. ReadersandWritersProblem,

Bounded-buffer(orProducer-Consumer)Problem

Bounded Bufferproblem is also called **producerconsumerproblem** and itis one of the classic problems of synchronization. This problem is generalized in termsofthe



Producer-Consumer problem. Solution to this problem is, creatingtwocounting semaphores “full” and “empty” to keep track of the current number of full and empty buffers respectively. Producers produce a productand consumers consume the product, but both use of one of the containers each time.

Aproducertriestoinsertdatainto anemptyslotofthebuffer.Aconsumer triesto removedatafromafilled slotin thebuffer.Thereneedstobeawayto maketheproducer and consumer work in an independent manner.

One solution of this problem is to use semaphores. The semaphores which will be used here are:

- **m**,a**binarysemaphore**whichisusedtoacquireandreleasethelock.
- **empty**,a**countingsemaphore**whoseinitialvalueisthenumberofslots in the buffer, since, initially all slots are empty.
- **full**,a**countingsemaphore**whoseinitialvalueis0.

At anyinstant, the current value of empty represents the number of empty slotsin the buffer and full represents the number of occupied slots in the buffer.

TheProducerOperation

```
do
{
   wait(empty);
                          //waituntilempty>0andthendecrement'empty'wait(mute
   x);                    // acquire lock

   /*performtheinsertoperationinaslot*/

   signal(mutex);         // release lock
   signal(full);          //increment'full'

 }while(TRUE);
```

- Looking at the above code for a producer, we can see that a producer firstwaitsuntil there is atleast one empty slot.
- Thenitdecrementsthe**empty**semaphorebecause,therewillnowbeoneless empty slot, since the producer is going to insert data in one of those slots.

- Then,itacquireslockonthebuffer,sothattheconsumercannotaccessthebuffer untilproducercompletesitsoperation.
- Afterperformingtheinsertoperation,thelockisreleasedandthevalueof **full**is incrementedbecausetheproducer has just filled aslot in the buffer.

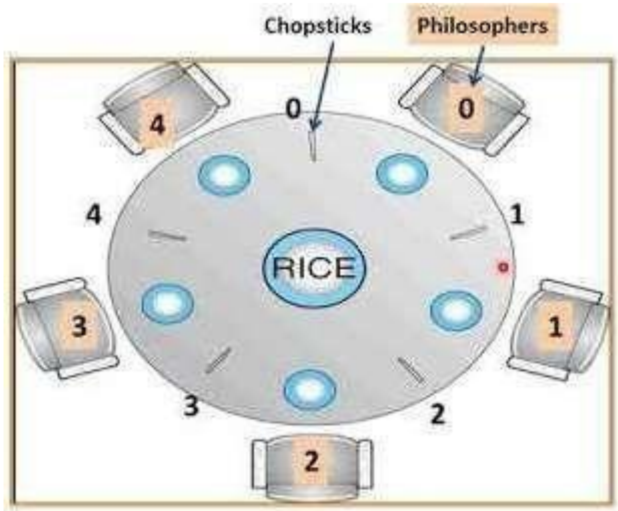
TheConsumerOperation

```
do
{
   wait(full);
                              //waituntilfull>0andthendecrement'full'wait(mut
   ex);                       // acquire the lock

   /*performtheremoveoperationinaslot*/

   signal(mutex);             //releasethelock
   signal(empty);             // increment 'empty'

 }while(TRUE);
```

- Theconsumerwaitsuntilthereisatleast onefullslotinthebuffer.
- Thenitdecrementsthe **full** semaphorebecausethenumberofoccupiedslotswill be decreased by one, after the consumer completes its operation.
- Afterthat,theconsumeracquireslockonthebuffer.
- Followingthat,theconsumercompletestheremovaloperationsothatthedata from one of the full slots is removed.
- Then,theconsumer releasesthelock.
- Finally,the **empty** semaphoreisincrementedby1,becausetheconsumerhas just removed data from an occupied slot, thus making it empty.

Dining-PhilosophersProblem

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticksadjacenttohim.Onechopstickmaybepickedup byanyoneofitsadjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.



The design of the problem was to illustrate the challenges ofavoiding deadlock, a deadlock state of a system is a state in which no progress of system is possible. Considera proposal where each philosopher is instructed to behave as follows:

- The philosopher is instructed to think till the left fork is available,whenit isavailable, hold it.
- Thephilosopherisinstructedtothinktilltherightforkisavailable,when it is available, hold it.
- Thephilosopheris instructedtoeatwhenboth forksare available.
- then,puttherightforkdownfirst
- then,puttheleftforkdownnext
- repeatfromthebeginning.

Thestructureof Philosopher i is asfollows.do

```
{
 Wait(take_chopstick[i]);
 Wait(take_chopstick[(i+1)%5]);
 …
 EAT
 …
 Signal(put_chopstick[i]);
 Signal(put_chopstick[(i+1)%5]);
 …
 THINK
}while(TRUE);
```

Intheabovecode,firstwaitoperationisperformedontake_chopstick[i]and take_chopstick[(i+1)%5].Thisshowsphilosopherihavepickedupthechopsticks fromitsleftandright. Theeatingfunctionisperformedafter that.

On completion of eating by philosopher i the, signal operation is performed on take_chopstick[i] and take_chopstick[(i+1) % 5]. This shows that the philosopher ihave eaten and put down both the left and right chopsticks. Finally,the philosopher starts thinking again.

Let value of i = 0(initial value), Suppose Philosopher P_0wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstick[i]);** by doing this itholds **C0 chopstick** and reduces semaphore C0 to 0**,** after that it execute **Wait(take_chopstick[(i+1) % 5]);** by doing this it holds **C1 chopstick** (since i =0, therefore(0 + 1) % 5 = 1) and reduces semaphore C1 to 0.

Similarly, suppose now Philosopher P1 wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstick[i]);** by doing this it will try to hold **C1 chopstick** but will not be able to do that**,** since the value of semaphore C1 has already been set to 0 by philosopher P0, therefore it will enter into an infinite loop because of which philosopher P1 will not be able to pick chopstick C1 whereas if Philosopher P2 wants to eat, it will enter in Philosopher() function, and execute **Wait(take_chopstickC[i]);** by doing this it holds **C2 chopstick** and reduces semaphore C2 to 0, after that, it executes **Wait(take_chopstickC[(i+1) % 5]);** by doing this it holds **C3 chopstick**(since i =2, therefore (2 + 1) % 5 = 3) and reduces semaphore C3 to 0.

Hence the above code is providing a solution to the dining philosopher problem, A philosopher can only eat if both immediate left and right chopsticks ofthe philosopherareavailableelsephilosopherneedstowait.Alsoatonegotwo independentphilosophers can eat simultaneously (i.e., philosopher **P0 and P2, P1 and P3 & P2 and P4** can eat simultaneously as all are the independent processes and they are following the above constraint of dining philosopher problem)

Thedrawbackoftheabovesolutionofthediningphilosopher problem

- Notwoneighbouringphilosopherscaneatatthesamepointin time.
- This solution can lead to a deadlock condition. This situation happens if all the philosophers pick their left chopstick at the same time, which leads to the condition of deadlock and none of the philosophers can eat.

Toavoiddeadlock,someofthesolutionsareasfollows:

- Maximum number of philosophers on the table should not be more than four, in this case, chopstick C4 will be available for philosopher P3, so P3 willstarteatingandafterthefinishofhiseatingprocedure,hewillputdown his both the chopstick C3 and C4, i.e. semaphore C3 and C4 will nowbe incremented to 1. NowphilosopherP2 whichwas holding chopstick C2 willalsohave chopstickC3 available, hence similarly, he will put down hischopstick after eating and enable other philosophers to eat.
- Aphilosopheratanevenpositionshouldpick the rightchopstickandthentheleft

chopstickwhile aphilosopherat an odd position should picktheleftchopstick and then the right chopstick.

- Only incaseifboththechopsticks(leftandright)areavailableatthesame time, only then a philosopher should be allowed to pick their chopsticks
- All the four starting philosophers (P0, P1, P2, and P3) should pick the left chopstick and then the right chopstick, whereas the last philosopher P4 should pick the right chopstick and then the left chopstick. This will force P4 to hold his right chopstickfirst since theright chopstickofP4 is C0, which is alreadyheld by philosopher P0 and its value is set to 0, i.e C0 is already 0, because of which P4 will get trapped into an infinite loop and chopstick C4 remains vacant. Hence philosopher P3 has both left C3 and right C4 chopstick available, therefore it will start eating and will put down its both chopsticks once finishes and let others eat which removes theproblem of deadlock.

ReadersandWritersProblem

Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processesbyreferringto theformer as readers and to thelatter as writers.Preciselyin OS we call this situation as the readers-writers problem. Problem parameters:

- Onesetofdataissharedamonganumber ofprocesses.
- Once a writer is ready, it performs its write. Onlyone writer maywrite at atime.
- If a process is writing, no other process can read it.
- If atleastonereaderis reading,nootherprocesscan write.
- Readersmaynotwriteand onlyread.

Therearefourtypesofcasesthatcouldhappenhere.

Case	**Process1**	**Process2**	**Allowed/NotAllowed**
Case1	Writing	Writing	NotAllowed
Case2	Writing	Reading	NotAllowed
Case3	Reading	Writing	NotAllowed
Case4	Reading	Reading	Allowed

Threevariablesareused:**mutex,wrt,readcnt**

1. Semaphore**mutex**isusedtoensuremutualexclusionwhen**readcnt**isupdated i.e.whenanyreaderentersorexitfromthecriticalsection.

2. Semaphore**wrt**isusedbybothreadersandwriters.
3. **readcnt**tellsthenumberofprocessesperformingreadinthecriticalsection, initially 0 amd it is integer variable.

Functionsforsemaphore

wait():decrementsthesemaphorevalue.

signal():incrementsthesemaphorevalue.

Readerprocess

- Reader requests the entryto critical section.
- If allowed:
  - it increments the count of number of readers inside the critical section.If thisreaderisthefirstreaderentering,itlocks the **wrt**semaphore to restrict the entry of writers if any reader is inside.
  - It then, signals mutex as any other reader is allowed to enter while others are already reading.
  - After performing reading, it exits the critical section. When exiting,it checks if no more reader is inside, it signals the semaphore "wrt"as now, writer can enter the critical section.
- Ifnotallowed,itkeepsonwaiting.

```
do
{
   wait(mutex);         // Reader wants to enter the critical section
   readcnt++;           // The number of readers has now increased by 1

   if (readcnt==1)      //thereisatleastonereaderinthecriticalsection  wait(wrt);
                        // no writer can enter if there is even one reader

   signal(mutex);       //otherreaderscanenterwhereothererisinside

    .....   performREADING

   wait(mutex);         //areaderwantstoleave
   readcnt--;

if (readcnt == 0)       //noreaderisleftinthecriticalsection,
      signal(wrt);      // writers can enter
```

```
    signal(mutex);          //readerleaves

}while(true);
```

Writerprocess

1. Writerrequeststheentrytocriticalsection.
2. Ifallowedi.e.wait()givesatruevalue,itentersandperformsthe write. If not allowed, it keeps on waiting.
3. Itexitsthecriticalsection.

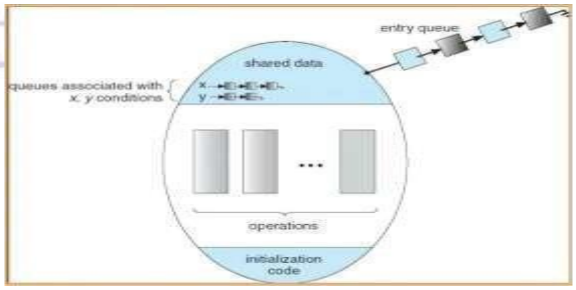
```
do
{
   wait(wrt);                          //writerrequestsforcriticalsection

        …performWRITING

   signal(wrt);                        //leavesthecriticalsection

}while(true);
```

Thus, the semaphore „**wrt**„ is queued on both readers and writers in a mannersuchthatpreferenceisgiventoreadersifwritersarealsothere.Thus,no readeriswaiting simply because a writer has requested to enterthecritical section.

MONITOR

It is a synchronization technique that enables threads to mutual exclusion and the **wait()** for a given condition to become true. It is an abstract data type. It hasashared variableandacollectionofproceduresexecutingonthesharedvariable. A process may not directly access the shared data variables, and procedures are required to allow several processes to access the shared data variables simultaneously.

At any particular time, only one process may be active in a monitor. Other processes that require access to the shared variables must queue and are only granted access after the previous process releases the shared variables.



Syntax:

```
monitor
{
  //sharedvariabledeclarations
```

```
    data variables;
    ProcedureP1(){...}
    ProcedureP2(){...}
    .
    .
    .
    Procedure    Pn()    {    ...    }
    Initialization Code() { ... }
}
```

Advantages

- Mutualexclusionisautomaticin monitors.
- Monitorsarelessdifficulttoimplementthansemaphores.
- Monitors may overcome the timing errors that occur when semaphores areused.
- Monitorsareacollectionofproceduresandconditionvariablesthatarecombined in a special type of module.

Disadvantages

- Monitorsmustbeimplementedintotheprogramminglanguage. The
- compiler should generate code for them.
- It gives the compiler the additional burden of knowing what operating system features is available for controlling access to crucial sections in concurrent processes.

your roots to success...

ComparisonbetweentheSemaphoreand Monitor

Features	Semaphore	Monitor
Definition	A semaphore is an integer variablethat allows many processes in a parallel system to manage access toa common resource like amultitasking OS.	Itisasynchronizationprocess that enables threads to have mutual exclusion and the wait() for a given condition to becometrue.

Syntax	// Wait Operation
	wait(Semaphore S)
	{
	while(S<=0);
	S--;
	}
	// Signal Operation
	signal(SemaphoreS)
	{
	S++;
	}
		Monitor
		{
		//sharedvariabledeclarations
		
		ProcedureP1(){...}
		ProcedureP2(){...}
		.
		.
		.
		ProcedurePn(){...}
		InitializationCode(){...}
		}

Basic	Integervariable	Abstractdata type
Access	When a process uses sharedresources, it calls the wait() method on S, and when it releases them, it uses the signal() method on S.	When a process uses shared resources in the monitor, it has to access them via procedures.
Action	The semaphore's value shows the number of shared resources available in the system.	The Monitor type includesshared variables as well as a set of procedures that operate on them.
Condition Variable	Nocondition variables.	Ithascondition variables.

your roots to success...

Whatis InterProcessCommunication

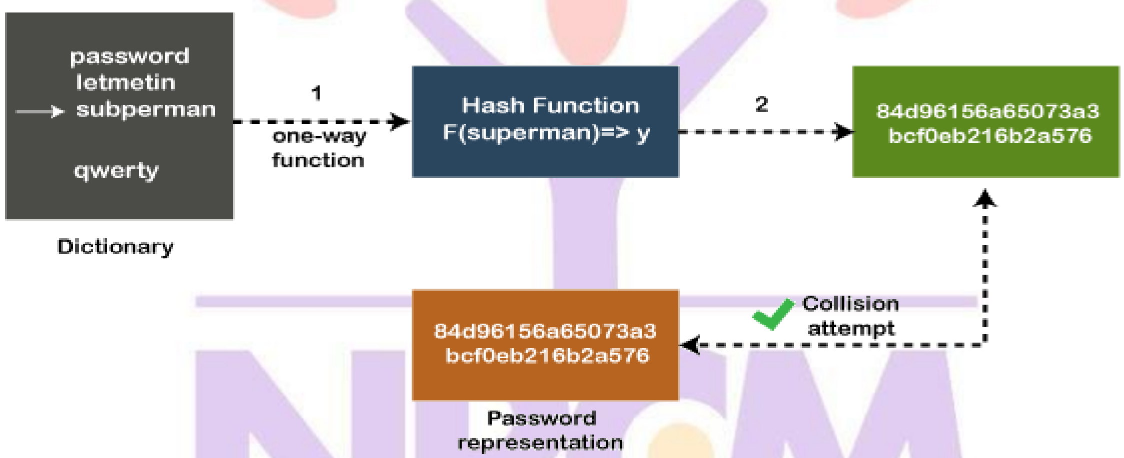
In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

Let us now look at the general definition of inter-process communication, which will explain the same thing that we have discussed above.

Definition

"Inter-process communication is used for exchanging useful information between numerous threads in one or more processes (or programs)."

Tounderstandinterprocesscommunication,youcanconsiderthefollowinggivendiagramthat illustrates the importance of inter-process communication:



RoleofSynchronizationinInterProcess Communication

It is one of the essential parts of inter process communication. Typically, this is provided by inter process communicationcontrolmechanisms,butsometimesitcan alsobecontrolledbycommunication processes.

Thesearethe followingmethods that usedto providethe synchronization:

1. **MutualExclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

MutualExclusion:-

Itisgenerallyrequiredthatonlyoneprocessthreadcanenterthecriticalsectionatatime.Thisalso helps in synchronization and creates a stable state to avoid the race condition.

Semaphore:-

Semaphoreisatypeofvariablethatusuallycontrolstheaccesstothesharedresourcesbyseveral processes. Semaphore is further divided into two types which are as follows:

1. BinarySemaphore
2. CountingSemaphore

Barrier:-

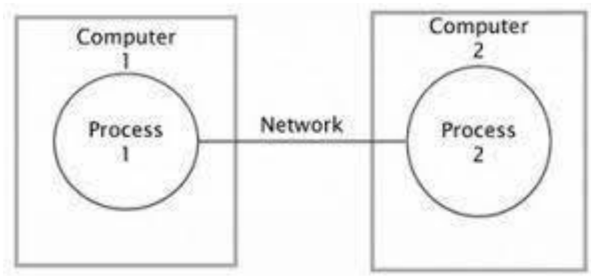
Abarriertypicallynot allowsan individual process to proceed unless all theprocesses does not reach it. It is used by many parallel languages, and collective routines impose barriers.

Spinlock:-

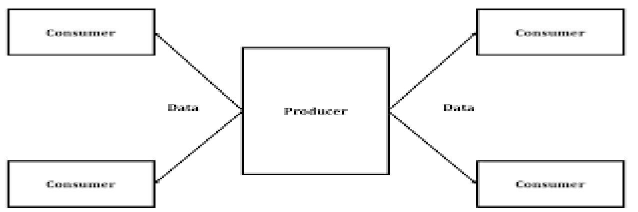
Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or staysin aloop whilecheckingthatthelockisavailableornot. It isknownasbusywaitingbecauseeven though the process active, the process does not perform any functional operation (or task).

IPCbetweenprocessesonasinglecomputersystem:-

IPC refers to the mechanisms and techniques that operating systems use to facilitate communication between different processes. In a multitasking environment, numerous processes are running concurrently, and IPC serves as the bridge that allows them to exchange information and coordinate their actions.



IPCbetweenprocessesondifferent system:-



Approachesto Interprocess Communication

Wewillnowdiscuss somedifferentapproaches tointer-processcommunication whichareasfollows:

Thesearea fewdifferentapproachesforInter-ProcessCommunication:

1. **Pipes**
2. **Shared Memory**
3. **Message Queue**
4. **Direct Communication**
5. **Indirectcommunication**
6. **MessagePassing**
7. **FIFO**

Tounderstandthem inmoredetail,we will discusseach ofthem individually.

Pipe:-

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

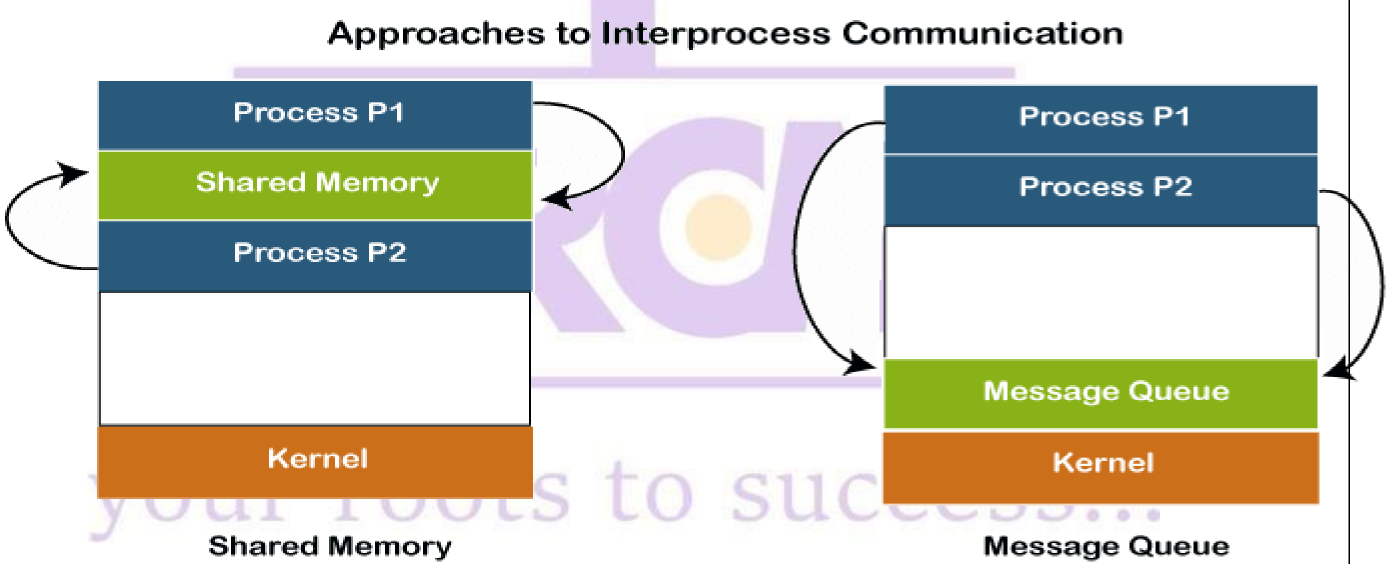
Shared Memory:-

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

Message Queue:-

In general, several different messages are allowed to read and write the data to the message queue. In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them. In short, we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

To understand the concept of Message queue and Shared memory in more detail, let's take a look at its diagram given below:



MessagePassing:-

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the hared variables.

Usually,theinter-processcommunicationmechanismprovidestwooperationsthatareasfollows:

- send(message)
- received(message)

Direct Communication:-

Inthistypeofcommunicationprocess,usually,alinkiscreatedorestablishedbetweentwo communicating processes. However, in everypair of communicating processes, onlyone link can exist.

IndirectCommunication

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

FIFO:-

Itisatypeofgeneralcommunicationbetweentwounrelatedprocesses.Itcanalsobeconsideredas full-duplex, which means that one process can communicate with another process and vice versa.

Someotherdifferentapproaches

- **Socket:-**

It acts as a type of endpoint for receiving or sending the data in a network. It is correct for data sent between processes on the same computer or data sent between different computers on the samenetwork. Hence, it used by several types of operating systems.

- **File:-**

file server. Another most important thing is that several processes can access that file as required or needed.

- **Signal:-**

As its name implies, they are a type of signal used in inter process communication in a minimal way. Typically, they are the massages of systems that are sent by one process to another. Therefore, they are not used for sending data but for remote commands between multiple processes.

Usually,theyarenot usedto sendthe databutto remotecommands inbetween several processes.

Whyweneed interprocesscommunication?

There are numerous reasons to use inter-process communication for sharing the data. Here are some of the most important reasons that are given below:

- Ithelpstospeedupmodularity
- Computational
- Privilegeseparation
- Convenience
- Helpsoperatingsystemto communicatewitheach otherandsynchronizetheiractionsas well.

UNIT-4

Memory Management and virtual memory-Logical versus physical address space, Swapping, Contiguousallocations,Paging,segmentation,segmentationwithpaging,Demandpaging,Page replacement, Page Replacement algorithms.

Memory Management:-Memory is central to the operation of a modern computer system. Memory consists ofa large array of bytes, each with its own address.

A typical instruction-execution cycle, for example, first fetches an instruction from memory. The instruction is then decoded and may cause operands to be fetchedfrom memory. After the instruction has been executed on the operands, results may be stored back in memory.

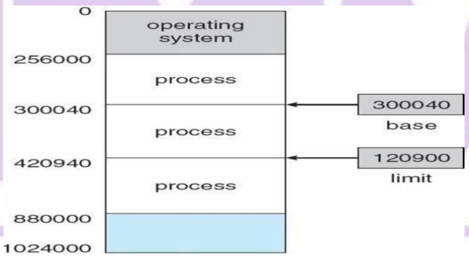
1. BasicHardware

Main memory and the registers built into the processor itself are the only general-purpose storage that the CPU can access directly. Therefore, any instructions in execution,and any data being used by the instructions, must be in one of these direct-access storage devices. If the data are not in memory, theymust be moved there before the CPU can operate onthem.

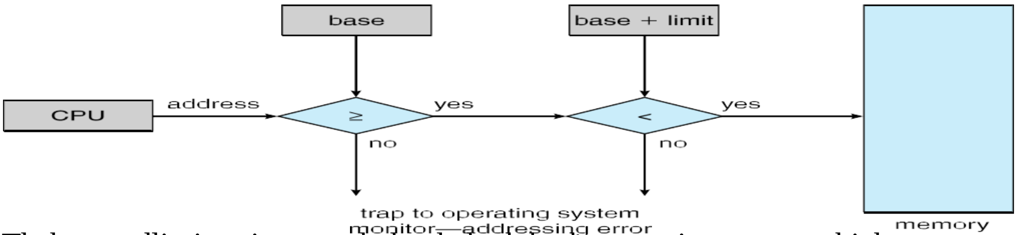
Protectinguserprocessesfromoneanother:

We first need to make sure that each process has a separate memory space. Separate per-process memory space protects the processes from each other and is fundamental to having multiple processes loaded in memory for concurrent execution. To separate memory spaces, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection byusing two registers, usually a base and a limit

The **base register** holds the smallest legal physical memory address; the**limit register**specifies thesizeoftherange.Forexample,ifthebaseregisterholds 300040 and the limit register is 120900, and then the program can legally access all addresses from 300040 through 420939 (inclusive).



Any attempt by a program executing in user mode toaccessoperating-system memoryor other users' memoryresults in a trap to the operating system.



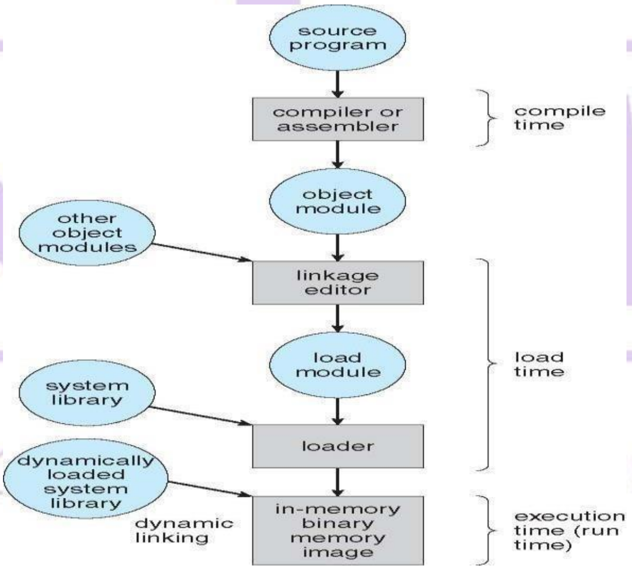
Thebaseandlimitregisterscanbeloadedonlybytheoperatingsystem,whichusesa specialprivilegedinstruction.Sinceprivilegedinstructionscanbeexecutedonlyinkernel mode,andsinceonlytheoperatingsystemexecutesinkernelmode,onlytheoperating system can load the base and limit registers.

2. AddressBinding

Usually, a program resides on a disk as a binary executable file. To be executed, the program must be brought into memory and placed within a process. Depending on the memory management in use, the process may be moved between disk and memory during its execution. The processes on the disk that are waiting to be brought into memoryforexecution form the **input queue**.

Classically, the binding of instructions and data to memory addresses can be done at any step along the way:

- **Compile time**. If you know at compile time where the process will reside in memory, then **absolute code** can be generated. For example, if you know that a user process will reside starting at location *R*, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time.
- **Load time**. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**. In this case, final binding is delayed until load time. If the starting addresses changes, we need only reload the user code to incorporate this changed value.
- **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Most general-purpose operating systems use this method.



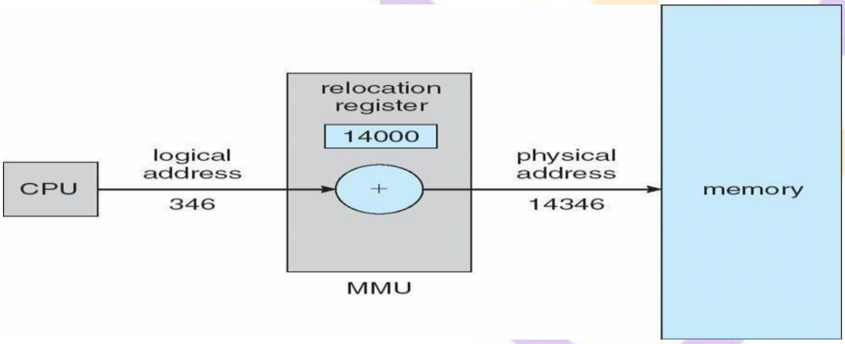
3. LogicalVersusPhysicalAddressSpace

- An address generated by the CPUis commonly referred to as a**logical address or virtual address**.
- Anaddressseenbythememoryunit—thatis,theoneloadedintothe**memory- address register** of the memory—is commonlyreferred to as a **physical address**.
- Thesetofalllogicaladdressesgenerated byaprogramisa**logicaladdress space**.

- Thesetofallphysicaladdressescorrespondingtotheselogicaladdressesisa **physicaladdress space.**

Memory-ManagementUnit (MMU)

- The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU)**.
- The base register is now called a **relocation register**. The value in the relocation register is added to every address generated bya user process at the time the address is sent to memory.
- For example,if the baseis at14000, then an attemptbythe user to addresslocation 0 is dynamically relocated to location14000; an access to location346 is mapped to location 14346.



WhatisSwapping?

A process must be in memory to be executed. A process, however, can be **swapped** temporarily out of memory to a **backing store** and then brought back into memory for continued execution. Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system, thus increasing the degree of multiprogramming in a system.

1. **StandardSwapping**

Standard swapping involves moving processes between main memory and a backing store. The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

Ready Queue: The system maintains a **ready queue** consisting of all processes whose memory images are on the backing storeor in memoryand are readyto run.

Dispatcher: Whenever the CPU scheduler decides to execute a process, it callsthe dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if thereis no free memoryregion, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.

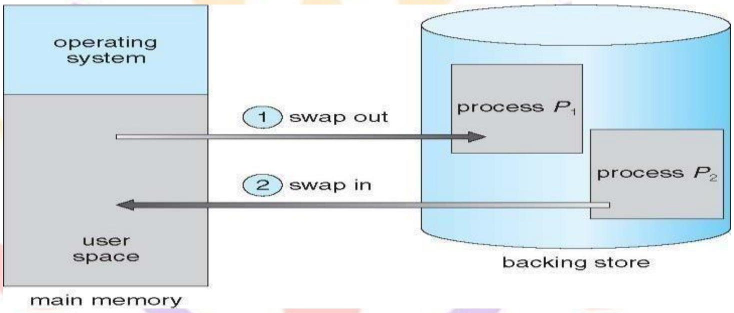
Factors

- Thecontext-switchtimeinsuchaswappingsystemis fairlyhigh.
- Thetotaltransfer timeis directlyproportionaltotheamount ofmemoryswapped.

- Ifwewanttoswapaprocess,wemustbesurethatitiscompletelyidle.

Standardswappinginmodernoperatingsystems

- Standard swapping is not used in modern operating systems. It requires too much swapping time and provides too little execution time to be a reasonable memory-management solution.
- Modified versions of swapping, however, are found on many systems, including UNIX, Linux, and Windows.
- In one common variation, swapping is normally disabled but will start if the amount of free memoryfalls below a threshold amount. Swapping is halted when the amount of free memory increases.
- Another variation involves swapping portions of processes—rather than entire processes—to decrease swap time.



2. SwappingonMobile Systems

Mobilesystemstypicallydo notsupportswappinginanyform.

Reasons

- Mobile devices generally use flash memory rather than hard disks. The resulting spaceconstraints avoid swapping.
- Thelimitednumberofwritesthatflashmemorycantoleratebeforeitbecomesunreliable
- Thepoorthroughputbetweenmainmemoryandflashmemoryin thesedevices.

MechanismsinsteadofSwapping

- **Apple's iOS** *asks* applications to voluntarily relinquish allocated memory. Any applications that fail to free up sufficient memory may be terminated by the operating system.
- **Android** does not support swapping and adopts a strategy similar to that used by iOS. It may terminate a process if insufficient free memory is available. However, before terminating a process, Android writes its **application state** to flash memoryso that it can be quickly restarted.

your roots to success...

ContiguousMemoryAllocation

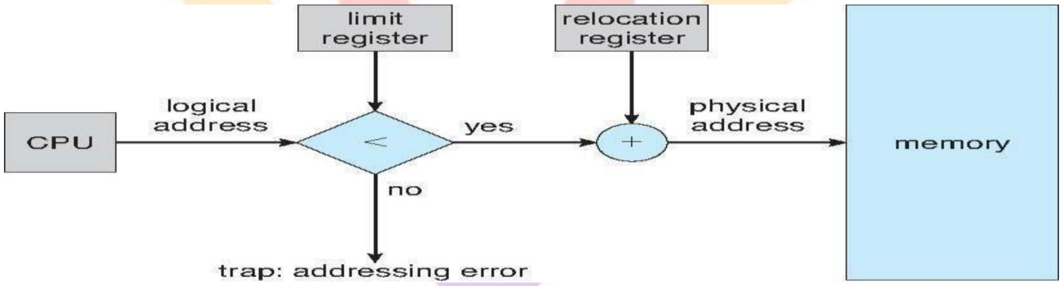
We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In **contiguous memory allocation**, each process is contained in a singlesection ofmemorythat is contiguous to thesection containing the next process.

1. **MemoryProtection**

We can prevent a process from accessing memory it does not own by combining two ideas. If we have a system with a relocation register, together with a limit register, we accomplish our goal.

Process

- The relocation register contains the value of the smallest physical address; the limit register contains the range of logical addresses (for example, relocation = 100040 and limit = 74600).
- Eachlogicaladdressmustfallwithintherangespecifiedbythelimitregister.
- The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory.
- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- Because every address generated by a CPU is checked against these registers, we can protect both the operating system and the other users programs and data from being modified by this running process.



2. **Memoryallocationmethodsformemoryallocation**

   a. **Fixed-Sized Partitions**

   - Oneofthesimplestmethodsforallocatingmemoryistodividememoryinto several fixed-sized **partitions**.
   - Eachpartitionmaycontainexactlyone process.
   - In this **multiple partition method**, when a partition is free, a process is selected from the input queue and is loaded into the free partition.
   - Whentheprocessterminates,thepartitionbecomesavailableforanother process.
   - Thismethodwasoriginally usedbytheIBMOS/360operatingsystem(called MFT) but is no longer in use.

   b. **VariableSized -Partition**

   - In the **variable-partition** scheme, the operating system keeps a table indicatingwhich parts of memory are available and which are occupied.
   - Initially, all memory isavailablefor user processes andis considered one large block of available memory, a **hole**.
   - When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process.

- If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- If the new hole is adjacent to other holes, these adjacent holes aremerged to form one larger hole.
- At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.

DynamicStorageAllocationProblem(MemoryAllocation Techniques)

This concerns how to satisfy a request of size *n* from a list of free holes. There are many solutions to this problem. The **first-fit**, **best-fit**, and **worst-fit** strategies are the ones most commonlyused to select a free hole from the set of available holes.

- **First fit**. Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- **Best fit**. Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered bysize. This strategyproduces the smallest leftover hole.
- **Worst fit**. Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may bemore useful than the smaller leftover hole from a best-fit approach.

Comparison:

- FirstfitandBestfitarebetterthanWorstfitintermsofdecreasingtimeandstorage utilization.
- Neither first fit nor Best fit is clearlybetter than the other in terms ofstorage utilization, but First fit is generally faster.

3. Fragmentation

Memoryfragmentationcanbeinternalaswellasexternal.

a. InternalFragmentation

- The overhead to keep track of this hole will be substantially larger than the hole itself. The general approach to avoiding this problem is to break the physical memoryinto fixed-sized blocksandallocate memoryinunitsbasedonblock size.
- With this approach, the memoryallocated to a process maybe slightlylarger than the requested memory.
- The difference between these two numbers is **internal fragmentation**—unused memory that is internal to a partition.

b. ExternalFragmentation

- Both the first-fit and best-fit strategies for memory allocation sufferfrom **externalfragmentation**.Asprocessesareloaded andremovedfrommemory,the free memory space is broken into little pieces.

- External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

50-percent rule: Depending on the total amount of memory storage and the average process size, external fragmentation may be a minor or a major problem. Statistical analysis of firstfit, for instance, reveals that, even with some optimization, given *N* allocated blocks, another 0.5 *N* blocks will be lost to fragmentation. That is, one-third of memory may be unusable! This property is known as the **50-percent rule**.

SolutiontoExternalFragmentation

a. **Compaction**
    - Thegoal is to shufflethememorycontents so as to placeall freememorytogetherin one large block.
    - Compaction is not always possible, however. If relocation is static and is done at assemblyor load time, compaction cannot be done. It is possible onlyif relocation is dynamic and is done at execution time.
    - The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive.

b. **Noncontiguouslogicaladdressspace**
    - This permits the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever such memory is available.
    - Twocomplementarytechniquesachievethissolution:segmentationandpaging.

Segmentation

Dealing with memory in terms of its physical properties is inconvenient to both the operating system and the programmer. What if the hardware could provide a memory mechanism that mapped the programmer's view to the actual physical memory? The system would have more freedom to manage memory, while the programmer would have a more natural programming environment. Segmentation provides such a mechanism.

1. BasicMethod

Segmentation is a memory-management scheme that supports the programmer view of memory. A logical address space is a collection of variable sized segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The programmer therefore specifies each address by two quantities: a segment name and an offset.

segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a *two tuple:*

<segment-number,offset>.

ExampleofSegments

When a program is compiled, the compiler automatically constructs segments reflecting the input program.

ACcompilermightcreateseparatesegmentsforthefollowing:

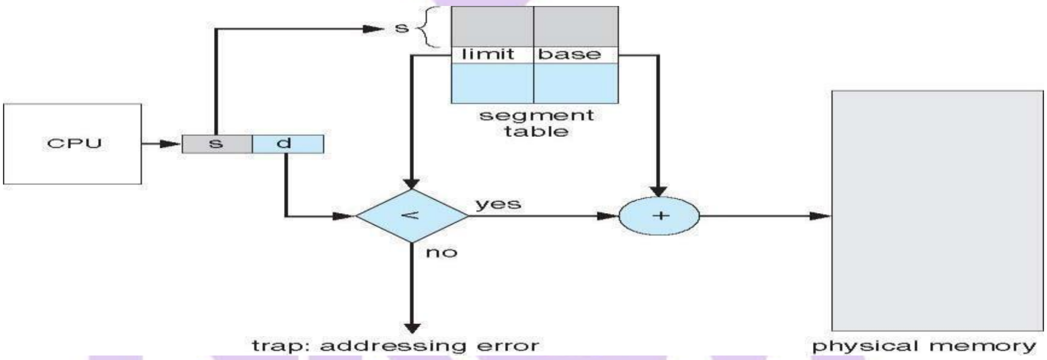
1. Thecode
2. Globalvariables
3. Theheap,fromwhichmemoryis allocated
4. Thestacks used byeachthread
5. ThestandardC library\

2. SegmentationHardware

Although the programmer can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one dimensional sequence of bytes. Thus, we must define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by a **segment table**. Each entry in the segment table has a **segment base** and a **segment limit**.

- **Segment base:** The segment base contains the starting physical address where the segment resides in memory.
- **Segmentlimit**:Thesegmentlimitspecifiesthelengthofthesegment.

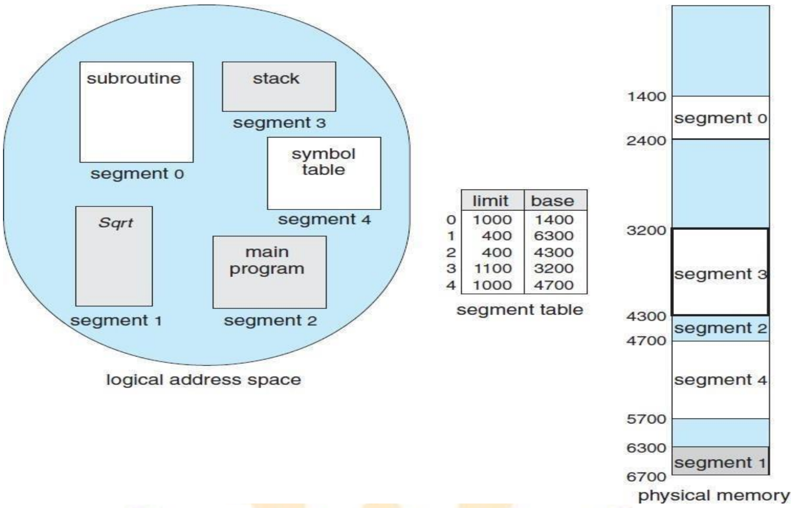
A logical address consists of two parts: a segment number, *s*, and an offset into that segment, *d*. The segment number are used as an index to the segment table. The offset*d* of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offsetis legal, itis added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an arrayof base–limit register pairs.



Example:

We have five segments numbered from 0 through 4. The segments are stored in physical memory. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the lengthof that segment (or limit).

Consider, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segmentis only 1,000 bytes long.



Paging

Pagingisanothermemory-managementschemethatoffersphysicaladdressspaceof a process to be non-contiguous. Paging also avoids external fragmentation and the need for compaction, whereas segmentation does not. Because of its advantages, paging in its various forms is used in most operating systems, from mainframes to smart phones.

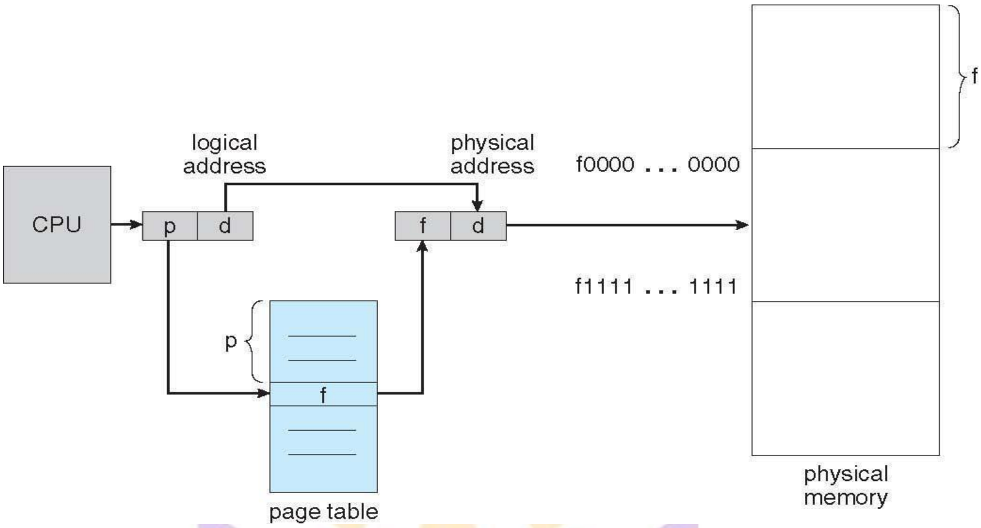
1. **BasicMethod**
   - **Frames**:Paginginvolvesbreakingphysicalmemoryintofixed-sizedblockscalled **frames.**
   - **Pages:**Breakinglogicalmemoryintoblocksofthesamesizecalled**pages**.

When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store).

HardwareSupport forPaging

EveryaddressgeneratedbytheCPUisdividedintotwoparts:a**pagenumber(p)** and a**pageoffset(d)**.

- **Page Table:** The page number is used as an index into a **page table**. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memoryunit.
- **Frame Table:** Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory—which frames are allocated, which frames are available, how many total frames there are, and so on? This information is generally kept in a data structure called a **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 which page of which process or processes.



DefiningofPageSize

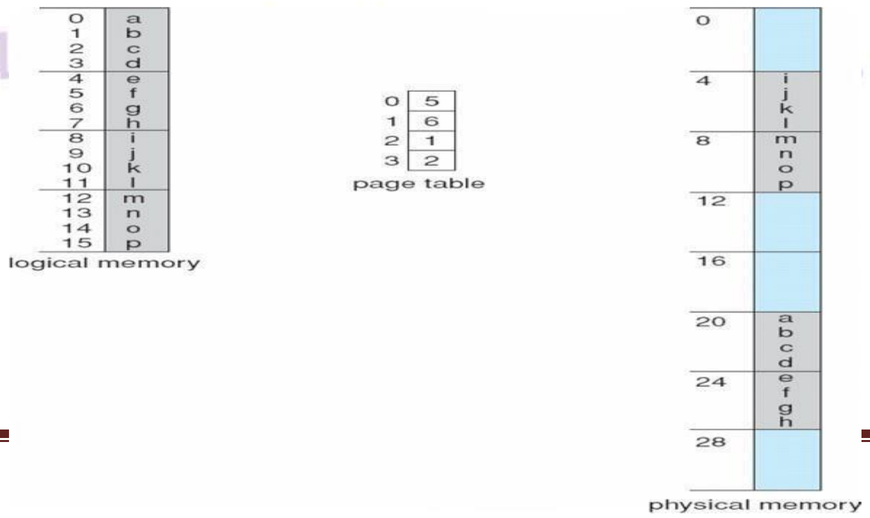
The page size (like the frame size) is defined by the hardware. The size of a page is a power of 2, varying between 512 bytes and 1 GB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

If the size of the logical address space is 2^m, and a page size is 2^nbytes, then the high-order $m- n$ bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:

where*p*isanindexintothepagetableand*d*isthedisplacementwithinthepage.

Example

Here, in the logical address, n= 2 and m = 4. Using a page size of 4 bytes and a physical memoryof32 bytes (8pages).Logical address 0 is page0, offset 0. Indexinginto the page table, wefind that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [=(5 × 4) +0]. Logicaladdress 3 (page 0,offset 3) maps to physicaladdress 23 [= (5 ×4) + 3]. Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 [= (6× 4) + 0]. Logical address 13mapstophysicaladdress9.



2. HardwareSupport

Methods for storing page table: Each operating system has its own methodsfor storing page tables.

a) Some allocate a page table for each process. A pointer to the page table is stored with the other register values (like the instruction counter) in the process control block. When the dispatcher is told to start a process, it must reload the user registers and define the correct hardware page-table values from the stored user page table.

b) Other operating systems provide one or at most a few page tables, which decreases the overhead involved when processes are context-switched.

HardwareImplementationofthePageTable

Registers

- In the simplest case, the page table is implemented as a set of dedicated **registers**. These registers should be built with very high-speed logic to make the paging-address translation efficient.
- Every access to memory must go through the paging map, so efficiency is a major consideration.
- The CPU dispatcher reloads these registers, just as it reloads the other registers. Instructions to load or modify the page-table registers are, of course, privileged, so that only the operating system can change the memory map. The use of registers for the page table is satisfactory if the page table is reasonably small.
- TheDECPDP-11 isanexample.

Page-TableBaseRegister(PTBR)

Most contemporary computers, allow the page table to be very large (for example, 1 million entries). For these machines, the use of fast registers to implement the page table is not feasible. Rather, the page table is kept in main memory, and a **page-table base register (PTBR)** points to the page table. Changing page tables requires changing only this one register, substantially reducing context-switch time.

Problem

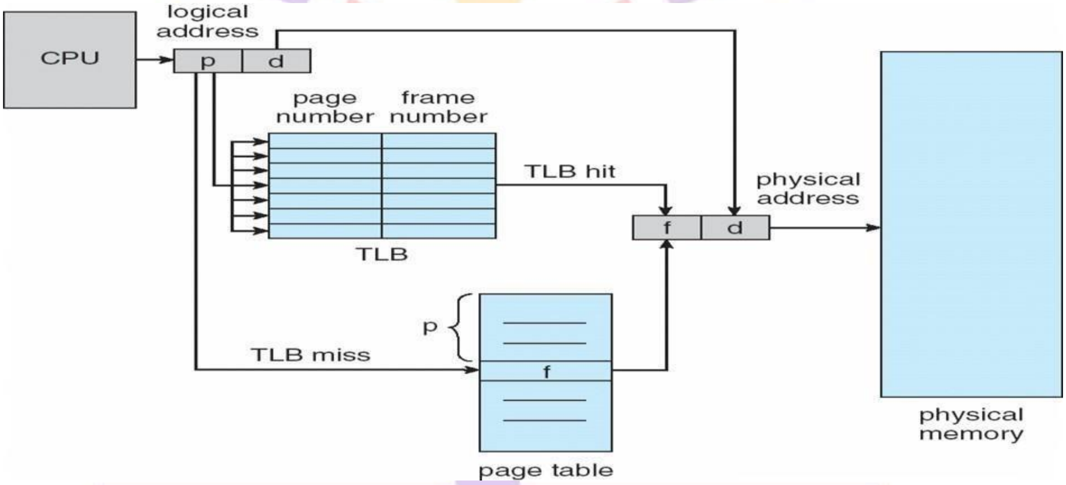
Theproblem with this approach is the time required to access a user memorylocation. If we want to access location *i*, we must first index into the page table, using the value in the PTBR offset by the page number for*i*. This task requires a memory access. It provides us with the frame number, which is combined with the page offset to produce the actual address. Wecan then access the desired placein memory. With this scheme, ***two*** memoryaccesses are needed to access a byte (one for the page-table entry, one for the byte). Thus, memory access is slowed by a factor of 2.

Solution:TranslationLook-AsideBuffer(TLB).

The standard solution to this problem is to use a special, small, fast lookup hardware cache called a **translation look-aside buffer (TLB)**. The TLB is associative, high-speed memory. Each entryin the TLB consists of two parts: a key(or tag) and a value.

Workingoftranslationlook-asidebuffer(TLB):

- The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries.
- When a logical address is generated by the CPU, its page numberis presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory.
- If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made.
- DependingontheCPU,thismaybedoneautomaticallyin hardwareor via aninterruptto the operating system.
- When theframenumber is obtained, wecan useit to access memory. In addition, weadd the page number and frame number to the TLB, so that theywill be found quicklyon the next reference. If the TLBis alreadyfull of entries, an existingentrymust be selected for replacement.



Address-SpaceIdentifiers(ASIDs)

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry. An ASID uniquely identifies each process and is used to provide address-space protection for that process.
- When the TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently running process matches the ASID associated with the virtual page. If the ASIDs do not match the attempt is treated as a TLB miss.

Hit ratio: The percentage of times that the page number of interest is found in the TLB is called the **hit ratio**

3. Protection

Memoryprotectioninapagedenvironmentisaccomplishedbyprotectionbits associated with each frame. Normally, these bits are kept in the page table.

Read–WriteorRead-OnlyBit

- Onebitcandefineapagetoberead–writeorread-only.Every referencetomemory goes through the page table to find the correct frame number.
- At the same time thatthe physicaladdress is being computed,the protection bits can be checked to verifythat no writes are being made to a read-only page.

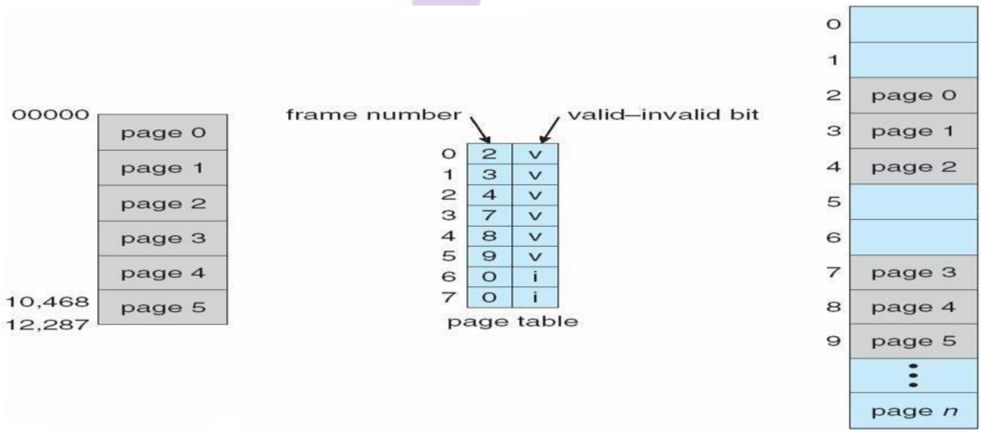
- An attempt to write to a read-only page causes a hardware trap to the operating system (or memory-protection violation).
- Wecaneasilyexpandthisapproachtoprovideafinerlevelofprotection.
- We can create hardware to provide read-only,read–write, or execute-only protection; or, by providing separate protection bits for each kind of access, we can allow any combination of these accesses. Illegal attempts will be trapped to the operating system.

Valid–Invalid Bit

- Oneadditionalbitisgenerallyattachedtoeachentryinthepagetable:a**valid–invalid** bit.
- Whenthis bit isset to *valid*,theassociatedpage isintheprocess’s logicaladdressspace and is thus a legal (or valid) page.
- Whenthebitissetto*invalid*,thepageisnotintheprocess’slogicaladdressspace. Illegal addresses are trapped byuse of the valid–invalid bit.
- Theoperatingsystemsetsthisbitforeachpagetoallowor disallowaccesstothepage.

Example

- Suppose, for example, that in a system with a 14-bitaddress space (0 to 16383), we have a program that should use only addresses 0 to 10468.
- Given a page size of 2 KB, Addresses in pages 0, 1, 2, 3, 4, and 5 are mapped normally through the page table.
- Any attempt to generate an address in pages 6 or 7, however, will find that the valid–invalid bit is set to invalid, and the computer will trap to the operating system (invalid page reference).



HardwareforProtection:Page-TableLengthRegister(PTLR)

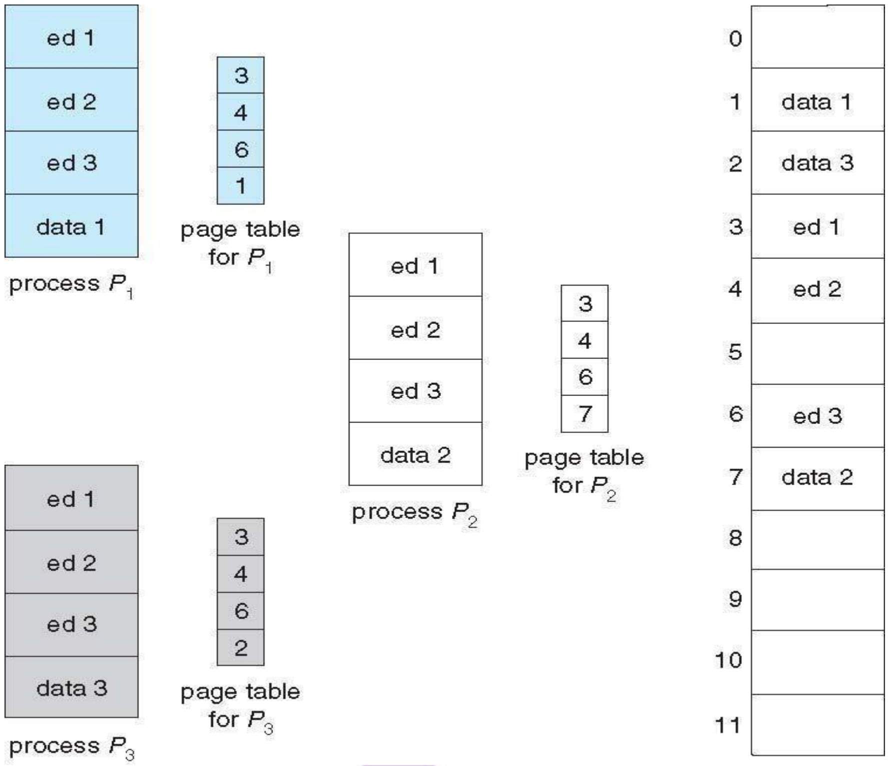
Some systems provide hardware, in the form of a **page-table lengthregister(PTLR)**, to indicate the size of the page table. This value is checked against every logical address to verify that the address is in the valid range for the process. Failure of this test causes an error trap to the operating system.

4. SharedPages

Anadvantageofpagingisthepossibilityof***sharing***commoncode. This consideration is particularly important in a time-sharing environment.

Example:

Consider a system that supports 40 users, each of whom executes a text editor. If the text editorconsistsof150 KBofcodeand 50 KB of dataspace, weneed 8,000 KBto support the 40 users. If the code is **reentrant code** or **pure code (**Reentrant code is non-self- modifying code: it never changes during execution. Thus, two or more processes can execute the same code at the same time. However, it can be shared.



Each process has its own copy of registers and data storage to hold the data for the process's execution. The data for two different processes will, of course, be different. Only onecopyoftheeditorneedbekeptinphysical memory.Each user's page tablemapsontothe same physical copy of the editor, but data pages are mapped onto different frames. Thus, to support 40 users, we need onlyone copyof the editor (150 KB), plus 40 copies of the 50 KB of data space per user. The total space required is now 2,150 KB instead of 8,000 KB—a significant savings.

Other heavily used programs can also be shared—compilers, window systems, run-time libraries, database systems, and so on.

SegmentationwithPaging

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages. Pages aresmaller than segments. Each Segment hasa page table which means every program has multiple page tables.

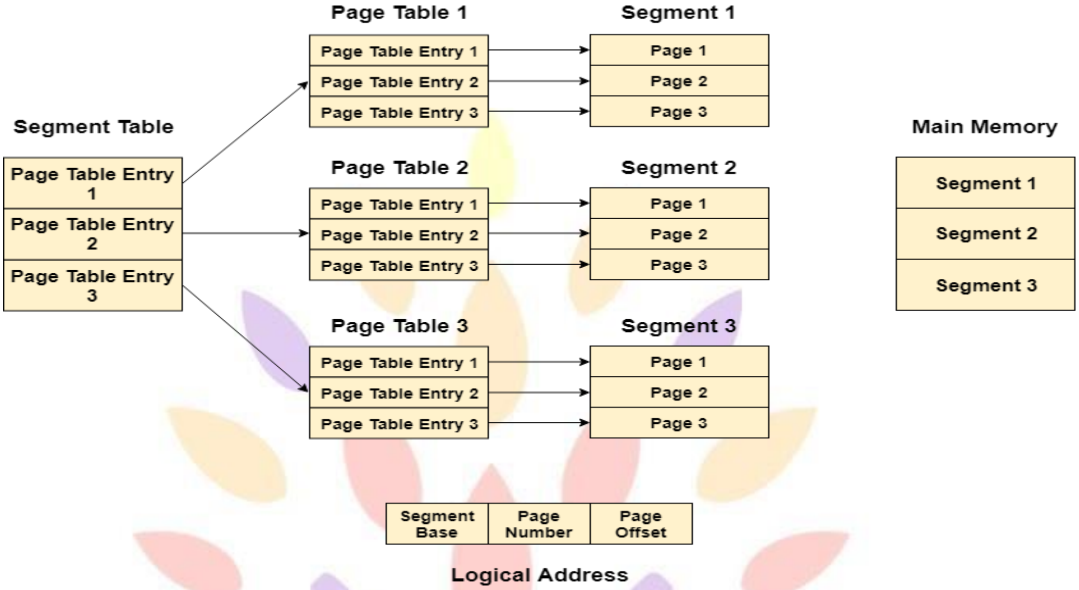
The logical address is represented as Segment Number (base address), Page number and page offset.

SegmentNumber→ItpointstotheappropriateSegmentNumber. Page Number → It Points to the exact page within the segment

NRCM your roots to success...

PageOffset→Usedasanoffset withinthepage frame

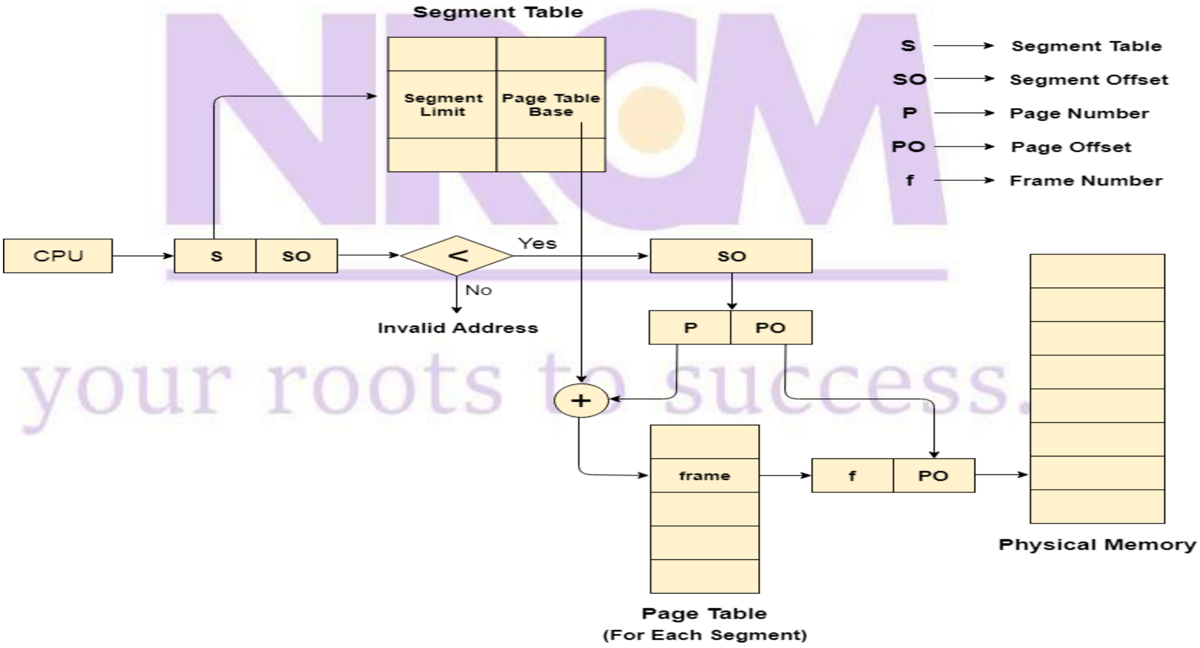
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



Translationoflogicaladdresstophysicaladdress

The CPU generates a logical address which is divided into two parts:Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory toget the desired word in the page of the certain segment of the process.



AdvantagesofSegmented Paging

1. Itreducesmemoryusage.
2. Pagetablesizeis limitedbythesegmentsize.
3. gmenttablehasonlyoneentrycorrespondingtooneactualsegment.
4. ExternalFragmentationisnotthere.
5. Itsimplifies memoryallocation.

DisadvantagesofSegmented Paging

1. InternalFragmentationwillbethere.
2. Thecomplexitylevelwillbemuchhigherascomparetopaging.
3. PageTablesneedtobecontiguouslystoredinthememory.

DemandPaging

Definition

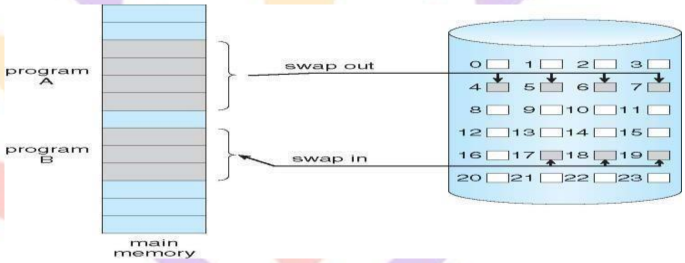
NRCM
your roots to success...

Loading the entire program into memory results in loading the executable code for ***all*** options, regardless of whether or not an option is ultimately selected by the user. An alternative strategy is to load pages only as they are needed. This technique is known as **demand paging** and is commonly used in virtual memory systems.

With demand-paged virtual memory, pages are loaded only when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.

LazySwapper

A demand-paging system is similar to a paging system with swappingwhere processes reside in secondary memory (usually a disk). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, though, we usea**lazy swapper**.Alazyswapperneverswaps apage intomemoryunlessthatpage willbe needed. In the context of a demand-paging system, use of the term "swapper" is technically incorrect. We thus use "pager," rather than "swapper," in connection with demand paging.



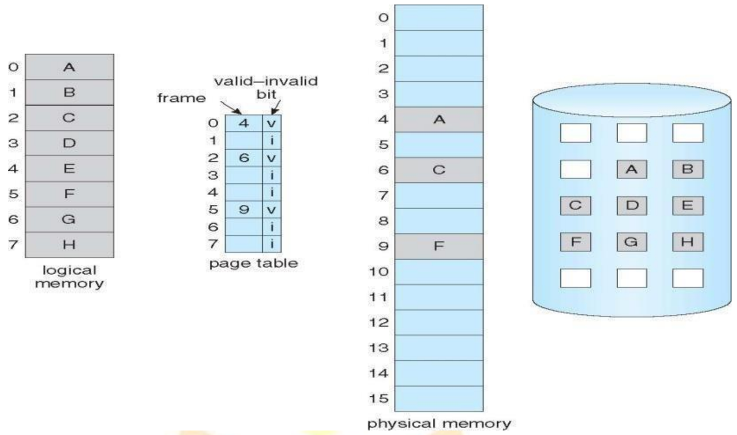
Transferofapagedmemoryto contiguousdisk space.

1. **Basic Concepts**

When a process is to be swapped in, the pager guesses which pages will beused before the process is swapped out again. Instead of swapping in a whole process, the pager brings onlythose pages into memory. Thus, it avoids readinginto memorypages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

Valid–InvalidBit

- We need some form of hardware support to distinguish between the pages that are in memoryand the pages that areon the disk. when this bit is set to "valid," the associated page is both legal and in memory.
- If thebitissetto "invalid,"thepageeitherisnotvalid(thatis,notinthelogicaladdress space of the process) or is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page thatis not currently in memory is either simply marked invalid or contains the address ofthe page on disk.

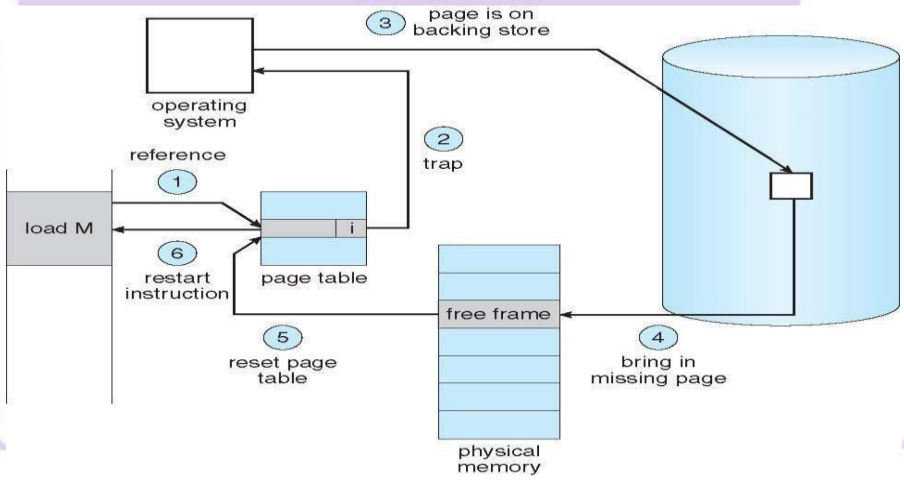


PageFault

Access to a page marked invalid causes a **page fault**. The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory.

Theprocedureforhandlingthispagefaultisstraightforward

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memoryaccess.
2. Ifthe reference was invalid, we terminatethe process. Ifit was valid but wehave not yet brought in that page, we now page it in.
3. Wefindafreeframe(bytakingonefromthefree-framelist, for example).
4. Wescheduleadisk operationtoreadthe desiredpageintothenewlyallocated frame.
5. When the disk read is complete, we modify the internal table kept with the process andthe page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now accessthe page as though it had always been in memory.



PureDemand Paging

In theextremecase, we can start executingaprocess with ***no*** pages in memory. When the operating system sets the instruction pointer to the firstinstruction of the process, whichis on a non-memory-residentpage, the process immediately faults for the page. After this pageisbroughtintomemory,theprocesscontinuestoexecute,faultingasnecessaryuntil every page that it needs is in memory. At that point, it can execute with no more faults. This scheme is **pure demand paging**: never bring a page into memoryuntil it is required.

HardwaretoSupportDemandPaging

- **Page table**. This table has the ability to mark an entry invalidthrough a valid–invalid bit or a special value of protection bits.
- **Secondary memory**. This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known as **swap space**.

A crucial requirement for demand pagingis the abilityto restart anyinstruction after a page fault. Because we save the state (registers, condition code, andinstruction counter) of the interrupted process when the page fault occurs, we must be able to restart the process in ***exactly*** the same place and state, except that the desired page is now in memory and is accessible. In most cases, this requirement is easy to meet.

A page fault may occur at any memory reference. If the page fault occurs on the instruction fetch, we can restart by fetching the instruction again. If a page fault occurs while wearefetchingan operand, wemust fetchand decodetheinstruction again and thenfetchthe operand.

2. PerformanceofDemandPaging

Demand paging can significantly affect the performance of a computer system let's compute the **effective access time** for a demand-paged memory. For most computer systems, the memory-access time, denoted *ma*, ranges from 10 to200 nanoseconds. As long as we have no page faults, the effective access time is equal to the memory access time.If,however, a page fault occurs, we must first read the relevant page from disk and then access the desired word.

Let p be the probability of a page fault ($0 \leq p \leq 1$). We would expect p to be close to zero—that is, we would expect to have only a few page faults.

The**effectiveaccesstime**isthen

EffectiveAccess Time= $(1-p)\times ma + p \times$pagefaulttime

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

Example:

With an averagepage-faultservice time of 8milliseconds anda memory access time of 200 nanoseconds, the effective access time in nanoseconds is

Effective AccessTime=$(1-p)\times(200) + p$(8milliseconds)

$= (1-p)\times 200 + p\times 8{,}000{,}000$

$= 200 + 7{,}999{,}800\times p.$

Wesee,then,thattheeffectiveaccesstimeisdirectlyproportionaltothe**page-fault rate**.

Anadditionalaspectofdemandpagingisthehandlingandoveralluseof swapspace. Disk I/O to swap space is generallyfaster than that to the file system. It is a faster file system because swap space is allocated in much larger blocks, and file lookups andindirect allocation methods are not used. However, swap space must still be used for pages not associated with a file (known as **anonymous memory**).

Mobile operating systems typically do not support swapping. Instead, these systems demand-page from the file system and reclaim read-only pages (such as code) from applications if memory becomes constrained. Such data can be demand-paged from the file system if it is later needed. Under iOS, anonymous memory pages are never reclaimed from an application unless the application is terminated or explicitlyreleases the memory.

PageReplacement

In Demand Paging,pages are only broughtintomemory only when needed. This has two benefits,

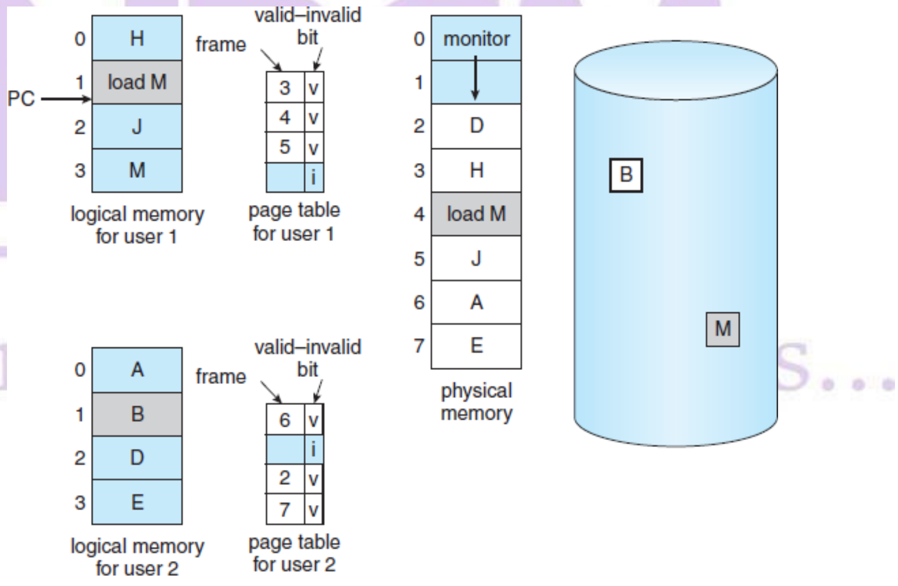
1. SavesI/Onecessarytoloadunusedpages.
2. Increasesthedegreeof multiprogramming.

Butincreasingdegreeofmultiprogrammingmayarisenewproblemcalled“Over allocating of memory”.

Over-AllocatingMemory

For example, there are 10 processes and each has 10 pages out of which only 5may be used. If there are 50 frames then we can allocate only 5 processes if all the 10 pages are loaded. But by using demand paging (we load only used or demanded pages) we can accommodate 10 processes as only 5 pages are in demand. Problem arises when suddenly a process needs all 10 pages but no frames are free.

Over-allocation of memory manifests itself as follows. While a user process is executing, a page fault occurs. The operating system determines where the desired page is residing on the disk but then finds that there are ***no*** free frames on the free-frame list; all memory is in use. The operating system has several options at this point.It could terminate the user process. This option is not the best choice. The operating system could instead swap out a process, freeing all its frames and reducing the level of multiprogramming. This option is a good one but requires page replacement.

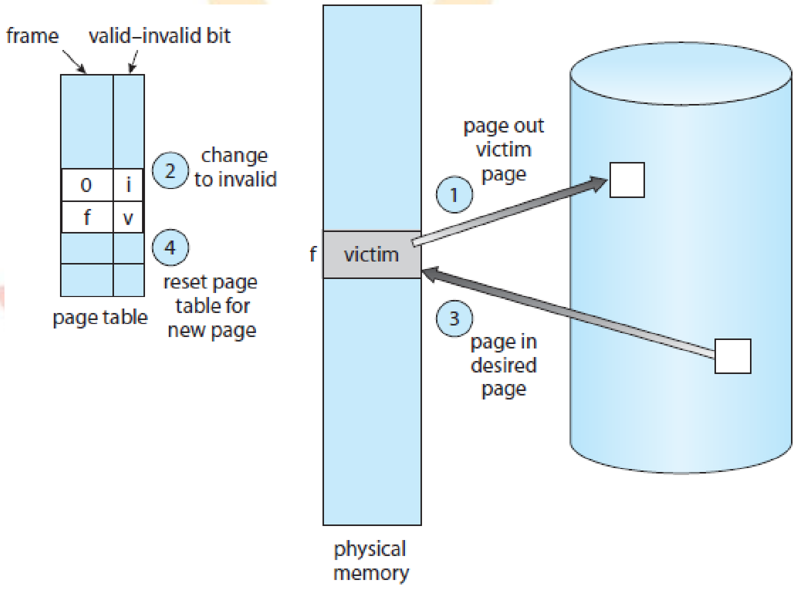


Need for page replacement.

1. BasicPageReplacement

Pagereplacementtakesthefollowingapproach,

1. Findthelocationofthedesiredpageonthedisk.
2. Findafree frame:
   a. Ifthereisafreeframe,useit.
   b. Ifthereisnofreeframe,useapage-replacementalgorithmtoselectavictim frame.
   c. Writethevictimframetothedisk;changethepageandframetablesaccordingly.
3. Readthedesiredpageintothenewlyfreedframe;changethepageandframetables.
4. Continuetheuser processfromwherethepagefault occurred.



Page replacement.

ModifyBit (or**DirtyBit**).

- If no frames are free, ***two*** page transfers (one out and one in)are required.This situation effectively doubles the page-fault service time and increases the effective access time accordingly. We can reduce this overhead by using a **modify bit** (or **dirty bit**).
- When this scheme is used, each page or frame has a modifybit associated with it in the hardware. The modify bit for a page is set by the hardware whenever any byte in the page is written into, indicating that the page has been modified.
- When we select a page for replacement, we examine it's modifybit. If the bit is set, we know that the page has been modified since itwas read in from the disk. In this case,we must write the page to the disk. If the modify bit is not set, however, the page has ***not*** been modified since it was read into memory. In this case, we need not write the memory page to the disk: it is already there.

MajorProblemstoImplementDemand Paging

Wemustsolvetwomajorproblemstoimplementdemandpaging:wemustdevelopa **frame-allocationalgorithm**anda**page-replacementalgorithm**.

That is, if we have multiple processes in memory, we must decide how many frames toallocatetoeachprocess;andwhenpagereplacementisrequired,wemust selecttheframes that are to be replaced.

ReferenceString

There are many different page-replacement algorithms. We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a **reference string**.

Wecangeneratereferencestrings

- Artificially(byusingarandom-numbergenerator,for example).
- We can trace a given system and record the address of each memory reference. But this produces large amount of data.

Toreducethis,weusetwo facts

a. First, for a given page size (and the page size is generally fixed bythe hardware or system),we need to consider only the page number, rather than the entire address.

b. Second,ifwehaveareferencetoapage*p*,thenanyreferencestopage*p*that ***immediately*** followwillnevercauseapagefault.

Example

Ifwetraceaparticularprocess,wemightrecordthefollowingaddresssequence: 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 reference string:

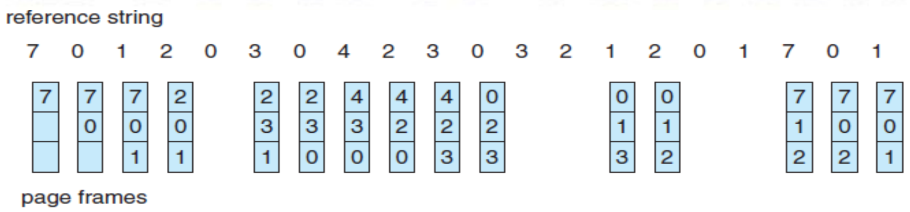
1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

PageReplacementAlgorithms

FIFOPageReplacement

- Thesimplestpage-replacementalgorithmisafirst-in,first-out(FIFO)algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.
- We can create a FIFO queue to hold all pages in memory. We replace the page atthe head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
- TheFIFOpage-replacementalgorithmiseasytounderstandand program.
- However, its performance is not always good. a bad replacement choice increases the page-fault rate and slows process execution. If we place an active page, some other page should be replaced to bring it back.

Example:



FIFO page-replacement algorithm.

Belady's anomaly:Forsomepage-replacementalgorithms,thepage-faultratemay***increase*** asthenumberofallocatedframes increases.

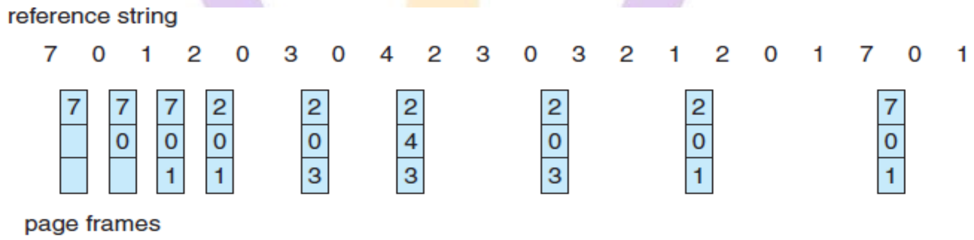
Considerthefollowingreferencestring: 1,2, 3,4, 1, 2, 5, 1, 2, 3, 4, 5

Numberoffaultsforfourframes(ten)is***greater***thanthenumberoffaultsforthreeframes (nine)

OptimalPageReplacement

- Itsaysthat**,**Replacethepagethatwillnotbeusedforthelongestperiodoftime.
- Ithasthelowestpage-faultrateofallalgorithmsandwillneversufferfromBelady's anomaly.
- Unfortunately,theoptimalpage-replacement algorithmisdifficulttoimplement,because it requires future knowledge of the reference string.
- Asaresult,theoptimalalgorithmisusedmainlyforcomparisonstudies.

Example:

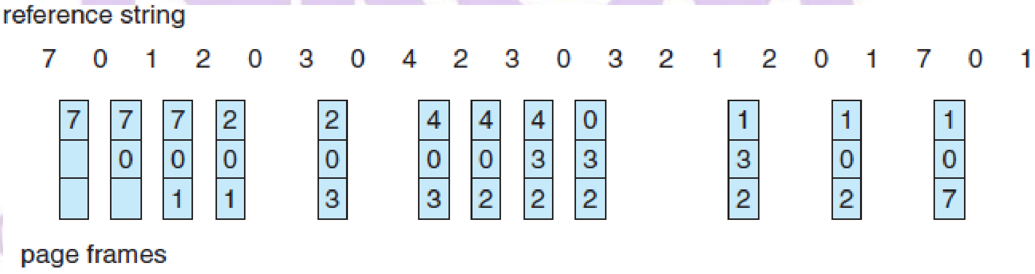


Optimal page-replacement algorithm.

LRUPageReplacement

- LRUreplacementassociateswitheachpagethetimeofthatpage'slastuse.
- Whenapagemustbereplaced,LRUchoosesthepagethathasnotbeenusedforthe longest period of time.
- Wecanthinkofthisstrategyastheoptimalpage-replacementalgorithmlooking backward in time, rather than forward.
- Likeoptimalreplacement,LRUreplacementdoesnotsufferfromBelady'sanomaly. Both belong to a class of page-replacement algorithms, called **stack algorithms**.

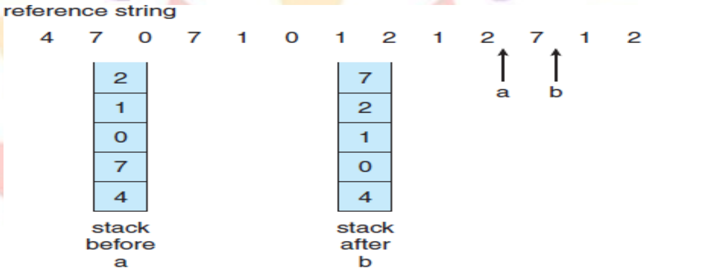
Example:



LRU page-replacement algorithm.

- The major problem is ***how*** to 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 time of last use.
- Twoimplementationsare feasible:

- **Counters**. In the simplest case, we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter. The clock is incremented for every memory reference.Whenever a reference to a page is made,the contents of the clock register are copied to the time-of-use field in the page-table entry for that page. In this way, we always have the "time" of the last reference to each page. We replace the page with the smallest time value. This scheme requires a search of the page table to find the LRU page and a write to memory (to the time-of-use field in the page table) for each memory access.
- **Stack**. Another approach to implementing LRU replacement is to keep a stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on the top. In this way, the most recently used page is always at the top of the stack and the least recently used page is always at the bottom. Because entries must be removed from the middle of the stack, it is best to implement this approach by using a doubly linked list with a head pointer and a tail pointer.



Use of a stack to record the most recent page references

LRU-ApproximationPageReplacement

- Few computer systems provide sufficient hardware support for true LRU page replacement. In fact, some systems provide no hardware support, and other page-replacement algorithms (such as a FIFO algorithm) must be used. Many systems provide some help, however, in the form of a **reference bit**.
- The reference bit for a page is set by the hardware whenever that page is referenced (either a read or a write to any byte in the page). Reference bits are associated with each entry in the page table.
- Initially, all bits are cleared (to 0) by the operating system.As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware. After some time, we can determine which pages have been used and which have not been used by examining the reference bits, although we do not know the ***order*** of use.Thisinformation is the basis for many page-replacement algorithms that approximate LRU replacement.

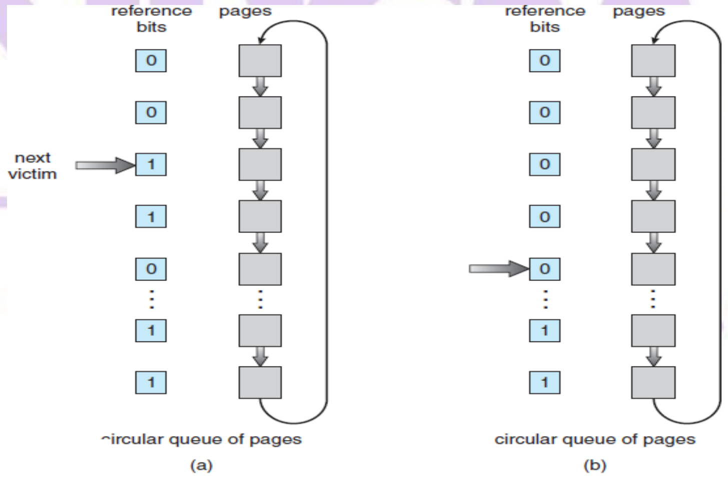
Additional-Reference-BitsAlgorithm

- Wecangainadditional orderinginformationby recording thereferencebits atregular intervals.
- Wecankeep an8-bitbyteforeachpageinatablein memory.
- At regularintervals (say, every 100 milliseconds), a timer interrupt transfers control tothe operating system.

- Theoperatingsystemshiftsthereferencebitforeachpageintothehigh-orderbitofits8- bit byte, shifting the other bits right by1 bit and discarding the low-order bit. These 8-bit shift registers contain the historyof page use for the last eight time periods.
- If the shift register contains 00000000, for example, then the page has not been used for eight time periods.
- A page that is used at least once in each period has a shift register value of 11111111. A page with a history register value of 11000100 has been usedmore recently than onewith a value of 01110111.
- If we interpret these 8-bit bytes as unsigned integers, the page with the lowest number is the LRU page, and it can be replaced. Notice that the numbers are not guaranteed to be unique, however. We can either replace (swap out) all pages with the smallest value or use the FIFO method to choose among them.

Second-ChanceAlgorithmORclock algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, we inspect its reference bit.
- If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
- When apage gets asecond chance, its referencebit is cleared, and its arrival timeis reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances).
- In addition, if a page is used often enough to keep its reference bit set, it will never be replaced.
- One way to implement the second-chance algorithm is as a circular queue.A pointer(that is, a hand on the clock) indicates which page is to be replaced next.
- When a frame is needed, the pointer advances until it finds a page with a 0 reference bit. As it advances, it clears the reference bits. Once a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position.



Second-chance (clock) page-replacement algorithm.

EnhancedSecond-ChanceAlgorithm

- We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair. With these two bits, we have the following four possible classes:

- **(0,0)**neitherrecentlyused normodified—best pageto replace.
- **(0, 1)**not recentlyused but modified—not quite as good, because the page will need to be written out before replacement.
- **(1,0)**recentlyused butclean—probablywillbeused againsoon.
- **(1, 1)** recently used and modified—probably will be used again soon, and the pagewill be need to be written out to disk before it can be replaced.

Counting-BasedPageReplacement

There are many other algorithms that can be used for page replacement. For example, we can keep a counter of the number of references that have been made to each page and develop the following two schemes,

LeastFrequentlyUsed(LFU)

- The **least frequently used (LFU)** page-replacementalgorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again.
- Since it was used heavily, it has a large count and remains in memoryeven though it isno longer needed.
- One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

MostFrequentlyUsed(MFU)

- The **most frequently used (MFU)** page-replacement algorithm is based on the argument thatthe pagewiththe smallest count wasprobablyjust brought inand has yet to beused.

NRCM
your roots to success...

UNIT-5

File system interface and operation- Access methods, directory structure, Protection, File system structure, Allocation methods, Free space management, Usage of Open, Create, Read, Write, Close, Iseek, Stat, ioctl System calls

File:-

Afileisanamedcollectionofrelatedinformationthatisrecordedonsecondary storage.

(or)Afileisthesmallestallotmentoflogicalsecondarystorage.

(or)A file is a sequence of bits, bytes, lines, or records, the meaning of which is defined bythe file's creator and user. Many different types of information may be stored in a file.

FileAttributes

FileAttributesgivestheOperatingSysteminformationaboutthefileandhowitisintended to use.

Afile'sattributesvaryfromoneoperatingsystemto another but typicallyconsistofthese:

- **Name**.Thesymbolicfilenameistheonlyinformationkeptinhumanreadableform.
- **Identifier**. This unique tag, usuallya number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type**.Thisinformationisneededforsystemsthatsupportdifferenttypesoffiles.
- **Location**. This information is a pointer to a device and to the location of the file on that device.
- **Size**. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection**. Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification**. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

Some newer file systems also support**extended file attributes**, including characterencoding of the file and security features such as a file checksum.

FileTypes

When we design a file system we always consider whether the operating system should recognizeand supportfiletypes. Ifan operatingsystem recognizes thetypeofafile, it can then operate on the file in reasonable ways. A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts—a name and an extension, usuallyseparated bya period. Examples include resume.docx, server.c, and ReaderThread.cpp.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Common file types.

FileStructure

Filetypesalso canbeused to indicatethe internalstructureofthe file. Someoperating systems extend this idea by supporting their own file structures. But it has the following disadvantages

1. If operating system support multiple file structures: the resulting size of the operating system is large.
2. Some applications may require information structured in a way that is not supported by the OS some operating systems impose (and support) a minimal number offile structures. This approach has been adopted in UNIX, Windows, and others.

InternalFileStructure

Block Structure

Disksystemstypicallyhaveawell-definedblocksizedeterminedbythesizeofa sector.AlldiskI/Oisperformedinunitsofoneblock(physicalrecord),andallblocksarethe same size.

RecordStructure

Files contain a sequence of fixed length records. Physical records may or may not get exact match with the logical record. Logical records even vary in length.

your roots to success...

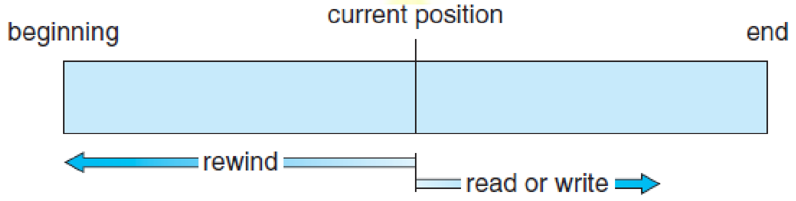
Accessmethods

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in the following ways,

1. SequentialAccess

- The simplest access method is **sequential access**. Information in the file is processed in order, one record after the other. It is based on a tape model of a file and works as well on sequential-access devices.

- **Example**:EditorsandCompilersusuallyaccessfiles inthis fashion.
- **Operations**
  - ➢ A read operation—read next ()—reads the next portion of the file and automaticallyadvances a filepointer, which tracksthe I/O location.Similarly, the writeoperation—write next ()—appends to the end ofthe file and advances to the end of the newly written material (the new end of file). On some systems, a program maybe able to skip forward or backward *n* records for some integer *n*—perhaps only for *n* = 1.



Sequential-access file.

2. **DirectAccess**(or**RelativeAccess**)

- Another method is **direct access** (or **relative access**). Here, a file is made up of fixed-length **logical records** that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.
- **Examples**:
  - ➢ Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.
  - ➢ On an airline-reservation system, we might store all the information about a particular flight (for example, flight 713) in the block identified by the flight number. Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people's names or search a small in-memory index to determine a block to read and search.
- **Operations**
  - ➢ For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have read (n), where *n* is the block number, rather than read next (), and write (n) rather than write next ().
  - ➢ An alternative approach is to retain read next () and write next (), as with sequential access, and to add an operation position file (n) where *n* is the block number. Then, to affect a read (n), we would position file (n) and then read next().

3. **IndexedAccess**
   - It involves the construction of an index for the file. The **index**, like an index in the back of a book, contains pointers to the various blocks. To find a record in the file, we first search the index and then use the pointer to access the file directlyand to find the desired record.
   - **Example**:
     A retail-price file might list the universal product codes (UPCs) for items, with the associated prices. Each record consists of a 10-digit UPC and a 6-digit price, for a 16-byte record. If our disk has 1,024 bytes per block, we can store 64 records per block.A file of 120,000 records would occupy about 2,000 blocks (2 million bytes). By keeping the file sorted by UPC, we can define an index consisting of the first UPC in each block. This index wouldhave 2,000 entries of 10 digits each, or 20,000 bytes, and thus could be kept in memory. To find the price of a particular item, we can make a binary search of the index. From this search, we learn exactly which block contains the desired record and access that block. This structure allows us to search a large file doing little I/O.

With large files, the index file itself may become too large to be kept in memory. One solutionistocreate anindex forthe index file.Theprimaryindex file containspointersto secondary index files, which point to the actual data items. For example, IBM's indexed sequential-access method (ISAM)usesasmallmasterindexthat pointsto disk blocksofa secondary index.

DirectoryOverview

Afilesystem canbecreatedoneachofthese parts ofthedisk. Anyentitycontaininga file system is generally known as a **volume**. Each volume thatcontains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory** or **volume table of contents**. The device directory (or **directory**) records information—such as name, location, size, and type—for all files on that volume.

The directory can be viewed as a symbol table that translates file names into their directoryentries. The following are the operations that are to be performed on a directory:

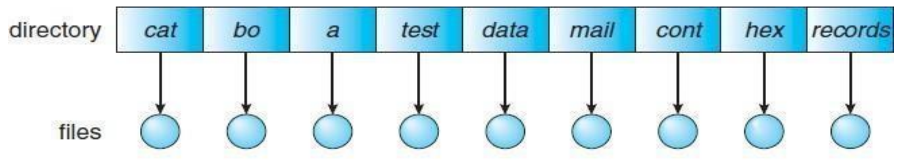
- **Searchfora file**.
- **Createafile**
- **Deleteafile**.
- **Listadirectory**.
- **Renameafile**
- **Traversethefilesystem**

Directory Structure

Themostcommonschemesfordefiningthelogicalstructureofadirectoryare thefollowing,

1. **Single-LevelDirectory**
   - The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easyto support and understand.

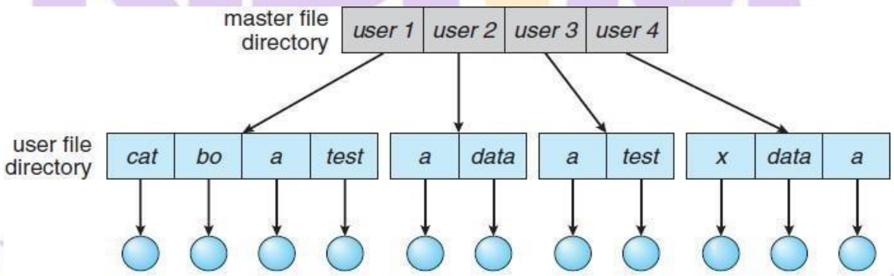
- **Limitations**
  - All files are in the same directory, theymust have unique names. If two users call their data file test.txt, then the unique-name rule is violated.
  - Even a single user on a single-level directory may find it difficult toremember the names of all the files as the number of files increases. Keeping track of so many files is a problem.



Single-level directory.

2. Two-LevelDirectory

- The standard solution to eliminate confusion of file names among different users isto create a separate directory for each user.
- Sothe two leveldirectorystructurecontains2 directories
  - MasterFile Directory(MFD) atthetoplevel.
  - UserFileDirectory(UFD)atthesecondleveland
  - Actualfilesareatthethirdlevel.
- Each user has his own **user file directory (UFD).** When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed byuser name or account number, and each entrypoints to the UFD for that user
- Whenauserreferstoaparticularfile,onlyhisownUFDis searched.
- To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists.
- To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.



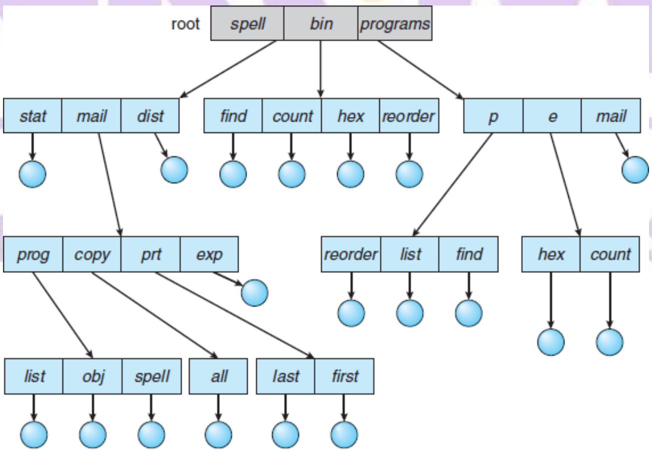
Two-level directory structure.

- Although the two-level directory structure solves the name-collision problem, it still has disadvantages.
- This structure effectively isolates one user from another. Isolation is an advantage when the users are completely independent butis a disadvantage when the users want to cooperate on some task and to access one another's files. Some systems simply do not allow local user files to be accessed byother users.

- If access is permitted one user must have the abilityto name a file in another user's directory. To name a file uniquely, the user must give both user name and file name as Path Name.
- If user A wishes to access her own test file named test.txt, she can simply refer to test.txt. Toaccess thefile namedtest.txt ofuserB (with directory-entrynameuserb), however,shemighthavetoreferto/userb/test.txt(windowsos)and/u/pbg/test(Unix, Linux).
- A special situation occurs with the system files. If a user wants them, they are searched in USD if found ok if not found we should copy the system files into each UFD but copying all the system files would waste an enormous amount of space.
- The standard solution is to use special user directory. Whenever a file name is given to be loaded, the operatingsystem first searches the local UFD. If the file is found, it is used. If it is not found, the system automatically searches the specialuserdirectory that contains the system files.
- The sequenceofdirectories searchedwhenafileis namediscalledthe**searchpath**.

3. Tree-StructuredDirectories

- A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory(1). Special system calls are used to create and delete directories.
- **CurrentDirectory**
  - Each process has a current directory. The **current directory**should contain most of the files that are of current interest to the process.
  - When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory (using change directory () system call) to be the directory holding that file.



Tree-structured directory structure.

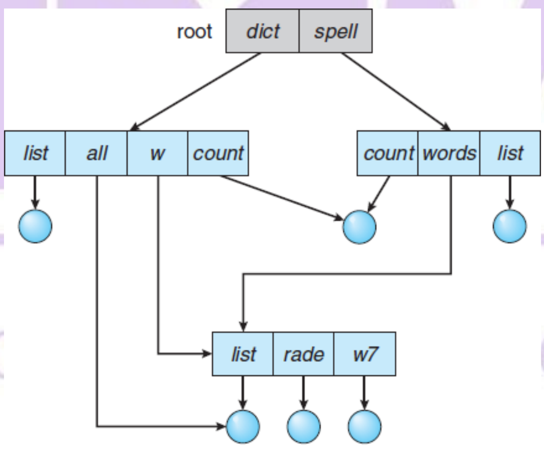
- **PathNames**

- ItdescribesthepaththeOSmust taketogettosomepoint.
- Pathnamescanbeoftwo types: absoluteand relative.
- An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A**relativepathname**definesapathfromthecurrentdirectory.
- If the current directory is root/spell/mail, then the relative path name prt/first refers to the same file as does the absolute path name root/spell/mail/prt/first.

- **Deletionofadirectory**
  - If a directory is empty, its entry in the directory that contains it can simply be deleted.
  - However, suppose the directory to be deleted is not empty but contains several files or subdirectories.
  - One of two approaches can be taken. Some systems will not delete a directory unless it is empty. Thus, to delete a directory, the user must first delete all the files in that directory. If any subdirectories exist, this procedure must be applied recursivelyto them, so that theycan be deleted also. This approach can result in a substantial amount of work.
  - An alternative approach, such as that taken by the UNIX rm command, is to provide an option: when a request is made to delete a directory, all that directory's files and subdirectories are also to be deleted.

4. Acyclic-GraphDirectories

- A tree structure prohibits the sharing of files or directories. An **acyclic graph** i.e.**,** a graph with no cycles which allows directories to share subdirectories and files.
- The same file or subdirectory may be in two different directories. An acyclic-graph directory structure is more flexible than a simple tree structure, but it is also more complex.



Acyclic-graph directory structure.

- **Implementation**

a. Link: A common way is to create a new directory entry called a link. A **link** is effectively a pointer to another file or subdirectory. A link may be implemented as an absolute or a relative path name. When a reference to a file is made, we searchthedirectory.Ifthe directory entry ismarkedasa link,then thename of therealfile is included inthe link information. We **resolve** the link byusingthat path name to locate the real file.

b. **Duplication:**Sharedfilesduplicateallinformationabouttheminbothsharing directories. Thus, both entries are identical and equal. A major problem with duplicate directoryentries is maintaining consistencywhen a file is modified.

- **Problems**
  - A file may now have multiple absolute path names creating problem in traversing.
  - **Deletion**: When can the space allocated to a shared file be deallocated and reused?
    - One possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointers to the now-nonexistent file.
    - Another possibility occurs when symbolic links are used. The deletion of a link need not affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is deallocated, leaving the links dangling.Wecansearch fortheselinksandremovethemas well,butunless a list of the associated links is kept with each file, this search can be expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist and can fail to resolve the link name; the access is treated just as with any other illegal file name.
    - Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we musthave some mechanism fordetermining that the last reference to the file has been deleted. We could keep a list of all references to a file (directory entries or symbolic links). When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link or directoryentryis deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty.

Protection

When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).

Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.

File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism, accidentaldeletion , Bugs in the file-system software etc.,

Protection can be provided in many ways. For a single-user laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. In a larger multiuser system, however, other mechanisms are needed.

1. TypesofAccess

Protection mechanisms provide controlled access by limiting the types of file access that can bemade. Access is permitted ordenieddependingon several factors, oneof which is the type of access requested. Several different types of operations may be controlled:

- **Read**.Readfromthefile.
- **Write**.Writeorrewritethefile.
- **Execute**. Loadthefileintomemoryandexecuteit.
- **Append**.Writenewinformationattheend ofthefile.
- **Delete**.Deletethefileand freeitsspaceforpossiblereuse.
- **List**.List thenameandattributesofthefile.

Otheroperations,suchasrenaming,copying,andeditingthefile,mayalsobecontrolled.

2. AccessControl

Themost commonapproachtotheprotectionproblemistomakeaccessdependenton the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement identity dependent access is to associate with each file and directory an **access-control list (ACL)**specifying user names and thetypes of access allowed for each user.

Whenauserrequestsaccesstoaparticularfile,theoperatingsystemchecksthe access list associated with that file. Ifthat user is listed forthe requested access, the access is allowed. Otherwise, aprotection violationoccurs,andtheuser job isdenied accesstothefile.

This approach has the advantage of enabling complex access methodologies.The main problem with access lists is their length. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

These problems can be resolved by use of a condensed version of the access list. To condense thelength oftheaccess-control list, manysystems recognizethreeclassifications of users in connection with each file:

- **Owner**.Theuserwhocreatedthefileistheowner.
- **Group**.Asetofuserswhoaresharingthefileandneedsimilaraccessisagroup,or work group.
- **Universe**.Allother usersinthesystemconstitutetheuniverse.

The most common recent approach is to combine access-control lists with the more general (and easier to implement) owner, group, and universe access control scheme.

For this scheme to work properly, permissions and access lists must be controlled tightly. This control can be accomplished in several ways. For example, in the UNIX system, groups can be created and modified onlybythe manager of the facility(or byanysuperuser). Thus, control is achieved through human interaction.

With the more limited protection classification, only three fields are needed to define protection. Often, each field is a collection of bits, and each bit either allows or prevents the access associated with it. For example, the UNIX systemdefines three fields of3 bits each—rwx, where r controls read access, w controls write access, and x controls execution. A separate field is kept for the file owner, for the file's group, and for all other users. In this scheme, 9 bits per file are needed to record protection information.

Example:
19-rw-r--r--+1jimstaff130May2522:13 file1

3. OtherProtectionApproaches

Another approach to the protection problem is to associate a password with each file. Just as access to the computer system is often controlled by a password, access to each file can be controlled in the same way. If the passwords are chosen randomly and changed often, this scheme may be effective in limiting access to a file. The use of passwords has a few disadvantages, however.

First, the number of passwords that a user needs to remember may become large, making the scheme impractical.

Second, if only one password is used for all the files, then once it is discovered, all files are accessible; protection is on an all-or-none basis. Some systems allow a user to associate a password with a subdirectory, rather than with an individual file, to address this problem.

FileSystemStructure

Disksprovidemostofthesecondarystorageonwhichfilesystemsaremaintained.Two characteristics make them convenient for this purpose are,

1. Adisk canberewritten.
2. Adisk canaccessdirectlyanyblock ofinformation itcontains.

File systems provide efficient and convenient access to the disk by allowing data tobe stored, located, and retrieved easily.

DesignProblemsinaFile System

1. Howthefilesystemshouldlooktotheuseri.e.file,andthedirectorystructurefor organizing files.
2. Creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

Layereddesign ofaFilesystems

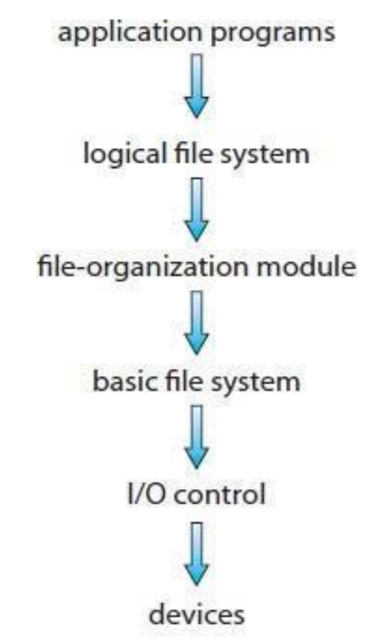
Eachlevelin thedesignusesthefeaturesoflowerlevelstocreatenewfeaturesforuse by higher levels.

Application Programs

Itcontainsuser codethatismakingarequest.

LogicalFile System

The **logical file system** manages metadata information. Metadata includes all of thefile-systemstructureexcepttheactualdata.Thelogicalfilesystemmanagesthedirectory structuretoprovidethefile-organizationmodulewiththisinformation.



Layered file system

File-OrganizationModule

The**file-organization module**knowsaboutfiles andtheir logical blocks and physical blocks. By knowing the type of file allocation used and the location of the file, the file organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.

BasicFileSystem

The **basic file system** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on thedisk. Eachphysicalblock is identified byitsnumericdiskaddress.

I/O control

The**I/Ocontrol** levelconsistsofdevicedrivers andinterrupts handlers to transfer information between the main memory and the disk system. It acts like a translator, inputting high-level commands such as "retrieve block 123."And outputting low-level, hardware -specific instructionsthatareusedbythehardware controller

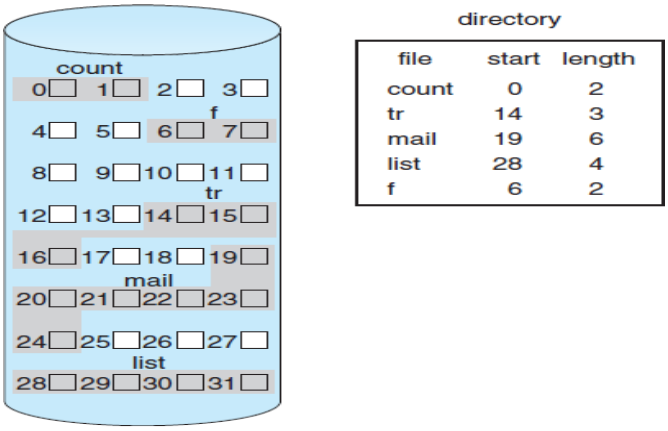
Devices

Thesearetheactualhardwaredeviceslikedisk.

Allocationmethods

Manyfiles can be stored on the same disk. Themain problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. The following are the three major methods of allocating disk space that are in wide use:

1. **ContiguousAllocation**
   - **Contiguous allocation** requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk.
   - Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, $b + 1$, $b + 2$, $b + n - 1$. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
   - **Accessingafile:**
     Accessing a file that has been allocated contiguously is easy. It supports both sequential and random access. For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block. For direct access to block i of a file that starts at block b, we can immediatelyaccess block $b + i$.



Contiguous allocation of disk space.

- **Drawbacks**
  - Finding space for a new file. The system chosen to manage free space determine show this task is accomplished. First fit and best fit are the most common strategies used to select a free hole fromthe set of available holes.
  - **ExternalFragmentation**

    As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks.It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.
  - **Solutiontoexternalfragmentation**

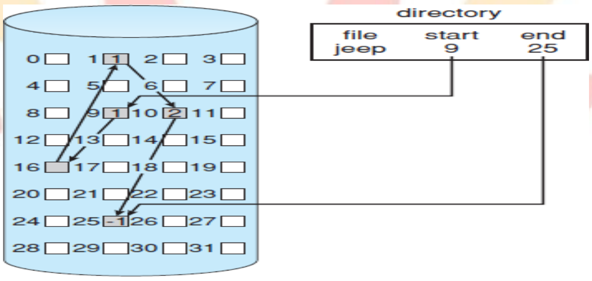
    Copy an entire file system onto another disk. The original disk is then freed completely, creatingonelargecontiguous freespace. Wethencopythefiles back onto the original disk by allocating contiguous space from this one large hole. This scheme effectively **compacts** all free space into one contiguous space, solving the fragmentation problem.
  - Determining how much space is needed for a file. When the file is created, the total amount of space it will need must be found and allocated. If we allocate too little space to a file, we may find that the file cannot be extended.

    Two possibilities then exist. First, the user program can be terminated, with an appropriate error message. The user must then allocate more space and run the program again. These repeated runs may be costly. To prevent them, the user will normally overestimate the amount of space needed, resulting in considerable wasted space. The other possibility is to find a larger hole, copy the contentsofthefiletothenewspace, and releasethe previous space.Thisseriesof actions can be repeated as long as space exists, although it can betime consuming. The user need never be informed explicitly about what is happening, however; the system continues despite the problem, although more and more slowly. Even if the total amount of space needed for a file is known in advance, preallocation may be inefficient. A file that will grow slowlyover a long period.
- **ModifiedContiguous-Allocation**
  - To minimize these drawbacks, some operating systems use a modified contiguous-allocationscheme.Here,acontiguouschunkofspaceisallocated

initially.Then,ifthatamountprovesnottobelargeenough,anotherchunkof contiguous space, known as an **extent**, is added.

2. LinkedAllocation

- **Linked allocation** solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might startat block 9 and continue at block 16, then block 1, then block 10, and finally block 25.
- To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to null (the end-of-list pointer value) to signify an empty file. The size field is also set to 0.
- Awritetothefilecauses thefree-spacemanagementsystemto findafree block,and this new block is written to and is linked to the end of the file.
- Toread afile,wesimplyreadblocks byfollowingthepointers fromblocktoblock.



Linked allocation of disk space.

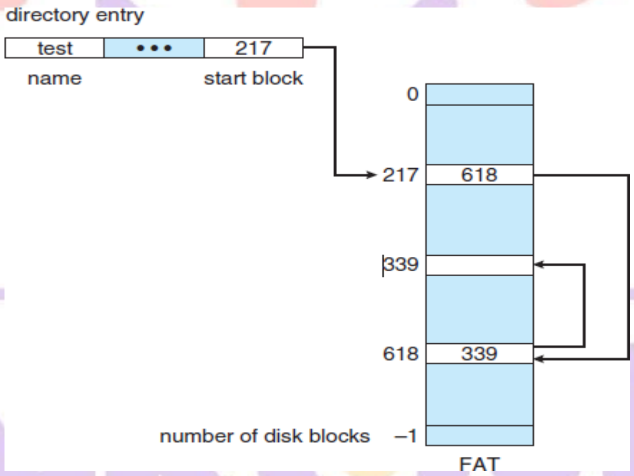
- **Advantages**
  - ➢ There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.
  - ➢ Thesizeofafileneednotbedeclaredwhenthefileiscreated.Afilecan continue togrowaslongasfreeblocksareavailable.Consequently,itisnevernecessaryto compact disk space.
- **Disadvantages**
  - ➢ The major problem is that it can be used effectively only for sequential-access files. To find the i^{th} block of a file, we must start at the beginning of that file and followthepointersuntilweget to the i^{th} block.Each access to apointerrequires a disk read, and some require a disk seek. Consequently,it is inefficient to supporta direct-access capability for linked-allocation files.
  - ➢ Anotherdisadvantage is thespace required for the pointers. If apointerrequires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers,ratherthan forinformation. Each filerequires slightlymorespace thanit would otherwise.
  - ➢ The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks. For instance, the file system maydefine a cluster as four blocks and operate on the disk only in cluster units.
  - ➢ Another problem of linked allocation is reliability the files are linked together by pointersscatteredallover thedisk,andconsiderwhatwouldhappenifapointer

werelostordamaged.

- One partial solution is to use doubly linked lists, and another is to store the file name and relative block number in each block. However, these schemes require even more overhead for each file.

VariationonLinkedAllocation

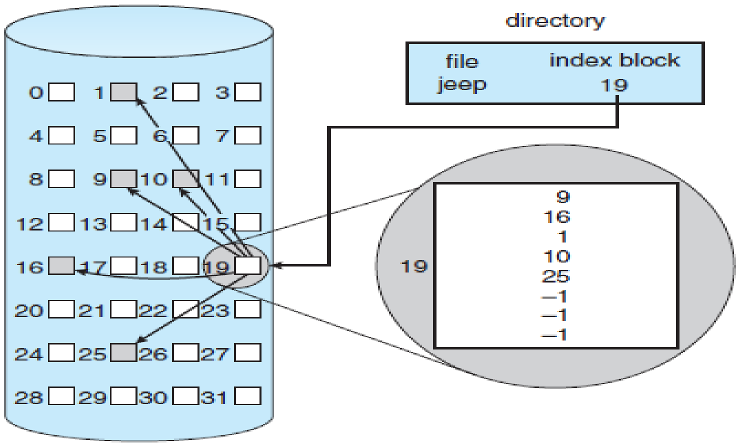
- An important variation on linked allocation is the use of a **file-allocation table (FAT)**. This simple but efficient method of disk-space allocation was used by the MS-DOS operating system.
- A section of disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed byblock number.
- TheFAT is used in much the same wayas a linked list. Thedirectoryentrycontains the block number of the first block of the file. The table entry indexed by thatblocknumber contains the block number of the next block in the file. This chain continues until it reaches the last block, which has a special end-of-file value as the table entry.
- An unused block is indicated by a table value of 0. Allocating a new block to a file is a simple matter of finding the first 0-valued table entry and replacing the previous end-of-file value with the address of the new block. The 0 is then replaced with the end-of-file value.



3. IndexedAllocation

- Linked allocation solvesthe external-fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.
- **Indexedallocation** solvesthisproblembybringingallthe pointerstogetherintoone location: the **index block**.
- Each file has its own index block, which is an array of disk-block addresses. The i^{th}entry in the index block points to the i^{th}block of the file. The directory contains the address of the index block.
- Tofindand readthei^{th}block, weusethepointerinthei^{th}index-blockentry.
- When the file is created, all pointers in the index block are set to null. When the i^{th}block is first written, a block is obtained from the free-space manager, andits address is put in the i^{th}index-block entry.

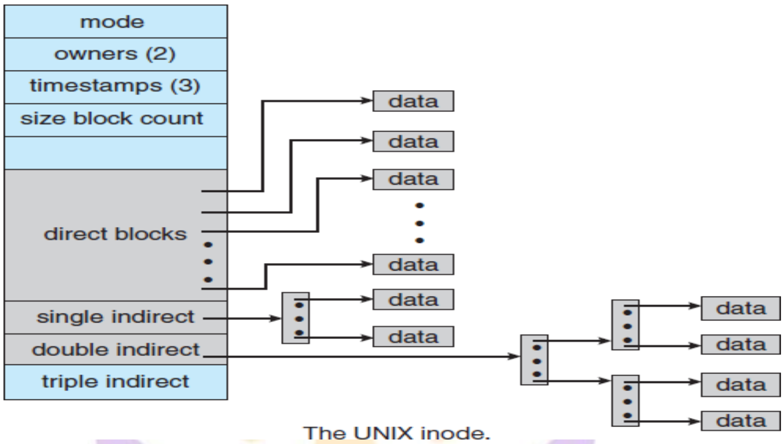
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.



Indexed allocation of disk space.

- **Disadvantages**
  - Indexedallocationdoessufferfromwasted space.
  - The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.
- **MechanismsforimplementingIndexBlock**
  - **Linked scheme**. An index block is normally one disk block. Thus, it can be read and written directlybyitself. To allow forlarge files, wecan link together several index blocks.For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (thelast wordin theindex block)isnull(forasmallfile) oris apointerto another index block (for a large file).
  - **Multilevel index**. A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size. With 4,096-byte blocks, we could store 1,024 four-byte pointers in an index block.Two levels of indexes allow 1,048,576 data blocks and a file size of up to 4 GB.
  - **Combined scheme**. Another alternative, used in UNIX-based file systems, is to keep the first, say, 15 pointers of the index block in the file's inode. The first 12of these pointers point to **direct blocks**; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files (of no more than 12 blocks) do notneed a separate index block. If the block size is 4 KB, then up to48 KB of data can be accessed directly. The next three pointers point to **indirect blocks**. The first points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The second pointstoa**doubleindirectblock**,whichcontainstheaddressofablockthat

containstheaddressesofblocksthatcontainpointerstotheactualdatablocks. The last pointer contains the address of a **triple indirect block**.



The UNIX inode.

Free-spaceManagement

To keep track of free disk space, the system maintains a **free-space list**. The free-space list records all free disk blocks—those not allocated to some file or directory. The following are implementations of free space list.

1. **BitVector**
   - Free-space list is frequently implemented as a **bit map** or **bit vector.** Each block is represented by1 bit. If the block is free, the bitis 1; if the block is allocated, the bitis 0.
   - For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be
     001111001111110001100000011100000...
   - **Advantage**

     Themain advantageof this approach is its relativesimplicityand its efficiency in finding the first free block or *n* consecutive free blocks on the disk. Indeed, many computers supply bit-manipulation instructions that can be used effectively for that purpose. One technique for finding the first free block on a system that uses a bit-vector to allocate disk space is to sequentially check each word in the bit map to see whether that value is not 0, since a0-valued word contains only 0 bits and represents a set of allocated blocks. The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.

     Thecalculationoftheblocknumberis

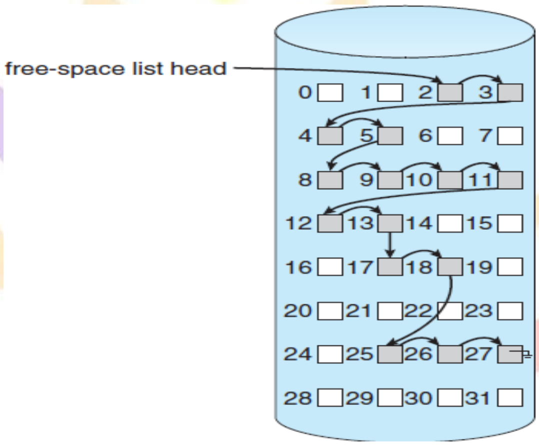
     (numberofbitsper word)×(numberof0-valuewords)+offset offirst1 bit.
   - **Disadvantage**

     Bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs). Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.

2. **Linked List**
   - Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the firstfree blockin a special location on the disk and

caching it in memory. This first block contains a pointer to the next free disk block, and so on.

- For example, blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.In this situation, we would keep a pointerto block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point tom block 8, and so on.
- **Disadvantages**
  This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time



Linked free-space list on disk.

3. Grouping

- A modification of the free-list approach stores the addresses of *n* free blocks in the first free block. The first *n*−1 of these blocks are actually free. The last, block contains the addresses of another *n* free block, and so on. The addresses of a large number of free blocks can now be found quickly.

4. Counting

- Several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
- Thus, rather than 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 entryin the free-space list then consists of a disk address and a count.

5. SpaceMaps

- Oracle's **ZFS** file system was designed to encompass huge numbers of files, directories, and even file systems.
- In its management of free space, ZFS creates **metaslabs** to divide the space on the device into chunks of manageable size. Each metaslab has an associated space map.
- The space map is a log of all block activity(allocating and freeing), in time order, in counting format. When ZFS decides to allocate or free space from a metaslab, it loads the associated space map into memory in a balanced-tree structure (for very efficient operation), indexed byoffset, and replays the login to that structure.

- Thein-memoryspacemap is thenan accuraterepresentation oftheallocated and free space in the metaslab.

Systemcallsforfileoperations-open(),read(),write(),close(),seek(),unlink() (FileOperations)

create()

This is used to create a file. Two steps arenecessaryto create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

open ()

Many systems require that an open () system call be made before a file is first used. When a file has been opened its entry is added in the open file table. It also contains open count associated with each file to indicate how manyprocesses have the file open.

read()

To read from a file, we use a system call that specifies the name of the file and **read pointer**tothelocationin thefile wherethenextreadisto takeplace. Oncethe read has taken place, the read pointer is updated.

write()

To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location.

The system must keep a **write pointer** to the location in the file where the next writeis to take place. The write pointer must be updated whenever a write occurs.

close()

This closes a file. Each close () decrements the open countand when the count reaches zero, the file is no longer in use so it can be closed.

delete()

To delete a file, we search the directory for the named file. Having found the associated directoryentry, we release all file space, so that it can be reused byother files, and erase the directory entry.

truncate()

The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

seek()

It is also called as Reposition. The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O.

unlink()

Deletes a name from the file system. If that name was the last link to a file and no processeshavethefileopenthefileisdeletedandthespaceitwasusingismadeavailablefor reuse.