PART -A

UNIT -1 INTRODUCTION TO OPERATING SYSTEMS, SYSTEM STRUCTURES: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management; Protection and security; Distributed system; Special-purpose systems; Computing environments. Operating System Services; User -Operating System interface; System calls; Types of system calls; System programs; Operating System design and implementation; Operating System structure; Virtual machines; Operating System generation; System boot.

6 Hour UNIT -

2 Process Management: Process concept; Process scheduling; Operations on processes; Inter-process communication. Multi-Threaded Programming: Overview; Multithreading models; Thread Libraries; Threading issues. Process Scheduling: Basic concepts; Scheduling criteria; Scheduling algorithms; Multiple-Processor scheduling; Thread scheduling.**7 Hours UNIT -3 PROCESS SYNCHRONIZATION:** Synchronization: The Critical section problem; Peterson's solution; Synchronization hardware; Semaphores; Classical problems of synchronization; Monitors.

7 Hours

UNIT -4 DEADLOCKS: Deadlocks: System model; Deadlock characterization; Methods for handling deadlocks; Deadlock prevention; Deadlock avoidance; Deadlock detection and recovery from deadlock.

6 Hours

PART -B

UNIT -5

MEMORY MANAGEMENT: Memory Management Strategies: Background; Swapping; Contiguous memory allocation; Paging; Structure of page table; Segmentation. Virtual Memory Management: Background; Demand paging; Copy-on-write; Page replacement; Allocation of frames; Thrashing.

7 Hours

UNIT -6 FILE SYSTEM, IMPLEMENTATION OF FILE SYSTEM: File System: File concept; Access methods; Directory structure; File system mounting; File sharing; Protection. Implementing File System: File system structure; File system implementation; Directory implementation; Allocation methods; Free space management.

7 Hours

UNIT-7 SECONDARY STORAGE STRUCTURES, PROTECTION: Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management; Swap space management. Protection: Goals of protection, Principles of protection, Domain of protection, Access matrix, Implementation of access **6 Hours**

UNIT -8 CASE STUDY: THE LINUX OPERATING SYSTEM: Linux history; Design principles; Kernel modules; Process management; Scheduling; Memory management; File systems, Input and output; Inter-process communication. **6Hours**

TEXT BOOK:

1. **Operating System Principles –** Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, 8th edition, Wiley-India, 2009

REFERENCE BOOKS:

1. **Operating Systems: A Concept Based Approach** – D.M Dhamdhere, 2nd Edition, Tata McGraw-Hill, 2002.
2. **Operating Systems** – P.C.P. Bhatt, 2nd Edition, PHI, 2006.
3. **Operating Systems** – Harvey M Deital, 3rd Edition, Addison Wesley, 1990.

Table of Contents

UNIT 1 INTRODUCTION TO OPERATING SYSTEMS, STRUCTURES

1.22 WHAT OPERATING SYSTEM DO.

1.23 COMPUTER SYSTEM ORGANIZATION.

1.24 COMPUTER SYSTEM ARCHITECTURE.

1.25 OPERATING SYSTEM STRUCTURE.

1.26 OPERATING SYSTEM OPERATIONS.

1.27 PROCESS MANAGEMENT.

1.28 MEMORY MANAGEMENT.

1.29 STORAGE MANAGEMENT.

1.30 PROTECTION AND SECURITY.

1 . 3 1 DISTRIBUTED SYSTEM.

1.32 SPECIAL-PURPOSE SYSTEMS.

1.33 COMPUTING ENVIRONMENTS.

1.34 OPERATING SYSTEM SERVICES.

1.35 USER-OPERATING SYSTEM INTERFACE.

1 . 3 6 SYSTEM CALLS, TYPES OF SYSTEM CALLS.

1.37 SYSTEM PROGRAMS.

1.38 OPERATING SYSTEM DESIGN AND IMPLEMENTATION.

1.39 OPERATING SYSTEM STRUCTURE.

1.40 VIRTUAL MACHINES.

1.41 OPERATING SYSTEM GENERATION.

1.42 SYSTEM BOOT.

UNIT -1 INTRODUCTION TO OPERATING SYSTEMS, STRUCTURES

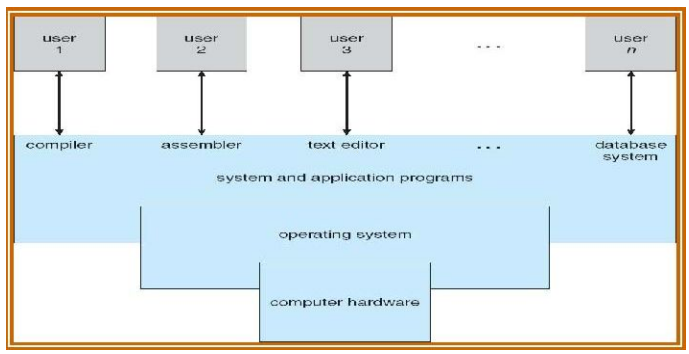
1.1 WHAT OPERATING SYSTEM DO

➓ An OS is an intermediary between the user of the computer & the computer hardware.

- It provides a basis for application program & acts as an intermediary between user of computer & computer hardware.
- ➓ The purpose of an OS is to provide a environment in which the user can execute the program in a convenient & efficient manner.
  - OS is an important part of almost every computer systems.
  - A computer system can be roughly divided into four components
  - The Hardware
  - The OS
  - The application Program
  - The user
- The Hardware consists of memory, CPU, ALU, I/O devices, peripherals devices & storage devices.
- The application program mainly consisted of word processors, spread sheets, compilers & web browsers defines the ways in which the resources are used to solve the problems of the users.
- The OS controls & co-ordinates the use of hardware among various application program for various users.

1.2 COMPUTER SYSTEM ORGANIZATION

The following figure shows the conceptual view of a computer system



Views OF OS

1. User Views:-The user view of the computer depends on the interface used.
   i. Some users may use PC's. In this the system is designed so that only one user can utilize the resources and mostly for ease of use where the attention is mailnly on performances and not on the resource utilization.
   ii. Some users may use a terminal connected to a mainframe or minicomputers.
   iii. Other users may access the same computer through other terminals. These users may share resources and exchange information. In this case the OS is designed to maximize resource utilization-so that all available CPU time, memory & I/O are used efficiently.
   iv. Other users may sit at workstations, connected to the networks of other workstation and servers. In this case OS is designed to compromise between individual visibility & resource utilization.

2. System Views:

i. We can view system as resource allocator i.e. a computer system has many resources that may be used to solve a problem. The OS acts as a manager of these resources. The OS must decide how to allocate these resources to programs and the users so that it can operate the computer system efficiently and fairly.

ii. A different view of an OS is that it need to control various I/O devices & user programs i.e. an OS is a control program used to manage the execution of user program to prevent errors and improper use of the computer.

iii. Resources can be either CPU Time, memory space, file storage space, I/O devices and so on. The OS must support the following tasks

a. Provide the facility to create, modification of programs & data files using on editors.
b. Access to compilers for translating the user program from high level language to machine language.
c. Provide a loader program to move the compiled program code to computers memory for execution.
d. Provides routines that handle the details of I/O programming.

1.3 OPERATING SYSTEM ARCHITECTURE Mainframe System:

a. Mainframe systems are mainly used for scientific & commercial applications.
b. An OS may process its workload serially where the computer runs only one application

or concurrently where computer runs many applications. **Batch Systems**:

a. Early computers where physically large machines.
b. The common I/P devices are card readers & tape drives.
c. The common O/P devices are line printers, tape drives & card punches.
d. The user do not interact directly with computers but we use to prepare a job with the program, data & some control information & submit it to the computer operator.
e. The job was mainly in the form punched cards.
f. At later time the O/P appeared and it consisted of result along with dump of memory and register content for debugging.

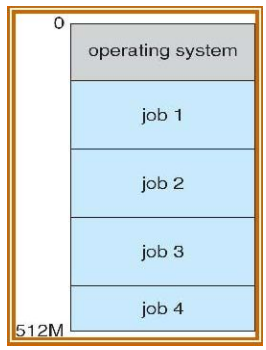
The OS of these computers was very simple. Its major task was to transfer control from one job to the next. The OS was always resident in the memory. The processing of job was very slow. To improve the processing speed operators batched together the jobs with similar needs and processed it through the computers. This is called Batch Systems.

- In batch systems the CPU may be idle for some time because the speed of the mechanical devices slower compared to the electronic devices.
- Later improvement in technology and introduction of disks resulted in faster I/O devices.
- The introduction of disks allowed the OS to store all the jobs on the disk. The OS could perform the scheduling to use the resources and perform the task efficiently.

Disadvantages of Batch Systems:

1. Turn around time can be large from user.
2. Difficult to debug the program.
3. A job can enter into infinite loop.
4. A job could corrupt the monitor.
5. Due to lack of protection scheme, one job may affect the pending jobs.

Multi programmed System:
a. If there are two or more programs in the memory at the same time sharing the processor, this is referred as multi programmed OS.
b. It increases the CPU utilization by organizing the jobs so that the CPU will always have one job to execute.
c. Jobs entering the systems are kept in memory.
d. OS picks the job from memory & it executes it.
e. Having several jobs in the memory at the same time requires some form of memory management.
f. Multi programmed systems monitors the state of all active program and system resources and ensures that CPU is never idle until there are no jobs.
g. While executing a particular job, if the job has to wait for any task like I/O operation to be complete then the CPU will switch to some other jobs and starts executing it and when the first job finishes waiting the CPU will switch back to that.
h. This will keep the CPU & I/O utilization busy. The following figure shows the memory layout of multi programmed OS



Time sharing Systems:
a. Time sharing system or multi tasking is logical extension of multi programming systems. The CPU executes multiple jobs by switching between them but the switching occurs so frequently that user can interact with each program while it is running.
b. An interactive & hands on system provides direct communication between the user and the system. The user can give the instruction to the OS or program directly through key board or mouse and waits for immediate results.
c. A time shared system allows multiple users to use the computer simultaneously. Since each action or commands are short in time shared systems only a small CPU time will be available for each of the user.
d. A time shared systems uses CPU scheduling and multi programming to provide each user a small portion of time shared computers. When a process executes it will be executing for a short time before it finishes or need to perform I/O. I/O is interactive i.e. O/P is to a display for the user and the I/O is from a keyboard, mouse etc.
e. Since it has to maintain several jobs at a time, system should have memory management & protection.
f. Time sharing systems are complex than the multi programmed systems. Since several jobs are kept in memory they need memory management and protection. To obtain less response time jobs are swapped in and out of main memory to disk. So disk will serve as backing store for main memory. This can be achieved by using a technique called virtual memory that allows for the execution of job i.e. not complete in memory.
g. Time sharing system should also provide a file system & file system resides on collection of disks so this need disk management. It supports concurrent execution, job synchronization & communication.

II. DESKTOP SYSTEMS:

- Pc's appeared in 1970's and during this they lacked the feature needed to protect an OS from user program

& they even lack multi user nor multi tasking.

- The goals pf those OS changed later with the time and new systems includes Microsoft Windows & Apple Macintosh.
- The Apple Macintosh OS ported to more advanced hardware & includes new features like virtual memory & multi tasking.
- Micro computers are developed for single user in 1970's & they can accommodate software with large capacity & greater speeds. MS-DOS is an example for micro computer OS & are used by commercial, educational, government enterprises.

III. Multi Processor Systems:

- Multi processor systems include more than one processor in close communication.
- They share computer bus, the clock, m/y & peripheral devices.
- Two processes can run in parallel.
- ➓ Multi processor systems are of two types
- a. Symmetric Multi processors (SMP)
- b. Asymmetric Multi processors.
- In symmetric multi processing, each processors runs an identical copy of OS and they communicate with one another as needed. All the CPU shares the common memory.
- In asymmetric multi processing, each processors is assigned a specific task. It uses a master slave relationship. A master processor controls the system. The master processors schedules and allocates work to slave processors. The following figure shows asymmetric multi processors.
- SMP means al processors are peers i.e. no master slave relationship exists between processors. Each processors concurrently runs a copy of OS.
- The differences between symmetric & asymmetric multi processing may be result of either H/w or S/w. Special H/w can differentiate the multiple processors or the S/w can be written to allow only master & multiple slaves.

Advantages of Multi Processor Systems:

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

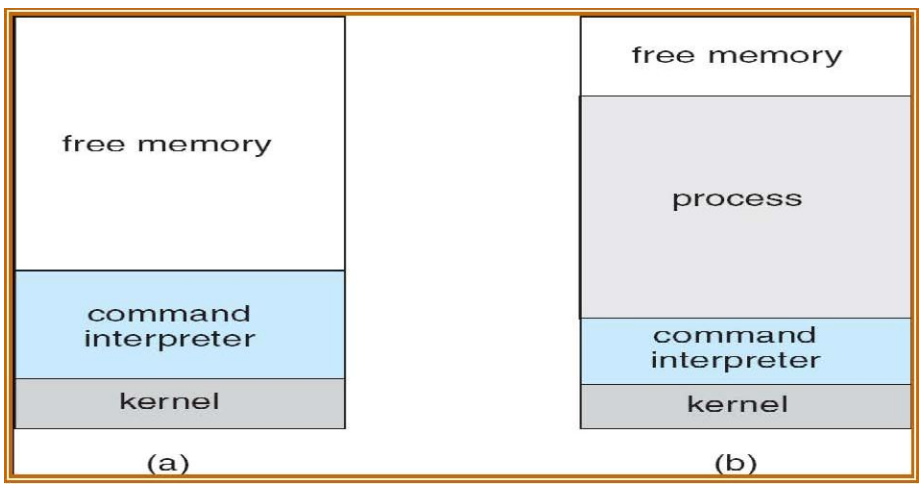
1.4 OPERATING SYSTEM STRUCTURES

PROCESS CONTROL & JOB CONTROL

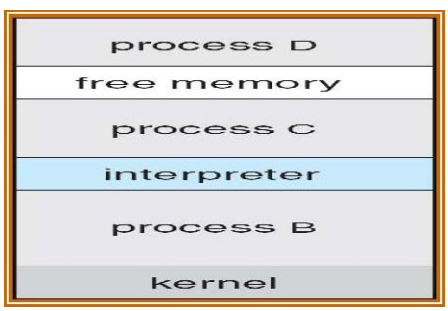
- A system call can be used to terminate the program either normally or abnormally. Reasons for abnormal termination are dump of m/y, error message generated etc.
- Debugger is mainly used to determine problem of the dump & returns back the dump to the OS.
- In normal or abnormal situations the OS must transfer the control to the command interpreter system.
- In batch system the command interpreter terminates the execution of job & continues with the next job.
- Some systems use control cards to indicate the special recovery action to be taken in case of errors.
- Normal & abnormal termination can be combined at some errors level. Error level is defined before & he command interpreter uses this error level to determine next action automatically.

MS-DOS:

MS-DOS is an example of single tasking system, which has command interpreter system i.e. invoked when the computer is started. To run a program MS-DOS uses simple method. It does not create a process when one process is running MS-DOS the program into m/y & gives the program as much as possible. It lacks the general multitasking capabilities.



BSD:Free BSD is an example of multitasking system. In free BSD the command interpreter may continue running while other program is executing. FORK is used to create new process.



1.5 OPERATING SYSTEM OPERATIONS

Modern OS supports all system components. The system components are,
1. Process Management.
2. Main M/y Management.
3. File Management.
4. Secondary Storage Management.
5. I/O System management.
6. Networking.
7. Protection System.
8. Command Interpreter System.

1.6 PROCESS MANAGEMENT

- A process is a program in execution.
- A process abstraction is a fundamental OS mechanism for the management of concurrent program execution.
- The OS responds by creating process.
- Process requires certain resources like CPU time, M/y, I/O devices. These resources are allocated to the process when it created or while it is running.
- When process terminates the process reclaims all the reusable resources.
- Process refers to the execution of M/c instructions.
- A program by itself is not a process but is a passive entity.

The OS is responsible for the following activities of the process management,
- Creating & destroying of the user & system process .
- Allocating H/w resources among the processes.
- Controlling the progress of the process.
- Provides mechanism for process communication.
- Provides mechanism for deadlock handling.

1.7 MEMORY MANAGEMENT

- Main M/y is the centre to the operation of the modern computer.
- Main M/y is the array of bytes ranging from hundreds of thousands to billions. Each byte will have their own address.
- The central processor reads the instruction from main M/y during instruction fetch cycle & it both reads & writes the data during the data-fetch cycle. The I/O operation reads and writes data in main M/y.
- The main M/y is generally a large storage device in which a CPU can address & access directly.
- When a program is to be executed it must be loaded into memory & mapped to absolute address. When it is executing it access the data & instruction from M/y by generating absolute address. When the program terminates all available M/y will be returned back.
- To improve the utilization of CPU & the response time several program will be kept in M/y.
- ⓾ Several M/y management scheme are available & selection depends on the H/w design of the
- system. The OS is responsible for the following activities.
- Keeping track of which part of the M/y is used & by whom.
- Deciding which process are to be loaded into M/y.
- Allocating & de allocating M/y space as needed.

File Management:

- File management is one of the most visible component of an OS.
- Computer stores data on different types of physical media like Magnetic Disks, Magnetic tapes, optical disks etc.
- For convenient use of the computer system the OS provides uniform logical view of information storage.
- The OS maps file on to physical media & access these files via storage devices.
- A file is logical collection of information.
- File consists of both program & data. Data files may be numeric, alphabets or alphanumeric.
- Files can be organized into directories. The OS is responsible for the following activities,
- Creating & deleting of files.
- Creating & deleting directories.
- Supporting primitives for manipulating files & directories.
- Maping files onto secondary storage.
- Backing up files on stable storage media.

1.8 STORAGE MANAGEMENT

- Is a mechanism where the computer system may store information in a way that it can be retrieved later.
- They are used to store both data & programs.
- The programs & data are stored in main memory.
- Since the size of the M/y is small & volatile Secondary storage devices is used.
- ➓ Magnetic disk is central importance of computer system. The OS is responsible for the following activities,

- Free space management.

- Storage allocation.

- Disk scheduling. The entire speed of computer system depends on the speed of the disk sub system.

I/O System Management:

- Each I/o device has a device handler that resides in separate process associated with that device. The I/O management consists of,
- A M/y management component that include buffering,, caching & spooling.
- General device-driver interface.
- Drivers for specific H/w device.

Networking :

- Networking enables users to share resources & speed up computations.
- ➓ The process communicates with one another through various communication lines like high
- speed buses or N/w. Following parameters are considered while designing the N/w,
- Topology of N/w.
- Type of N/w.
- Physical media.
- Communication protocol,
- Routing algorithms.

1.9 PROTECTION AND SECURITY

- Modern computer system supports many users & allows the concurrent execution of multiple processes organization rely on computers to store information. It necessary that the information & devices must be protected from unauthorized users or processors.
- The protection is a mechanism for controlling the access of program, processes or users to the resources defined by a computer system.
- Protection mechanism are implemented in OS to support various security policies.
- The goal of security system is to authenticate their access to any object.
- Protection can improve reliability by detecting latent errors at the interface B/w component sub system.
- Protection domains are extensions of H/w supervisor mode ability.

1.10 DISTRIBUTED SYSTEMS

- A distributed system is one in which H/w or S/w components located at the networked computers communicate & co ordinate their actions only by passing messages.
- A distributed systems looks to its user like an ordinary OS but runs on multiple, Independent CPU's.
- Distributed systems depends on networking for their functionality which allows for communication so that distributed systems are able to share computational tasks and provides rich set of features to users.
- N/w may vary by the protocols used, distance between nodes & transport media. Protocols->TCP/IP, ATM etc. Network-> LAN, MAN, WAN etc. Transport Media-> copper wires, optical fibers & wireless transmissions

Client-Server Systems:

- Since PC's are faster, power full, cheaper etc. designers have shifted away from the centralized system architecture.
- User-interface functionality that used to be handled by centralized system is handled by PC's.

So the centralized system today act as server program to satisfy the requests of client. Server system can be classified as follows

c. Computer-Server System:-Provides an interface to which client can send requests to perform some actions, in response to which they execute the action and send back result to the client.

d. File-Server Systems:-Provides a file system interface where clients can create, update, read & delete files.

Peer-to-Peer Systems:

- PC's are introduced in 1970's they are considered as standalone computers i.e. only one user can use it at a time.
- With wide spread use of internet PC's were connected to computer networks.
- With the introduction of the web in mid 1990's N/w connectivity became an essential component of a computer system.
- All modern PC's & workstation can run a web. Os also includes system software that enables the computer to access the web.
- In distributed systems or loosely coupled couple systems, the processor can communicate with one another through various communication lines like high speed buses or telephones lines.
- A N/w OS which has taken the concept of N/w & distributed system which provides features fir file sharing across the N/w and also provides communication which allows different processors on different computers to share resources.

Advantages of Distributed Systems:

1. Resource sharing. 2. Higher reliability. 3.Better price performance ratio. 4.Shorter response time. 5.Higher throughput. 6.Incremental growth

1.11 SPECIAL-PURPOSE SYSTEMS. Clustered Systems

- Like parallel systems the clustered systems will have multiple CPU but they are composed of two or more individual system coupled together.
- Clustered systems share storage & closely linked via LAN N/w.
- Clustering is usually done to provide high availability.
- Clustered systems are integrated with H/w & S/w. H/w clusters means sharing of high performance disk. S/w clusters are in the form of unified control of a computer system in a cluster.
- A layer of S/w cluster runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored M/c fails the monitoring M/c take ownership of its storage and restart the application that were running on failed M/c.
- ❿ Clustered systems can be categorized into two groups
  - Asymmetric Clustering &
  - Symmetric clustering
- In asymmetric clustering one M/c is in hot standby mode while others are running the application. The hot standby M/c does nothing but it monitors the active server. If the server fails the hot standby M/c becomes the active server.
- In symmetric mode two or more hosts are running the Application & they monitor each other. This mode is more efficient since it uses all the available H/w.
- Parallel clustering and clustering over a LAN is also available in clustering. Parallel clustering allows multiple hosts to access the same data on shared storage.
- Clustering provides better reliability than the multi processor systems.
- It provides all the key advantages of a distributed systems.
- Clustering technology is changing & include global clusters in which M/c could be anywhere in the world.

Real-Time Systems

Real time system is one which were originally used to control autonomous systems like satellites, robots, hydroelectric dams etc.

- Real time system is one that must react to I/p & responds to them quickly.
- A real time system should not be late in response to one event.
- ❿ A real time should have well defined time constraints.
  - Real time systems are of two types
  - Hard Real Time Systems
  - Soft Real Time Systems
- A hard real time system guarantees that the critical tasks to be completed on time. This goal requires that all delays in the system be bounded from the retrieval of stored data to time that it takes the OS to finish the request.
- In soft real time system is a less restrictive one where a critical real time task gets priority over other tasks & retains the property until it completes. Soft real time system is achievable goal that can be mixed with other type of systems. They have limited utility than hard real time systems.
- Soft real time systems are used in area of multimedia, virtual reality & advanced scientific projects. It cannot be used in robotics or industrial controls due to lack of deadline support.
- Real time OS uses priority scheduling algorithm to meet the response requirement of a real time application.
- Soft real time requires two conditions to implement, CPU scheduling must be priority based & dispatch latency should be small.
- The primary objective of file management in real time systems is usually speed of access, rather than efficient utilization of secondary storage.

1.12 COMPUTING ENVIRONMENTS

Different types of computing environments are:

- Traditional Computing.
- Web Based Computing.
- Embedded Computing.

- **Traditional Computing** Typical office environment uses traditional computing. Normal PC is used in traditional computing environment. N/w computers are essential terminals that understand web based computing. In domestic application most of the user had a single computer with internet connection. Cost of accessing internet is high.
- **Web Based Computing** has increased the emphasis on N/w. Web based computing uses PC, handheld PDA & cell phones. One of the feature of this type is load balancing. In load balancing, N/w connection is distributed among a pool of similar servers.
- **Embedded computing** uses real time OS. Application of embedded computing is car engines, manufacturing robots, microwave ovens. This type of system provides limited features.

1.13 OPERATING SYSTEM SERVICES:

An OS provides services for the execution of the programs and the users of such programs. The services provided by one OS may be different from other OS. OS makes the programming task easier. The common services provided by the OS are

1. Program Execution:-The OS must able to load the program into memory & run that program. The program must end its execution either normally or abnormally.
2. I/O Operation:-A program running may require any I/O. This I/O may be a file or a specific device users cant control the I/O device directly so the OS must provide a means for controlling I/O devices.
3. File System Interface:-Program need to read or write a file. The OS should provide permission for the creation or deletion of files by names.
4. Communication:-In certain situation one process may need to exchange information with another process. This communication May takes place in two ways.

    a. Between the processes executing on the same computer.

    b. Between the processes executing on different computer that are connected by a network. This communication can be implemented via shared memory or by OS.

5. Error Detection:-Errors may occur in CPU, I/O devices or in M/y H/w. The OS constantly needs to be aware of possible errors. For each type of errors the OS should take appropriate actions to ensure correct & consistent computing.

OS with multiple users provides the following services,

a. Resource Allocation:-When multiple users logs onto the system or when multiple jobs are running, resources must be allocated to each of them. The OS manages different types of OS resources. Some resources may need some special allocation codes & others may have some general request & release code.

b. Accounting:-We need to keep track of which users use how many & what kind of resources. This record keeping may be used for accounting. This accounting data may be used for statistics or billing. It can also be used to improve system efficiency.

c. Protection:-Protection ensures that all the access to the system are controlled. Security starts with each user having authenticated to the system, usually by means of a password. External I/O devices must also be protected from invalid access. In multi process environment it is possible that one process may interface with the other or with the OS, so protection is required.

1.14 USER OPERATING SYSTEM INTERFACE Command Interpreter System

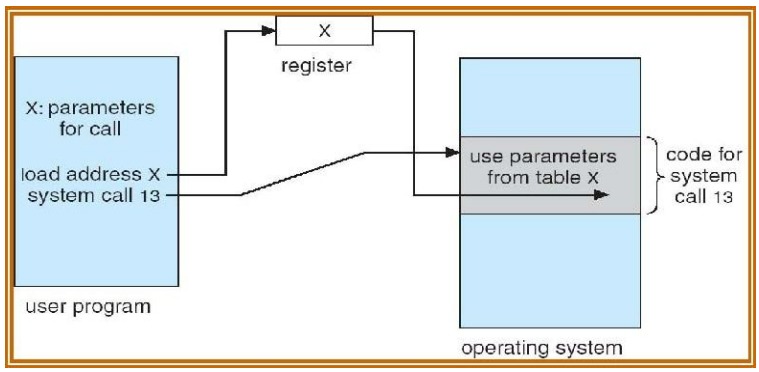
- Command interpreter system between the user & the OS. It is a system program to the OS.
- Command interpreter is a special program in UNIX & MS DOS OS i.e. running when the user logs on.
- Many commands are given to the OS through control statements when the user logs on, a program that reads & interprets control statements is executed automatically. This program is sometimes called the control card interpreter or command line interpreter and is also called as shell.
- The command statements themselves deal with process creation & management, I/O handling, secondary storage management, main memory management, file system access, protection & N/w.

1.15 SYSTEM CALLS

- System provides interface between the process & the OS.
- The calls are generally available as assembly language instruction & certain system allow system calls to be made directly from a high level language program.
- Several language have been defined to replace assembly language program.
- A system call instruction generates an interrupt and allows OS to gain control of the processors.
- System calls occur in different ways depending on the computer. Some time more information is needed to identify the desired system call. The exact type & amount of information needed may vary according to the particular OS & call.

TYPES OF SYSTEM CALLS

PASSING PARAMETERS TO OS



Three general methods are used to pass the parameters to the OS.

- The simplest approach is to pass the parameters in registers. In some there can be more parameters than register. In these the parameters are generally in a block or table in m/y and the address of the block is passed as parameters in register. This approach used by Linux.
- Parameters can also be placed or pushed onto stack by the program & popped off the stack by the OS.
- Some OS prefer the block or stack methods, because those approaches do not limit the number or length of parameters being passed.
- ➓ System calls may be grouped roughly into 5 categories
  - Process control.
  - File management.
  - Device management.
  - Information maintenance.
  - Communication.

1.16 SYSTEM PROGRAMS

- Many system calls are used to transfer information between user program & OS. Example:-Most systems

have the system calls to return the current time & date, number of current users, version number of OS, amount of free m/y or disk space & so on.

- In addition the OS keeps information about all its processes & there are system calls to access

this information. **COMMUNICATION:-There are two modes of communication,**

1. Message Passing Models:

- In this information is exchanged using inter-process communication facility provided by OS.
- Before communication the connection should be opened.
- The name of the other communicating party should be known, it ca be on the same computer or it can be on another computer connected by a computer network.
- Each computer in a network may have a host name like IP name similarly each process can have a process name which can be translated into equivalent identifier by OS.
- The get host id & process id system call do this translation. These identifiers are then passed to the open & close connection system calls.
- The recipient process must give its permission for communication to take place with an accept connection call.
- Most processes receive the connection through special purpose system program dedicated for that purpose called daemons. The daemon on the server side is called server daemon & the daemon on the client side is called client daemon.

2. Shared Memory:

- In this the processes uses the map m/y system calls to gain access to m/y owned by another process.
- The OS tries to prevent one process from accessing another process m/y.
- In shared m/y this restriction is eliminated and they exchange information by reading and writing data in shared areas. These areas are located by these processes and not under OS control.
- They should ensure that they are not writing to same m/y area.
- Both these types are commonly used in OS and some even implement both.
- Message passing is useful when small number of data need to be exchanged since no conflicts are to be avoided and it is easier to implement than in shared m/y. Shared m/y allows maximum speed and convenience of communication as it is done at m/y speed when within a computer.

1.17 OPERATING SYSTEM DESIGN AND IMPLEMENTATION FILE MANAGEMENT

- System calls can be used to create & deleting of files. System calls may require the name of the files with attributes for creating & deleting of files.
- Other operation may involve the reading of the file, write & reposition the file after it is opened.
- Finally we need to close the file.
- For directories some set of operation are to be performed. Sometimes we require to reset some of the attributes on files & directories. The system call get file attribute & set file attribute are used for this type of operation.

DEVICE MANAGEMENT:

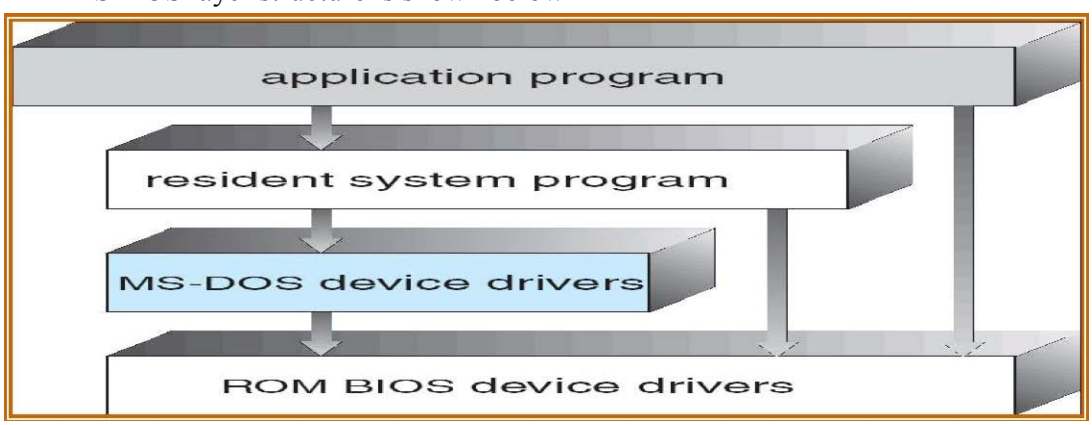
- The system calls are also used for accessing devices.
- Many of the system calls used for files are also used for devices.
- In multi user environment the requirement are made to use the device. After using the device must be released using release system call the device is free to be used by another user. These function are similar to open & close system calls of files.
- Read, write & reposition system calls may be used with devices.
- MS-DOS & UNIX merge the I/O devices & the files to form file services structure. In file device structure I/O devices are identified by file names.

1.18 OPERATING SYSTEM STRUCTURES

- Modern OS is large & complex.
- OS consists of different types of components.
- These components are interconnected & melded into kernel.
- For designing the system different types of structures are used. They are,
- Simple structures.
- Layered structured.
- Micro kernels

Simple Structures

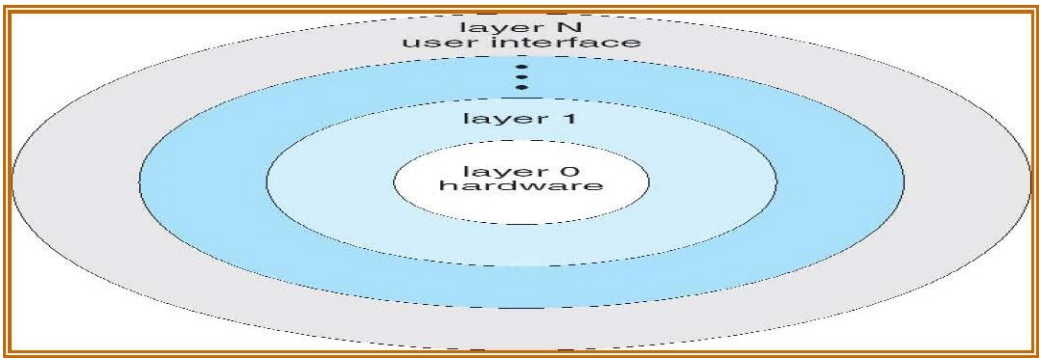
- Simple structure OS are small, simple & limited systems.
- The structure is not well defined
- MS-DOS is an example of simple structure OS.
- MS-DOS layer structure is shown below



- ⑩ UNIX consisted of two separate modules

- a. Kernel
- b. The system programs.
- Kernel is further separated into series of interfaces & device drivers which were added & expanded as the UNIX evolved over years.
- The kernel also provides the CPU scheduling, file system, m/y management & other OS function through system calls.
- System calls define API to UNIX and system programs commonly available defines the user interface. The programmer and the user interface determines the context that the kernel must support.
- New versions of UNIX are designed to support more advanced H/w. the OS can be broken down into large number of smaller components which are more appropriate than the original MS-DOS.

Layered Approach



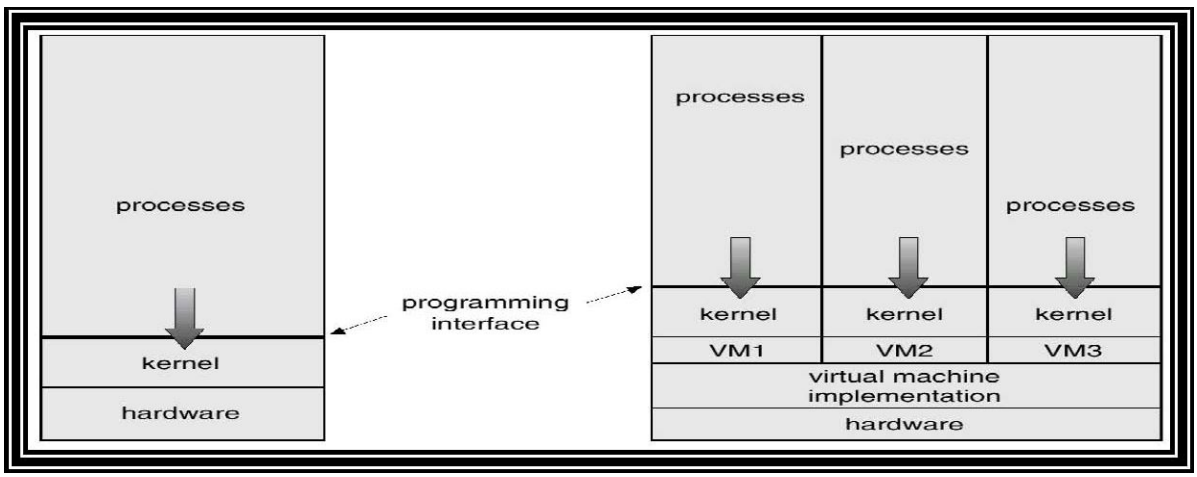
- In this OS is divided into number of layers, where one layer is built on the top of another layer. The bottom layer is hardware and higher layer is the user interface.
- An OS is an implementation of abstract object i.e. the encapsulation of data & operation to manipulate these data.
- The main advantage of layered approach is the modularity i.e. each layer uses the services & functions provided by the lower layer. This approach simplifies the debugging & verification. Once first layer is debugged the correct functionality is guaranteed while debugging the second layer. If an error is identified then it is a problem in that layer because the layer below it is already debugged.
- Each layer is designed with only the operations provided by the lower level layers.
- Each layer tries to hide some data structures, operations & hardware from the higher level layers.
- A problem with layered implementation is that they are less efficient then the other types.

Micro Kernels

- Micro kernel is a small Os which provides the foundation for modular extensions.
- The main function of the micro kernels is to provide communication facilities between the current program and various services that are running in user space.
- This approach was supposed to provide a high degree of flexibility and modularity.
- This benefits of this approach includes the ease of extending OS. All the new services are added to the user space & do not need the modification of kernel.
- This approach also provides more security & reliability.
- Most of the services will be running as user process rather than the kernel process.
- This was popularized by use in Mach OS.
- Micro kernels in Windows NT provides portability and modularity. Kernel is surrounded by a number of compact sub systems so that task of implementing NT on variety of platform is easy.
- Micro kernel architecture assign only a few essential functions to the kernel including address space, IPC & basic scheduling.
- QNX is the RTOS i.e. also based on micro kernel design.

1.19 VIRTUAL MACHINES

A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface identical to the underlying bare hardware. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory. The resources of the physical computer are shared to create the virtual machines. CPU scheduling can create the appearance that users have their own processor. Spooling and a file system can provide virtual card readers and virtual line printers. A normal user time-sharing terminal serves as the virtual machine operator's console.



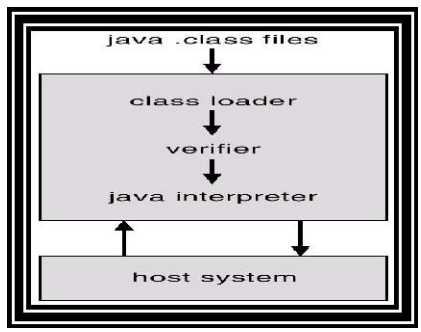
NON-VIRTUAL MACHINE VIRTUALMACHINE

Advantages and Disadvantages of Virtual Machines

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- JVM consists of -class loader -class verifier -runtime interpreter
- Just-In-Time (JIT) compilers increase performance



JAVA VIRTUAL MACHINE

1.20 OPERATING SYSTEM GENERATION;

➓ User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast.

o System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

➓ Mechanisms determine how to do something, policies decide what will be done.

o The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- ➓ Code written in a high-level language:
  - o can be written faster.
  - o is more compact.
  - o is easier to understand and debug.
- An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.

1.21 SYSTEM BOOT

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
- *Booting* – starting a computer by loading the kernel.
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

IMPORTANT QUESTIONS

1. What are the three main purposes of an operating system?
2. What is the main advantage of multiprogramming?

3. What are the main differences between operating systems for mainframe computers and personal computers?
4. Define the essential properties of the following types of operating systems:

a. Batch
b. Interactive
c. Time sharing
d. Real time
e. Network
f. Distributed

一. What are the differences between a trap and an interrupt? What is the use of each function?
一. What are the five major activities of an operating system in regard to process management?
一. What are the three major activities of an operating system in regard to secondary-storage management?
一. List five services provided by an operating system.
一. What is the main advantage of the layered approach to system design?
一. 10. What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?

UNIT 2 PROCESS MANAGEMENT

TOPICS

2.13 PROCESS CONCEPT.
2.14 PROCESS SCHEDULING.
2.15 OPERATIONS ON PROCESSES.
2.16 INTER-PROCESS COMMUNICATION.
2.17 MULTI-THREADED PROGRAMMING.
2.18 OVERVIEW; MULTITHREADING MODELS.
2.19 THREAD LIBRARIES; THREADING ISSUES.
2.20 PROCESS SCHEDULING: BASIC CONCEPTS.
2.21 SCHEDULING CRITERIA.
2.22 SCHEDULING ALGORITHMS.
2.23 THREAD SCHEDULING.
2.24 MULTIPLE-PROCESSOR SCHEDULING.

2.1 PROCESS CONCEPTS

Processes & Programs:

- Process is a dynamic entity. A process is a sequence of instruction execution process exists in a limited span of time. Two or more process may execute the same program by using its own data & resources.
- A program is a static entity which is made up of program statement. Program contains the instruction. A program exists in a single space. A program does not execute by itself.
- A process generally consists of a process stack which consists of temporary data & data section which consists of global variables.
- It also contains program counter which represents the current activities.
- A process is more than the program code which is also called text section.

Process State:

The process state consist of everything necessary to resume the process execution if it is somehow put aside temporarily. The process state consists of at least following:

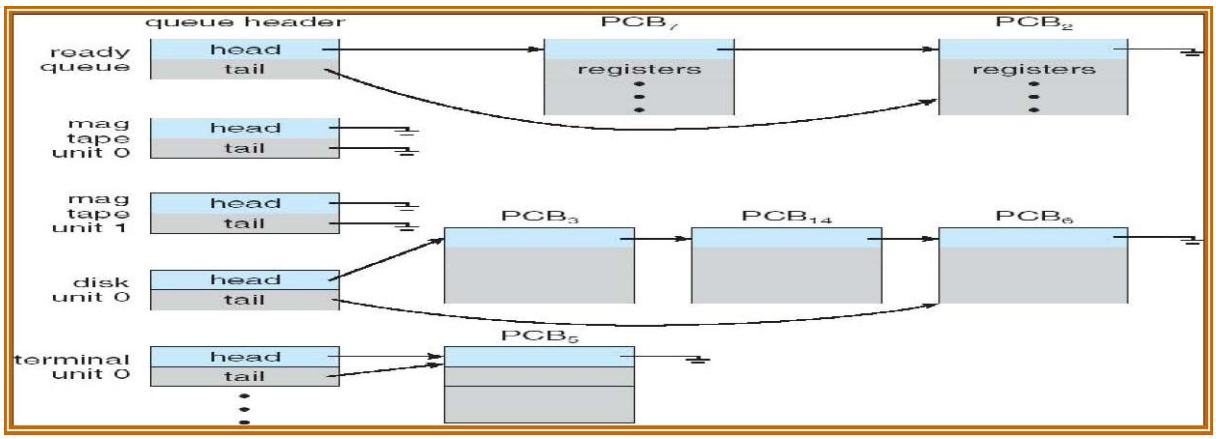
- Code for the program.
- Program's static data.
- Program's dynamic data.
- Program's procedure call stack.
- Contents of general purpose registers.
- Contents of program counter (PC)
- Contents of program status word (PSW).
- Operating Systems resource in use.

2.2 PROCESS SCHEDULING

PROCESS SCHEDULING QUEUES

The following are the different types of process scheduling queues.

1. Job queue – set of all processes in the system
2. Ready queue – set of all processes residing in main memory, ready and waiting to execute
3. Device queues – set of processes waiting for an I/O device
4. Processes migrate among the various queues



Ready Queue And Various I/O Device Queues

Ready Queue:

The process that are placed in main m/y and are already and waiting to executes are placed in a list called the ready queue. This is in the form of linked list. Ready queue header contains pointer to the first & final PCB in the list. Each PCB contains a pointer field that points next PCB in ready queue.

Device Queue:The list of processes waiting for a particular I/O device is called device. When the CPU is allocated to a process it may execute for some time & may quit or interrupted or wait for the occurrence of a particular event like completion of an I/O request but the I/O may be busy with some other processes. In this case the process must wait for I/O. This will be placed in device queue. Each device will have its own queue.

The process scheduling is represented using a queuing diagram. Queues are represented by the rectangular box & resources they need are represented by circles. It contains two queues ready queue & device queues. Once the process is assigned to CPU and is executing the following events can occur,

1.20 It can execute an I/O request and is placed in I/O queue.

1.21 The process can create a sub process & wait for its termination.

1.22 The process may be removed from the CPU as a result of interrupt and can be put back into ready queue.

Schedulers:

The following are the different type of schedulers

1. **Long-term scheduler (or job scheduler) –** selects which processes should be brought into the ready queue.
2. **Short-term scheduler (or CPU scheduler) –** selects which process should be executed next and allocates CPU.
3. **Medium-term schedulers**

-> Short-term scheduler is invoked very frequently (milliseconds) . (must be fast)

-> Long-term scheduler is invoked very infrequently (seconds, minutes) (may be slow)

-> The long-term scheduler controls the *degree of multiprogramming*

->Processes can be described as either:

x I/O-bound process – spends more time doing I/O than computations, many short CPU bursts

CPU-bound process – spends more time doing computations; few very long CPU bursts

2.3 OPERATION ON PROCESS

Process Creation

In general-purpose systems, some way is needed to create processes as needed during operation. There are four principal events led to processes creation.

x System initialization.
x Execution of a process Creation System calls by a running process.
x A user request to create a new process.
x Initialization of a batch job.

Foreground processes interact with users. Background processes that stay in background sleeping but suddenly springing to life to handle activity such as email, webpage, printing, and so on. Background processes are called daemons. This call creates an exact clone of the calling process. A process may create a new process by some create process such as 'fork'. It choose to does so, creating process is called parent process and the created one is called the child processes. Only one parent is needed to create a child process. Note that unlike plants and animals that use sexual representation, a process has only one parent. This creation of process (processes) yields a hierarchical structure of processes like one in the figure. Notice that each child has only one parent but each parent may have many children. After the fork, the two processes, the parent and the child, have the same memory image, the same environment strings and the same open files. After a process is created, both the parent and child have their own distinct address space. If either process changes a word in its address space, the change is not visible to the other process.

Following are some reasons for creation of a process

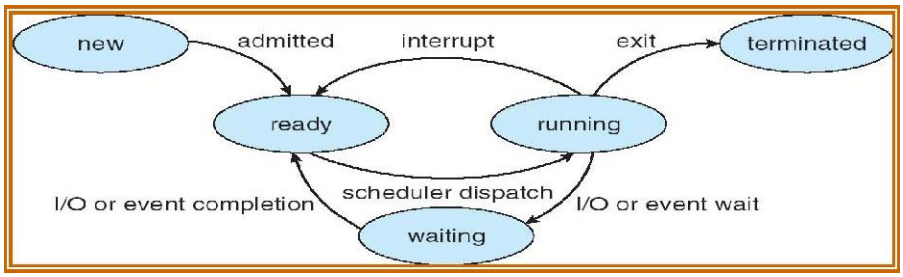
x User logs on.
x User starts a program.
x Operating systems creates process to provide service, e.g., to manage printer.
x Some program starts another process, e.g., Netscape calls *xv* to display a picture.

Process Termination

A process terminates when it finishes executing its last statement. Its resources are returned to the system, it is purged from any system lists or tables, and its process control block (PCB) is erased i.e., the PCB's memory space is returned to a free memory pool. The new process terminates the existing process, usually due to following reasons:

x **Normal Exist** Most processes terminates because they have done their job. This call is exist in UNIX. x **Error Exist** When process discovers a fatal error. For example, a user tries to compile a program that does not exist. x **Fatal Error** An error caused by process due to a bug in program for example, executing an illegal instruction, referring non-existing memory or dividing by zero. x **Killed by another Process** A process executes a system call telling the Operating Systems to terminate some other process. In UNIX, this call is kill. In x some systems when a process kills all processes it created are killed as well (UNIX does not work this way).

Process States :A process goes through a series of discrete process states.



x **New State** The process being created. x **Terminated State** The process has finished execution. x **Blocked (waiting) State** When a process blocks, it does so because logically it cannot continue, typically because it is waiting for input that is not yet available. Formally, a process is said to be blocked if it is waiting for some event to happen (such as an I/O completion) before it can proceed. In this state a process is unable to run until some external event happens.

x **Running State** A process is said t be running if it currently has the CPU, that is, actually using the CPU at that particular instant. x **Ready State** A process is said to be ready if it use a CPU if one were available. It is runable but temporarily stopped to let another process run.

Logically, the 'Running' and 'Ready' states are similar. In both cases the process is willing to run, only in the case of 'Ready' state, there is temporarily no CPU available for it. The 'Blocked' state is different from the 'Running' and 'Ready' states in that the process cannot run, even if the CPU is available.

Process Control Block

A process in an operating system is represented by a data structure known as a process control block (PCB) or process descriptor. The PCB contains important information about the specific process including x The current state of the process i.e., whether it is ready, running, waiting, or whatever.

- Unique identification of the process in order to track "which is which" information.
- A pointer to parent process.
- Similarly, a pointer to child process (if it exists).
- The priority of process (a part of CPU scheduling information).
- Pointers to locate memory of processes.
- A register save area.
- The processor it is running on.

The PCB is a certain store that allows the operating systems to locate key information about a process. Thus, the PCB is the data structure that defines a process to the operating systems.

The following figure shows the process control block.

Context Switch:

1. When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
2. Context-switch time is overhead; the system does no useful work while switching.
3. Time dependent on hardware support

Cooperating Processes & Independent Processes

Independent process: one that is independent of the rest of the universe.

- Its state is not shared in any way by any other process.
- Deterministic: input state alone determines results.
- Reproducible.
- Can stop and restart with no bad effects (only time varies). Example: program that sums the integers from 1 to i (input).

There are many different ways in which a collection of independent processes might be executed on a processor:

x programming: a single process is run to completion before anything else can be run on the processor.

- Multiprogramming: share one processor among several processes. If no shared state, then order of dispatching is irrelevant.
- Multiprocessing: if multiprogramming works, then it should also be ok to run processes in parallel on separate processors.

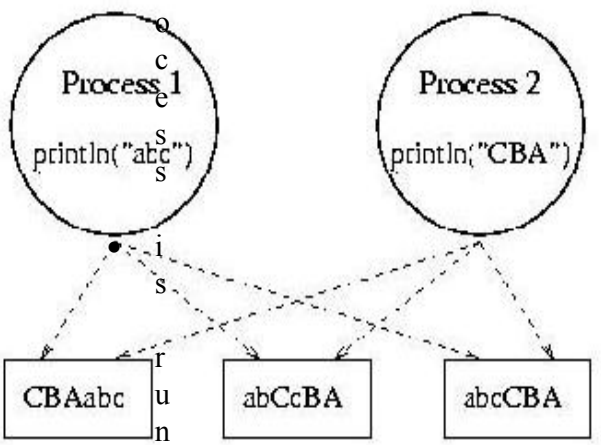
Unipro
. o A given process runs on only one processor at a time.
. o A process may run on different processors at different times (move state, assume processors are identical).
. o Cannot distinguish multiprocessing from multiprogramming on a very fine grain.

Cooperating processes:

x Machine must model the social structures of the people that use it. People cooperate, so machine must support that cooperation. Cooperation means shared state, e.g. a single file system. x Cooperating processes are those that share state. (May or may not actually be "cooperating") x Behavior is nondeterministic: depends on relative execution sequence and cannot be predicted a priori. x Behavior is irreproducible. x Example: one process writes "ABC", another writes "CBA". Can get different outputs, cannot tell what comes from which. E.g. which process output first "C" in "ABCCBA"? Note the subtle state sharing that occurs here via the terminal. Not just anything can happen, though. For example, "AABBCC" cannot occur.

1. Independent process cannot affect or be affected by the execution of another process
2. Cooperating process can affect or be affected by the execution of another process
3. Advantages of process cooperation

- Information sharing
- Computation speed-up
- Modularity
- Convenience



Processis run to Un

2.4 INTERPROCESS COMMUNICATION (IPC)

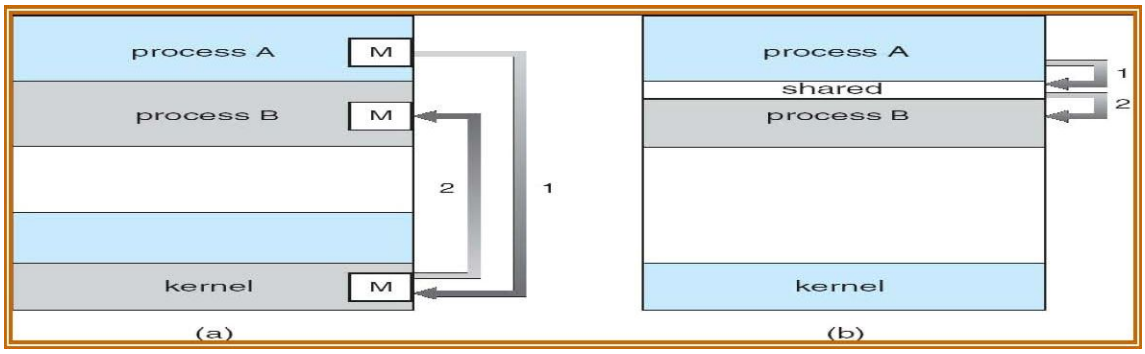
1. Mechanism for processes to communicate and to synchronize their actions.
2. Message system – processes communicate with each other without resorting to shared variables
3. IPC facility provides two operations:
   - send(*message*) – message size fixed or variable
   - receive(*message*)
4. If *P* and *Q* wish to communicate, they need to:exchange messages via send/receive
5. Implementation of communication link

physical (e.g., shared memory, hardware bus)
logical (e.g., logical properties)

Communications Models

there are two types of communication models

1. Multi programming
2. Shared Memory



Direct Communication

1. Processes must name each other explicitly:

x send (*P, message*) – send a message to process P x receive(*Q, message*) – receive a message from process Q

2. Properties of communication link

x Links are established automatically x A link is associated with exactly one pair of communicating processes x Between each pair there exists exactly one link x The link may be unidirectional, but is usually bi-directional

Indirect Communication

1. Messages are directed and received from mailboxes (also referred to as ports)

x Each mailbox has a unique id

x Processes can communicate only if they share a mailbox

2. Properties of communication link

x Link established only if processes share a common mailbox

x A link may be associated with many processes

x Each pair of processes may share several communication links x Link may be unidirectional or bi-directional

3**.** Operations

. o create a new mailbox
. o send and receive messages through mailbox
. o destroy a mailbox

4.Primitives are defined as: send(*A, message*) – send a message to mailbox A receive(*A, message*) – receive a message from mailbox A

5.Mailbox sharing *P1, P2,* and *P3* share mailbox A *P1*, sends; *P2* and *P3* receive Who gets the message?

6. Solutions Allow a link to be associated with at most two processes

Allow only one process at a time to execute a receive operation Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

1. Message passing may be either blocking or non-blocking
2. Blocking is considered synchronous

->Blocking send has the sender block until the message is received. ->Blocking receive has the receiver block until a message is available.

3. Non-blocking is considered asynchronous

->Non-blocking send has the sender send the message and continue.

->Non-blocking receive has the receiver receive a valid message or null.

Buffering

->Queue of messages attached to the link; implemented in one of three ways

1. Zero capacity – 0 messages sender must wait for receiver (rendezvous)
2. Bounded capacity – finite length of *n* messages Sender must wait if link full

3. Unbounded capacity – infinite length sender never waits

2.5 MULTI THREADED PROGRAMMING

Despite of the fact that a thread must execute in process, the process and its associated threads are different concept. Processes are used to group resources together and threads are the entities scheduled for execution on the CPU. *A thread is a single sequence stream within in a process*. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*. In a process, threads allow multiple executions of streams. In many respect, threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. Like a traditional process i.e., process with one thread, a thread can be in any of several states (Running, Blocked, Ready or Terminated). Each thread has its own stack. Since thread will generally call different procedures and thus a different execution history. This is why thread needs its own stack. An operating system that has thread facility, the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section, OS resources also known as task, such as open files and signals.

Processes Vs Threads

As we mentioned earlier that in many respect threads operate in the same way as that of processes. Some of the similarities and differences are:

Similarities

- Like processes threads share CPU and only one thread active (running) at a time.
- Like processes, threads within a processes, threads within a processes execute sequentially.
- Like processes, thread can create children.
- And like process, if one thread is blocked, another thread can run.

Differences

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task .
- Unlike processes, thread are design to assist one other. Note that processes might or might not assist one another because processes may originate from different users.

Why Threads?

Following are some reasons why we use threads in designing operating systems.

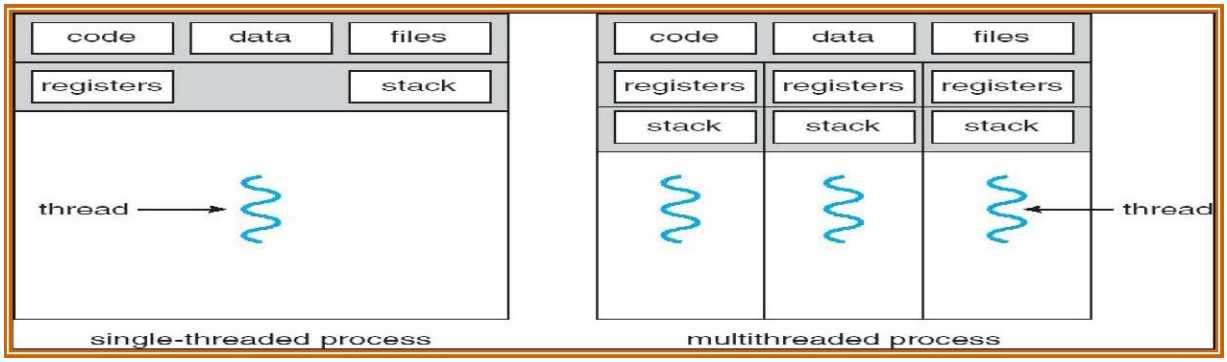
1. A process with multiple threads make a great server for example printer server.
2. Because threads can share common data, they do not need to use interprocess communication.
3. Because of the very nature, threads can take advantage of multiprocessors.
4. Responsiveness
5. Resource Sharing
6. Economy
7. Utilization of MP Architectures

Threads are cheap in the sense that

1. They only need a stack and storage for registers therefore, threads are cheap to create.
2. Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources.
3. Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

But this cheapness does not come free -the biggest drawback is that there is no protection between threads.

Single and Multithreaded Processes



User-Level Threads

1. Thread management done by user-level threads library

Three primary thread libraries: -> POSIX Pthreads
-> Win32 threads

-> Java threads

User-level threads implement in user-level libraries, rather than via systems calls, so thread switching does not need to call operating system and to cause interrupt to the kernel. In fact, the kernel knows nothing about user-level threads and manages them as if they were single-threaded processes.

Advantages:

The most obvious advantage of this technique is that a user-level threads package can be implemented on an Operating System that does not support threads. Some other advantages are

x User-level threads does not require modification to operating systems. x Simple representation: Each thread is represented simply by a PC, registers, stack and a small control block, all stored in the user process address space. x Simple Management: This simply means that creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel. x Fast and Efficient: Thread switching is not much more expensive than a procedure call.

Disadvantages:

x There is a lack of coordination between threads and operating system kernel. Therefore, process as whole gets one time slice irrespect of whether process has one thread or 1000 threads within. It is up to each thread to relinquish control to other threads.
x User-level threads requires non-blocking systems call i.e., a multithreaded kernel. Otherwise, entire process will blocked in the kernel, even if there are runable threads left in the processes. For example, if one thread causes a page fault, the process blocks.

Kernel-Level Threads

1. Supported by the Kernel

Examples: ->Windows XP/2000, ->Solaris , ->Linux, ->Tru64 UNIX, ->Mac OS X

In this method, the kernel knows about and manages the threads. No runtime system is needed in this case. Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system. In addition, the kernel also maintains the traditional process table to keep track of processes. Operating Systems kernel provides system call to create and manage threads.

Advantages:

x Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads. x

Kernel-level threads are especially good for applications that frequently block.

Disadvantages:

- The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.
- Since kernel must manage and schedule threads as well as processes. It require a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

Advantages of Threads over Multiple Processes

- **Context Switching** Threads are very inexpensive to create and destroy, and they are inexpensive to represent. For example, they require space to store, the PC, the SP, and the general-purpose registers, but they do not require space to share memory information, Information about open files of I/O devices in use, etc. With so little context, it is much faster to switch between threads. In other words, it is relatively easier for a context switch using threads.
- **Sharing** Treads allow the sharing of a lot resources that cannot be shared in process, for example, sharing code section, data section, Operating System resources like open file etc.

Disadvantages of Threads over Multiprocesses

- **Blocking** The major disadvantage if that if the kernel is single threaded, a system call of one thread will block the whole process and CPU may be idle during the blocking period.
- **Security** Since there is, an extensive sharing among threads there is a potential problem of security. It is quite possible that one thread over writes the stack of another thread (or damaged shared data) although it is very unlikely since threads are meant to cooperate on a single task.

Application that Benefits from Threads

A proxy server satisfying the requests for a number of computers on a LAN would be benefited by a multi-threaded process. In general, any program that has to do more than one task at a time could benefit from multitasking. For example, a program that reads input, process it, and outputs could have three threads, one for each task.

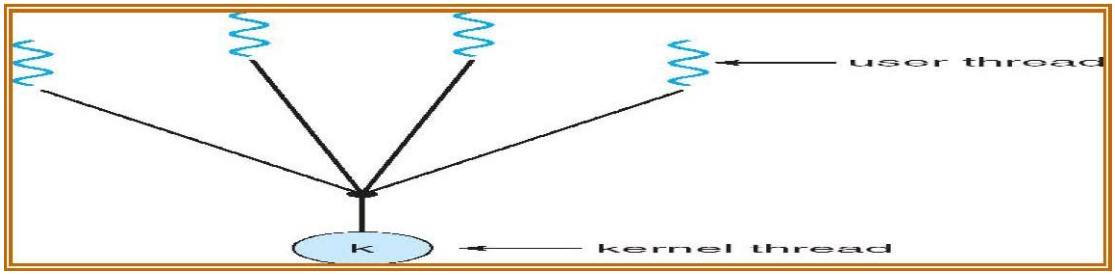
Application that cannot Benefit from Threads

Any sequential process that cannot be divided into parallel task will not benefit from thread, as they would block until the previous one completes. For example, a program that displays the time of the day would not benefit from multiple threads.

2.6 OVERVIEW:MULTITHREADING MODELS

- Many-to-One
- One-to-One
- Many-to-Many

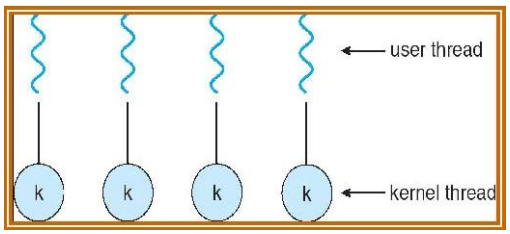
Many-to-One Many user-level threads mapped to single kernel thread
->Examples: ->Solaris Green Threads, ->GNU Portable Threads



One-to-One

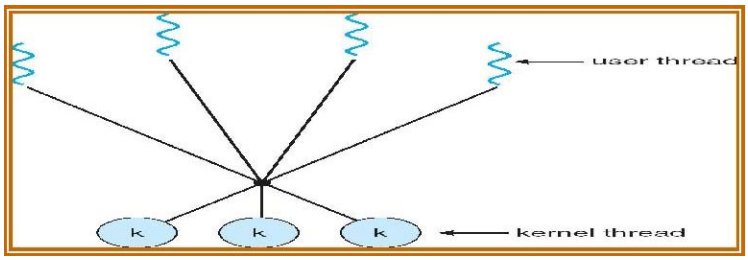
1. Each user-level thread maps to kernel thread

- Examples Windows NT/XP/2000
- Linux
- Solaris 9 and later



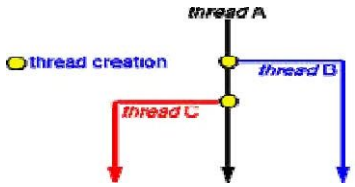
Many-to-Many Model

1. Allows many user level threads to be mapped to many kernel threads.
2. Allows the operating system to create a sufficient number of kernel threads.
3. Solaris prior to version 9.
4. Windows NT/2000 with the *ThreadFiber* package.



2.7 THREAD LIBRARIES, THREADING ISSUES.

Resources used in Thread Creation and Process Creation



When a new thread is created it shares its code section, data section and operating system resources like open files with other threads. But it is allocated its own stack, register set and a program counter.

The creation of a new process differs from that of a thread mainly in the fact that all the shared resources of a thread are needed explicitly for each process. So though two processes may be running the same piece of code they need to have their own copy of the code in the main memory to be able to run. Two processes also do not share other resources with each other. This makes the creation of a new process very costly compared to that of a new thread.

Thread Pools

1. Create a number of threads in a pool where they await work

- Advantages: Usually slightly faster to service a request with an existing thread than create a new thread
- Allows the number of threads in the application(s) to be bound to the size of the pool

Context Switch

To give each process on a multiprogrammed machine a fair share of the CPU, a hardware clock generates interrupts periodically. This allows the operating system to schedule all processes in main memory (using scheduling algorithm) to run on the CPU at equal intervals. Each time a clock interrupt occurs, the interrupt handler checks how much time the current running process has used. If it has used up its entire time slice, then the CPU scheduling algorithm (in kernel) picks a different process to run. Each switch of the CPU from one process to another is called a context switch.

Major Steps of Context Switching

x The values of the CPU registers are saved in the process table of the process that was running just before the clock interrupt occurred. x The registers are loaded from the process picked by the CPU scheduler to run next.

In a multiprogrammed uniprocessor computing system, context switches occur frequently enough that all processes appear to be running concurrently. If a process has more than one thread, the Operating System can use the context switching technique to schedule the threads so they appear to execute in parallel. This is the case if threads are implemented at the kernel level. Threads can also be implemented entirely at the user level in run-time libraries. Since in this case no thread scheduling is provided by the Operating System, it is the responsibility of the programmer to yield the CPU frequently enough in each thread so all threads in the process can make progress.

Action of Kernel to Context Switch Among Threads

The threads share a lot of resources with other peer threads belonging to the same process. So a context switch among threads for the same process is easy. It involves switch of register set, the program counter and the stack. It is relatively easy for the kernel to accomplished this task.

Action of kernel to Context Switch Among Processes

Context switches among processes are expensive. Before a process can be switched its process control block (PCB) must be saved by the operating system. The PCB consists of the following information:

- The process state.
- The program counter, PC.
- The values of the different registers.
- The CPU scheduling information for the process.
- Memory management information regarding the process.
- Possible accounting information for this process.
- I/O status information of the process.

When the PCB of the currently executing process is saved the operating system loads the PCB of the next process that has to be run on CPU. This is a heavy task and it takes a lot of time.

2.8 PROCESS SCHEDULING: BASIC CONCEPTS

The assignment of physical processors to processes allows processors to accomplish work. The problem of determining when processors should be assigned and to which processes is called processor scheduling or CPU scheduling. When more than one process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

CPU Scheduler

a. Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
b. CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state

2. Switches from running to ready state

3. Switches from waiting to ready
4. Terminates

- Scheduling under 1 and 4 is *nonpreemptive*
- All other scheduling is *preemptive*

Dispatcher

1. Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:

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

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

2.9 SCHEDULING CRITERIA

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

General Goals

Fairness

Fairness is important under all circumstances. A scheduler makes sure that each process gets its fair share of the CPU and no process can suffer indefinite postponement. Note that giving equivalent or equal time is not fair. Think of *safety control* and *payroll* at a nuclear plant.

Policy Enforcement

The scheduler has to make sure that system's policy is enforced. For example, if the local policy is safety then the *safety control processes* must be able to run whenever they want to, even if it means delay in *payroll processes*.

Efficiency

Scheduler should keep the system (or in particular CPU) busy cent percent of the time when possible. If the CPU and all the Input/Output devices can be kept running all the time, more work gets done per second than if some components are idle.

Response Time

A scheduler should minimize the response time for interactive user.

Turnaround

A scheduler should minimize the time batch users must wait for an output.

Throughput

A scheduler should maximize the number of jobs processed per unit time.

A little thought will show that some of these goals are contradictory. It can be shown that any scheduling

algorithm that favors some class of jobs hurts another class of jobs. The amount of CPU time available is finite, after all.

Preemptive Vs Nonpreemptive Scheduling

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts.

Nonpreemptive Scheduling

A scheduling discipline is nonpreemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Following are some characteristics of nonpreemptive scheduling

1. In nonpreemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In nonpreemptive system, response times are more predictable because incoming high priority jobs can not displace waiting jobs.
3. In nonpreemptive scheduling, a schedular executes jobs in the following two situations.

a. When a process switches from running state to the waiting state.
b. When a process terminates.

Preemptive Scheduling

A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.

The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

2.10 SCHEDULING ALGORITHMS

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. Following are some scheduling algorithms we will study.

- FCFS Scheduling.
- Round Robin Scheduling.
- SJF Scheduling.
- SRT Scheduling.
- Priority Scheduling.
- Multilevel Queue Scheduling.
- Multilevel Feedback Queue Scheduling.

First-Come-First-Served (FCFS) Scheduling

Other names of this algorithm are:

- First-In-First-Out (FIFO)
- Run-to-Completion

x Run-Until-Done

Perhaps, First-Come-First-Served algorithm is the simplest scheduling algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a nonpreemptive discipline, once a process has a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait.

FCFS is more predictable than most of other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand. One of the major drawback of this scheme is that the average time is often quite long. The First-Come-First-Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

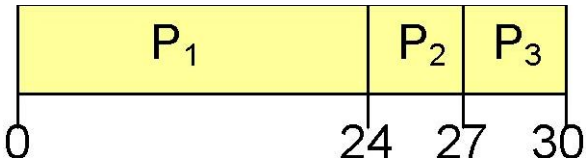
Example:-Process Burst Time

P1 24

P2 3

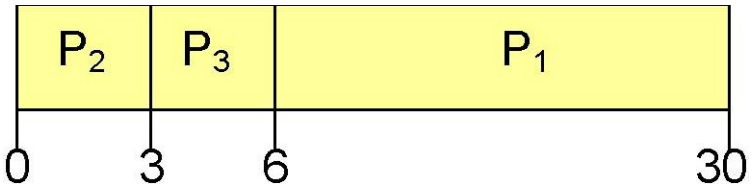
P3 3

Suppose that the processes arrive in the order: *P1* , *P2* , *P3* ,The Gantt Chart for the schedule is:



Waiting time for *P1* = 0; *P2* = 24; *P3* = 27, Average waiting time: (0 + 24 + 27)/3 = 17 Suppose that the

processes arrive in the order *P2* , *P3* , *P1* , The Gantt chart for the schedule is:



Waiting time for *P1* = 6*; P2* =0*; P3* = 3, Average waiting time: (6 + 0 + 3)/3 = 3

Round Robin Scheduling

x One of the oldest, simplest, fairest and most widely used algorithm is round robin (RR). x In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum.

x If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next

process waiting in a queue. The preempted process is then placed at the back of the ready list.

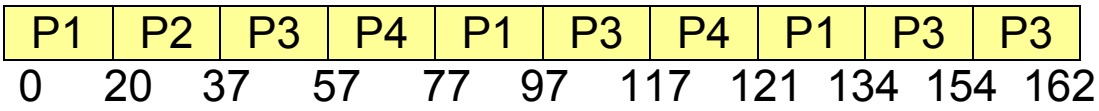
- x Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in timesharing environments in which the system needs to guarantee reasonable response times for interactive users.
- x The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency. On the other hand, setting the quantum too long may cause poor response time and appoximates FCFS.

In any event, the average waiting time under round robin scheduling is often quite long.

1. Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
2. If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
3. Performance ->*q* large . FIFO ->*q* small . *q* must be large with respect to context switch, otherwise overhead is too high. Example:Process Burst Time

P1 53 P2 17 P3 68 P4 24

The Gantt chart is: ->Typically, higher average turnaround than SJF, but better *response*



Shortest-Job-First (SJF) Scheduling

- x Other name of this algorithm is Shortest-Process-Next (SPN).
- x Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run next. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst.
- x The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.
- x The SJF algorithm favors short jobs (or processors) at the expense of longer ones. x The obvious problem with SJF scheme is that it requires precise knowledge of how long a job or process will run, and this information is not usually available. x The best SJF algorithm can do is to rely on user estimates of run times.

In the production environment where the same jobs run regularly, it may be possible to provide reasonable estimate of run time, based on the past performance of the process. But in the development environment users rarely know how their program will execute.

Like FCFS, SJF is non preemptive therefore, it is not useful in timesharing environment in which reasonable response time must be guaranteed.

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

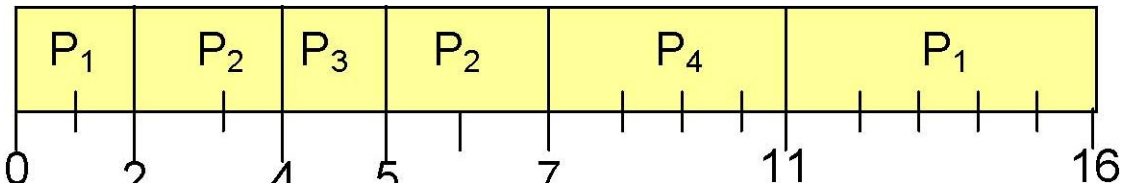
2 Two schemes:

- x nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
- x preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the ShortestRemaining-Time-First (SRTF)

3. SJF is optimal – gives minimum average waiting time for a given set of processes

Process Arrival Time Burst Time

P1	*0*.0	7
P2	*2*.0	4
P3	*4*.0	1
P4	*5*.0	4



-> SJF (preemptive)

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

Shortest-Remaining-Time (SRT) Scheduling

- The SRT is the preemtive counterpart of SJF and useful in time-sharing environment.
- In SRT scheduling, the process with the smallest estimated run-time to completion is run next, including new arrivals.
- In SJF scheme, once a job begin executing, it run to completion.
- In SJF scheme, a running process may be preempted by a new arrival process with shortest estimated run-time.
- The algorithm SRT has higher overhead than its counterpart SJF.
- The SRT must keep track of the elapsed time of the running process and must handle occasional preemptions.
- In this scheme, arrival of small processes will run almost immediately. However, longer jobs have even longer mean waiting time.

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer { highest priority)
  - ->Preemptive
  - ->nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem { Starvation – low priority processes may never execute
- Solution { Aging – as time progresses increase the priority of the process

The basic idea is straightforward: each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order. The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm. An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa.

Priority can be defined either internally or externally. Internally defined priorities use some measurable quantities or qualities to compute priority of a process.

Examples of Internal priorities are

x Time limits. x Memory requirements. x File requirements,
for example, number of open files. x CPU Vs I/O requirements.

Externally defined priorities are set by criteria that are external to operating system such as

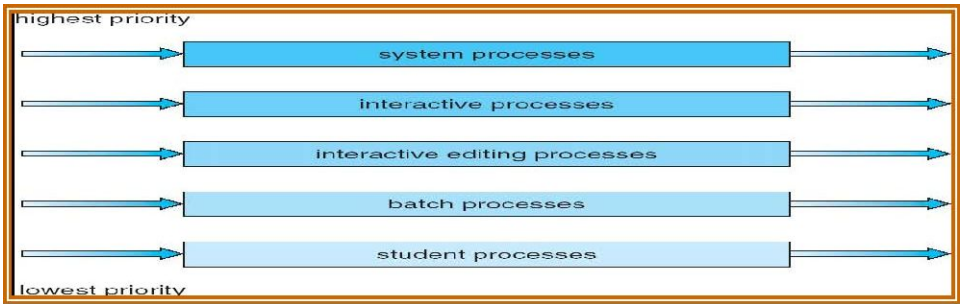
x The importance of process. x Type or amount of funds being paid for computer use. x The department sponsoring the work. x Politics.

Priority scheduling can be either preemptive or non preemptive

x A preemptive priority algorithm will preemptive the CPU if the priority of the newly arrival process is higher than the priority of the currently running process. x A non-preemptive priority algorithm will simply put the new process at the head of the ready queue.

A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is *aging*. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

Multilevel Queue Scheduling



A multilevel queue scheduling algorithm partitions the ready queue in several separate queues, for instance, In a multilevel queue scheduling processes are permanently assigned to one queues. The processes are permanently assigned to one another, based on some property of the process, such as Memory size , Process priority , Process type . Algorithm choose the process from the occupied queue that has the highest priority, and run that process either

Preemptive or Non-preemptively Each queue has its own scheduling algorithm or policy.

PossibilityI

If each queue has absolute priority over lower-priority queues then no process in the queue could run unless the queue for the highest-priority processes were all empty. For example, in the above figure no process in the batch queue could run unless the queues for system processes, interactive processes, and interactive editing processes will all empty.

Possibility II

If there is a time slice between the queues then each queue gets a certain amount of CPU times, which it can then schedule among the processes in its queue. For instance;

- 80% of the CPU time to foreground queue using RR.
- 20% of the CPU time to background queue using FCFS.

Since processes do not move between queue so, this policy has the advantage of low scheduling overhead, but it is inflexible.

Multilevel Feedback Queue Scheduling

Multilevel feedback queue-scheduling algorithm allows a process to move between queues. It uses many ready queues and associate a different priority with each queue. The Algorithm chooses to process with highest priority from the occupied queue and run that process either preemptively or unpreemptively. If the process uses too much CPU time it will moved to a lower-priority queue. Similarly, a process that wait too long in the lower-priority queue may be moved to a higher-priority queue may be moved to a highest-priority queue. Note that this form of aging prevents starvation.

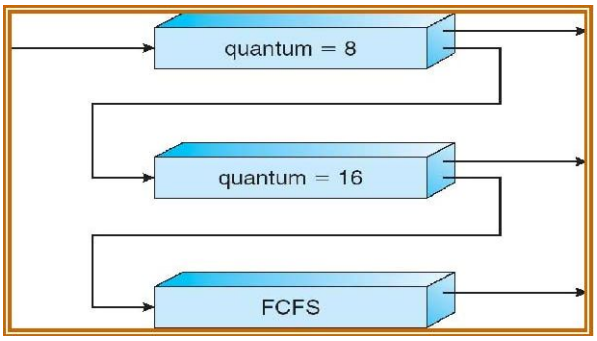
- A process entering the ready queue is placed in queue 0.
- If it does not finish within 8 milliseconds time, it is moved to the tail of queue 1.
- If it does not complete, it is preempted and placed into queue 2.
- Processes in queue 2 run on a FCFS basis, only when 2 run on a FCFS basis queue, only when queue 0 and queue 1 are empty.

Example:-Three queues:

- *Q0 – RR with time quantum 8 milliseconds*
- *Q1 – RR time quantum 16 milliseconds*
- *Q2 – FCFS*

1. Scheduling

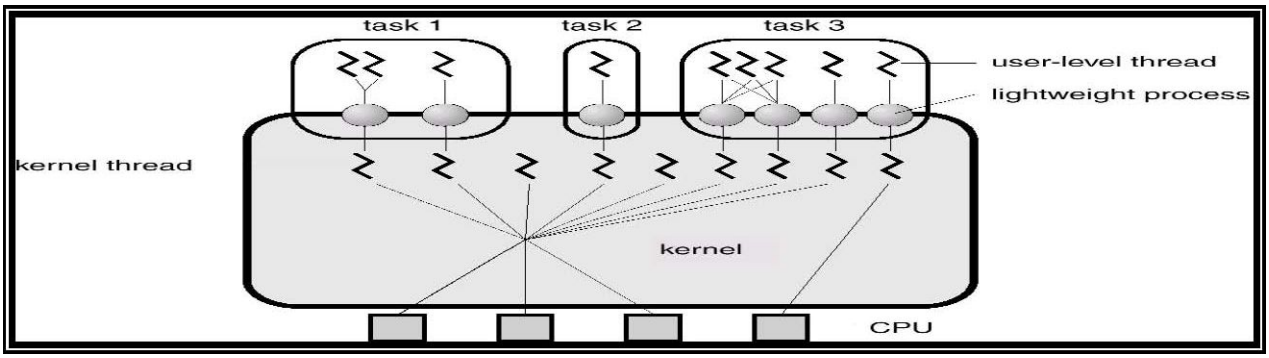
- A new job enters queue *Q0* which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue *Q*1.
- At *Q*1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue *Q*2.



2.11 THREAD SCHEDULING

Pthreads

x a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization. x API specifies behavior of the thread library, implementation is up to development of the library. x Common in UNIX operating systems.



Windows 2000 Threads

x Implements the one-to-one mapping.
x Each thread contains -a thread id -register set -separate user and kernel stacks -private data storage area

Linux threads

x Linux refers to them as *tasks* rather than *threads*. x Thread creation is done through clone() system call.
x Clone() allows a child task to share the address space of the parent task **(process)**

Java threads

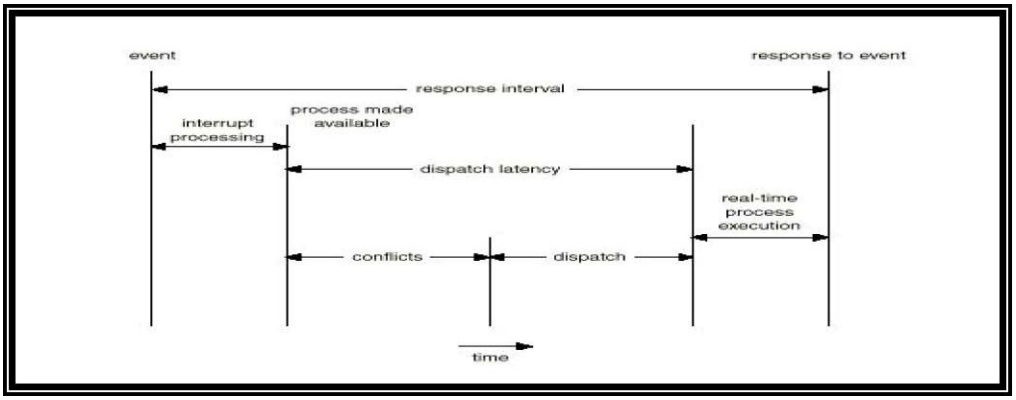
x Java threads may be created by:

- Extending Thread class
- Implementing the Runnable interface x Java threads are managed by the JVM.

2.12 MULTIPLE-PROCESSOR SCHEDULING

x CPU scheduling more complex when multiple CPUs are available. x Homogeneous processors within a multiprocessor. x Load sharing x Asymmetric multiprocessing – only one processor accesses the system data structures,
alleviating the need for data sharing.

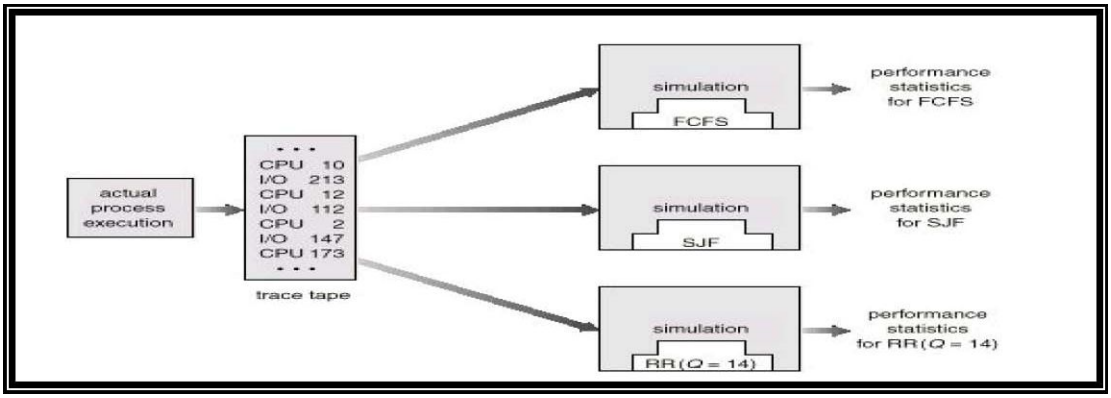
x Hard real-time systems – required to complete a critical task within a guaranteed amount of time. x Soft real-time computing – requires that critical processes receive priority over less fortunate ones.



Algorithm Evaluation

x Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload. x Queueing models x Implementation

Evaluation of CPU Schedulers by Simulation



IMPORTANT QUESTIONS:

1. Describe the differences among short-term, medium-term, and long-term scheduling.
2. Describe the actions a kernel takes to context switch between processes.
3. What are two differences between user-level threads and kernel-level threads?
4. Describe the actions taken by a kernel to context switch between kernel-level threads.
5. What resources are used when a thread is created? How do they differ from those used when a process is created?
6. Define the difference between preemptive and nonpreemptive scheduling
7. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds: Process Burst Time Priority *P*1 103

P2 11
P3 23
P4 14
P5 52

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
c. What is the waiting time of each process for each of the scheduling algorithms in part a?
d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

8. Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time. In answering the questions, use nonpreemptivescheduling and base all decisions on the information you have at the time the decisionmust be made.

Process Arrival Time Burst Time *P1 0.0* 8 *P2 0.4* 4 *P3 1.0* 1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?
b. What is the average turnaround time for these processes with the SJF scheduling algorithm?
c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process *P*1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P*1 and *P*2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

UNIT 3 PROCESS SYNCHRONIZATION

TOPICS

3.8 SYNCHRONIZATION
3.9 THE CRITICAL SECTION PROBLEM
3.10 PETERSON'S SOLUTION
3.11 SYNCHRONIZATION HARDWARE
3.12 SEMAPHORES
3.13 CLASSICAL PROBLEMS OF SYNCHRONIZATION
3.14 MONITORS

3.1 SYNCHRONIZATION

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

Race Conditions

In operating systems, processes that are working together share some common storage (main memory, file etc.) that each process can read and write. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions. Concurrently executing threads that share data need to synchronize their operations and processing in order to avoid race condition on shared data. Only one 'customer' thread at a time should be allowed to examine and update the shared variable. Race conditions are also possible in Operating Systems. If the ready queue is implemented as a linked list and if the ready queue is being manipulated during the handling of an interrupt, then interrupts must be disabled to prevent another interrupt before the first one completes. If interrupts are not disabled than the linked list could become corrupt.

1. count++ could be implemented as

   register1 = count

   register1 = register1 + 1

   count = register1

2. count--could be implemented as

   register2 = count

   register2 = register2 – 1

   count = register2

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

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

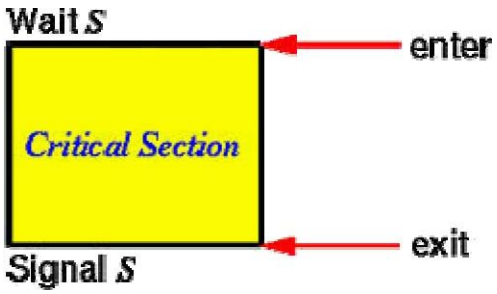
3.2 THE CRITICAL SECTION PROBLEM

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

2. Progress -If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. Bounded Waiting -A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

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

A. Critical Section



The key to preventing trouble involving shared storage is find some way to prohibit more than one process from reading and writing the shared data simultaneously. That part of the program where the shared memory is accessed is called the *Critical Section*. To avoid race conditions and flawed results, one must identify codes in *Critical Sections* in each thread. The characteristic properties of the code that form a *Critical Section* are

x Codes that reference one or more variables in a "read-update-write" fashion while any of those variables is possibly being altered by another thread. x Codes that alter one or more variables that are possibly being referenced in "read-updata-write"

fashion by another thread. x Codes use a data structure while any part of it is possibly being altered by another thread. x Codes alter any part of a data structure while it is possibly in use by another thread.

Here, the important point is that when one process is executing shared modifiable data in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time.

B. Mutual Exclusion

A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing. Formally, while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each

process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

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

Mutual Exclusion Conditions

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

- No two processes may at the same moment inside their critical sections.
- No assumptions are made about relative speeds of processes or number of CPUs.
- No process should outside its critical section should block other processes.
- No process should wait arbitrary long to enter its critical section.

3.3 PETERSON'S SOLUTION

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

Problem

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

Proposal 1 -Disabling Interrupts (Hardware Solution)

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

Conclusion

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

Proposal 2 -Lock Variable (Software Solution)

In this solution, we consider a single, shared, (lock) variable, initially 0. When a process wants to enter in its critical section, it first test the lock. If lock is 0, the process first sets it to 1 and then enters the critical section. If the lock is already 1, the process just waits until (lock) variable becomes 0. Thus, a 0 means that no process in its critical section, and 1 means hold your horses -some process is in its critical section.

Conclusion

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

Proposal 3 -Strict Alteration

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

Conclusion

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

Using Systems calls 'sleep' and 'wakeup'

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

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

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

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

The Bounded Buffer Producers and Consumers

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

Statement

To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions or lost updates. As an example how sleep-wakeup system calls are used, consider the producer-consumer problem also known

as bounded buffer problem. Two processes share a common, fixed-size (bounded) buffer. The producer puts information into the buffer and the consumer takes information out.

Trouble arises when

1. The producer wants to put a new data in the buffer, but buffer is already full. Solution: Producer goes to sleep and to be awakened when the consumer has removed data.
2. The consumer wants to remove data the buffer but buffer is already empty. Solution: Consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

Conclusion

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

3.4 SYNCHRONIZATION HARDWARE

1. Many systems provide hardware support for critical section code
2. Uniprocessors – could disable interrupts
   - Currently running code would execute without preemption
   - Generally too inefficient on multiprocessor systems
   - Operating systems using this not broadly scalable
   - 3. Modern machines provide special atomic hardware instructions

->Atomic = non-interruptable

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

3.5 SEMAPHORES

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

Definition

A semaphore is a protected variable whose value can be accessed and altered only by the operations P and V and initialization operation called 'Semaphoiinitislize'.

Binary Semaphores can assume only the value 0 or the value 1 counting semaphores also called general semaphores can assume only nonnegative values. The P (or wait or sleep or down) operation on semaphores S, written as P(S) or wait (S), operates as follows:

P(S): **IF** S>0
THEN S:= S-1
ELSE (wait on S)

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

V(S): **IF** (one or more process are waiting on S)
THEN (let one of these processes proceed)
ELSE S := S +1

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

If several processes attempt a P(S) simultaneously, only process will be allowed to proceed. The other processes will be kept waiting, but the implementation of P and V guarantees that processes will not suffer indefinite postponement. Semaphores solve the lost-wakeup problem.

Semaphore as General Synchronization Tool

1. Counting semaphore – integer value can range over an unrestricted domain.
2. Binary semaphore – integer value can range only between and 1; can be simpler to implement Also known as mutex locks.
3. Can implement a counting semaphore S as a binary semaphore.
4. Provides mutual exclusion

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

Critical Section

signal (S);

Semaphore Implementation

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

- Could now have busy waiting in critical section implementation

- But implementation code is short
- Little busy waiting if critical section rarely occupied
- 3. Note that applications may spend lots of time in critical sections and therefore this is not a good solution.

Semaphore Implementation with no Busy waiting

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

- value (of type integer)
- pointer to next record in the list

2. Two operations:

- block – place the process invoking the operation on the appropriate waiting queue.
- wakeup – remove one of processes in the waiting queue and place it in the ready queue. ->Implementation of wait: wait (S){ value--;

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

->Implementation of signal:

```
Signal (S){
    value++;
    if (value <= 0) {
        remove a process P from the waiting queue
        wakeup(P); }
```

}

3.6 CLASSICAL PROBLEMS OF SYNCHRONIZATION

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

Bounded-Buffer Problem

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

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

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

}

6. The structure of the consumer process

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

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

// consume the removed item }

Readers-Writers Problem

1. A data set is shared among a number of concurrent processes
   - Readers – only read the data set; they do not perform any updates
   - Writers – can both read and write.

2. Problem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time.
3. Shared Data
   - Data set
   - Semaphore mutex initialized to 1.
   - Semaphore wrt initialized to 1.
   - Integer readcount initialized to 0.

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

signal (wrt) ; }

5. The structure of a reader process

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

// reading is performed

wait (mutex) ;

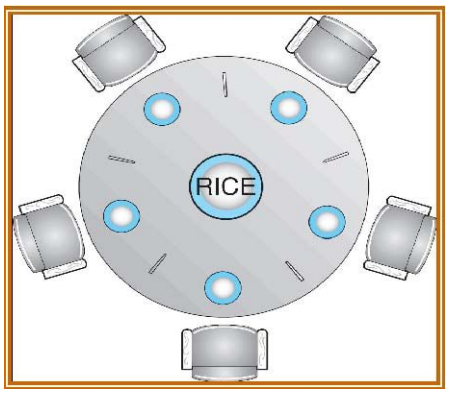
readcount --;

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

signal (mutex) ;

}

Dining-Philosophers Problem



1. Shared data

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

2. The structure of Philosopher i: While (true) { wait (chopstick[i]

); wait (chopStick[(i + 1) % 5]);

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

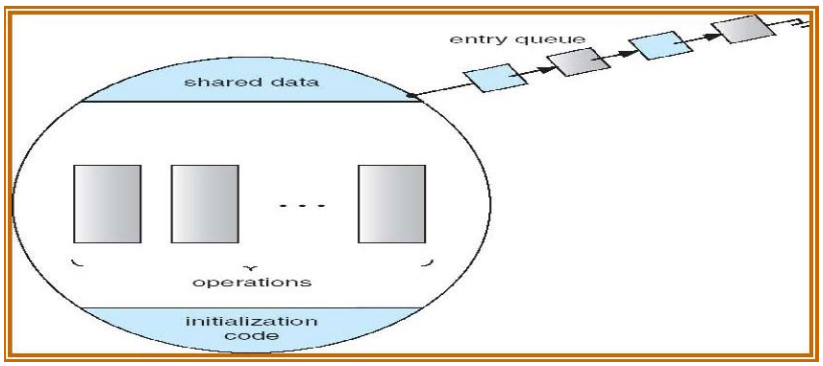
}

Problems with Semaphores

1. Correct use of semaphore operations:

- signal (mutex) …. wait (mutex)
- wait (mutex) … wait (mutex)
- Omitting of wait (mutex) or signal (mutex) (or both)

3.7 MONITORS



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

{ // shared variable declarations procedure P1 (…) { …. }

… procedure Pn (…) {……}

Initialization code (….) { … } … }

}

Solution to Dining Philosophers

```
monitor DP
  { enum { THINKING; HUNGRY, EATING) state [5] ; condition self [5];

        void pickup (int i) { state[i] = HUNGRY; test(i); if (state[i] != EATING) self [i].wait;
        }
     void putdown (int i) { state[i] = THINKING; // test left and right neighbors
         test((i + 4) % 5); test((i + 1) % 5); }
void test (int i) { if ( (state[(i + 4) % 5] != EATING) && (state[i] == HUNGRY) && (state[(i + 1) % 5] !=
             EATING) ) {
                state[i] = EATING ; self[i].signal () ; } }
     initialization_code() { for (int i = 0; i < 5; i++) state[i] = THINKING;
        }
} ->Each philosopher I invokes the operations pickup() and putdown() in the following sequence:
          dp.pickup (i) EAT dp.putdown (i)
```

Monitor Implementation Using Semaphores

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

2.Each procedure ***F*** will be replaced by wait(mutex); … body of F; … if (next-count > 0)

signal(next) else signal(mutex);

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

signal(next); else

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

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

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

}

Producer-Consumer Problem Using Semaphores

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

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

Initialization

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

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

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

IMPORTANT QUESTIONS:

1. What is the meaning of the term *busy waiting*?
2. Explain semaphores with the help of an example.
3. Explain dining philosophers problem, and how it is solved.
4. What is race condition? Explain how it is handled.
5. Define process synchronization.
6. How is producer-consumer problem with the help of semaphores?

UNIT 4 DEADLOCK

TOPICS

4.9 DEADLOCKS

4.10 SYSTEM MODEL

4.11 DEADLOCK CHARACTERIZATION

4.12 METHODS FOR HANDLING DEADLOCKS

4.13 DEADLOCK PREVENTION

4.14 DEADLOCK AVOIDANCE

4.15 DEADLOCK DETECTION

4.16 RECOVERY FROM DEADLOCK

4.1 DEADLOCKS

- When processes request a resource and if the resources are not available at that time the process enters into waiting state. Waiting process may not change its state because the resources they are requested are held by other process. This situation is called deadlock.
- The situation where the process waiting for the resource i.e., not available is called deadlock.

4.2 SYSTEM MODEL

- A system may consist of finite number of resources and is distributed among number of processes. There resources are partitioned into several instances each with identical instances.
- A process must request a resource before using it and it must release the resource after using it. It can request any number of resources to carry out a designated task. The amount of resource requested may not exceed the total number of resources available.

A process may utilize the resources in only the following sequences:

1. Request:-If the request is not granted immediately then the requesting process must wait it can acquire the resources.
2. Use:-The process can operate on the resource.
3. Release:-The process releases the resource after using it.

Deadlock may involve different types of resources.

For eg:-Consider a system with one printer and one tape drive. If a process Pi currently holds a printer and a process Pj holds the tape drive. If process Pi request a tape drive and process Pj request a printer then a deadlock occurs.

Multithread programs are good candidates for deadlock because they compete for shared resources.

4.3 DEADLOCK CHARACTERIZATION

Necessary Conditions:A deadlock situation can occur if the following 4 conditions occur simultaneously in a system:-1. Mutual Exclusion:Only one process must hold the resource at a time. If any other process requests for the resource, the requesting process must be delayed until the resource has been released.

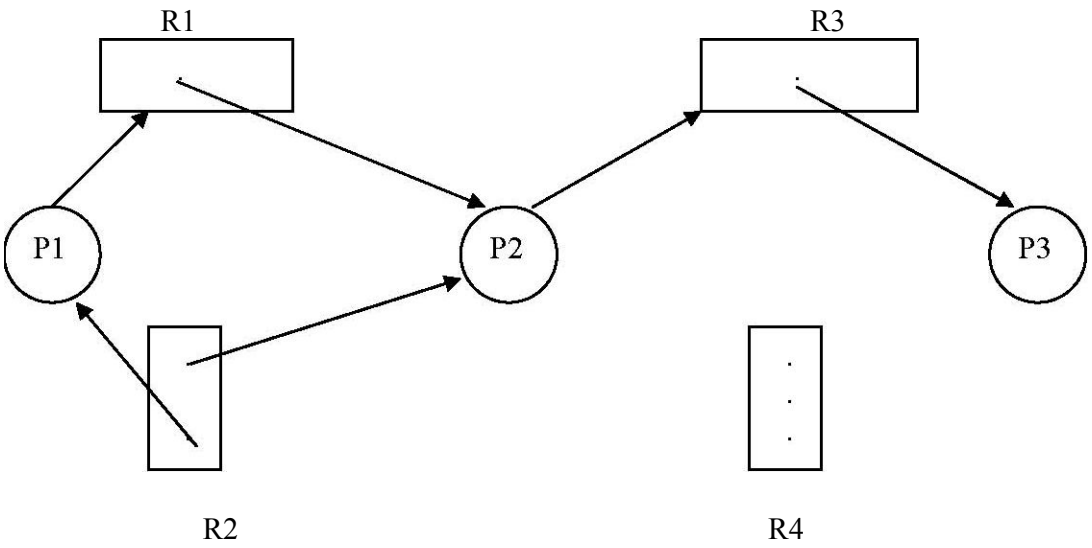
1. Hold and Wait:-A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by the other process.
2. No Preemption:-Resources can't be preempted i.e., only the process holding the resources must release it after the process has completed its task.
3. Circular Wait:-A set {P0,P1……..Pn} of waiting process must exist such that P0 is waiting for a resource i.e., held by P1, P1 is waiting for a resource i.e., held by P2. Pn-1 is waiting for resource held by process Pn and Pn is waiting for the resource i.e., held by P1. All the four conditions must hold for a deadlock to occur.

Resource Allocation Graph:

1. Deadlocks are described by using a directed graph called system resource allocation graph. The graph consists of set of vertices (v) and set of edges (e).
2. The set of vertices (v) can be described into two different types of nodes P={P1,P2……..Pn} i.e., set consisting of all active processes and R={R1,R2…….Rn}i.e., set consisting of all resource types in the system

7. A directed edge from process Pi to resource type Rj denoted by Pi->Ri indicates that Pi requested an instance of resource Rj and is waiting. This edge is called Request edge.
8. A directed edge Ri-> Pj signifies that resource Rj is held by process Pi. This is called Assignment edge

Eg:



- If the graph contain no cycle, then no process in the system is deadlock. If the graph contains a cycle then a deadlock may exist.
- If each resource type has exactly one instance than a cycle implies that a deadlock has occurred. If each resource has several instances then a cycle do not necessarily implies that a deadlock has occurred.

4.4 METHODS FOR HANDLING DEADLOCKS

There are three ways to deal with deadlock problem x We can use a protocol to prevent deadlocks ensuring that the system will never enter into the
deadlock state. x We allow a system to enter into deadlock state, detect it and recover from it. x We ignore the problem and pretend that the deadlock never occur in the system. This is used by most OS including UNIX.

- To ensure that the deadlock never occur the system can use either deadlock avoidance or a deadlock prevention.
- Deadlock prevention is a set of method for ensuring that at least one of the necessary conditions does not occur.
- Deadlock avoidance requires the OS is given advance information about which resource a process will request and use during its lifetime.
- If a system does not use either deadlock avoidance or deadlock prevention then a deadlock situation may occur. During this it can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and algorithm to recover from deadlock.

- Undetected deadlock will result in deterioration of the system performance.

4.5 DEADLOCK PREVENTION

- For a deadlock to occur each of the four necessary conditions must hold. If at least one of the there condition does not hold then we can prevent occurrence of deadlock.

1. Mutual Exclusion:This holds for non-sharable resources. *Eg*:-A printer can be used by only one process at a time.

Mutual exclusion is not possible in sharable resources and thus they cannot be involved in deadlock. Read-only files are good examples for sharable resources. A process never waits for accessing a sharable resource. So we cannot prevent deadlock by denying the mutual exclusion condition in non-sharable resources.

2. Hold and Wait:This condition can be eliminated by forcing a process to release all its resources held by it when it request a resource i.e., not available.

x One protocol can be used is that each process is allocated with all of its resources before its start execution. *Eg*:-consider a process that copies the data from a tape drive to the disk, sorts the file and then prints the results to a printer. If all the resources are allocated at the beginning then the tape drive, disk files and printer are assigned to the process. The main problem with this is it leads to low resource utilization because it requires printer at the last and is allocated with it from the beginning so that no other process can use it. x Another protocol that can be used is to allow a process to request a resource when the process has none. i.e., the process is allocated with tape drive and disk file. It performs the required operation and releases both. Then the process once again request for disk file and the printer and the problem and with this is starvation is possible.

3. No Preemption:To ensure that this condition never occurs the resources must be preempted. The following protocol can be used.

x If a process is holding some resource and request another resource that cannot be immediately allocated to it, then all the resources currently held by the requesting process are preempted and added to the list of resources for which other processes may be waiting. The process will be restarted only when it regains the old resources and the new resources that it is requesting.

x When a process request resources, we check whether they are available or not. If they are available we allocate them else we check that whether they are allocated to some other waiting process. If so we preempt the resources from the waiting process and allocate them to the requesting process. The requesting process must wait.

4.Circular Wait:-The fourth and the final condition for deadlock is the circular wait condition. One way to ensure that this condition never, is to impose ordering on all resource types and each process requests resource in an increasing order.

Let R={R1,R2,……Rn} be the set of resource types. We assign each resource type with a unique integer value. This will allows us to compare two resources and determine whether one precedes the other in ordering. *Eg*:-we can define a one to one function

F:R→N as follows :-F(disk drive)=5 F(printer)=12 F(tape drive)=1

Deadlock can be prevented by using the following protocol:

- Each process can request the resource in increasing order. A process can request any number of instances of resource type say Ri and it can request instances of resource type Rj only F(Rj) > F(Ri).
- Alternatively when a process requests an instance of resource type Rj, it has released any resource Ri such that F(Ri) >= F(Rj). If these two protocol are used then the circular wait can't hold.

4.6 DEADLOCK AVOIDANCE

- Deadlock prevention algorithm may lead to low device utilization and reduces system throughput.
- Avoiding deadlocks requires additional information about how resources are to be requested. With the knowledge of the complete sequences of requests and releases we can decide for each requests whether or not the process should wait.
- For each requests it requires to check the resources currently available, resources that are currently allocated to each processes future requests and release of each process to decide whether the current requests can be satisfied or must wait to avoid future possible deadlock.
- A deadlock avoidance algorithm dynamically examines the resources allocation state to ensure that a circular wait condition never exists. The resource allocation state is defined by the number of available and allocated resources and the maximum demand of each process.

Safe State:

- A state is a safe state in which there exists at least one order in which all the process will run completely without resulting in a deadlock.
- A system is in safe state if there exists a safe sequence.
- A sequence of processes <P1,P2,……..Pn> is a safe sequence for the current allocation state if for each Pi the resources that Pi can request can be satisfied by the currently available resources.
- If the resources that Pi requests are not currently available then Pi can obtain all of its needed resource to complete its designated task.
- A safe state is not a deadlock state.
- Whenever a process request a resource i.e., currently available, the system must decide whether resources can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in safe state.
- In this, if a process requests a resource i.e., currently available it must still have to wait. Thus resource utilization may be lower than it would be without a deadlock avoidance algorithm.

Resource Allocation Graph Algorithm:

- This algorithm is used only if we have one instance of a resource type. In addition to the request edge and the assignment edge a new edge called claim edge is used. *For eg*:-A claim edge Pi→Rj indicates that process Pi may request Rj in future. The claim edge

is represented by a dotted line.

x When a process Pi requests the resource Rj, the claim edge is converted to a request edge. x When resource Rj is released by process Pi, the assignment edge Rj→Pi is replaced by the claim edge Pi→Rj.

- When a process Pi requests resource Rj the request is granted only if converting the request edge Pi→Rj to as assignment edge Rj→Pi do not result in a cycle. Cycle detection algorithm is used to detect the cycle. If there are no cycles then the allocation of the resource to process leave the system in safe state

. **Banker's Algorithm**:

- This algorithm is applicable to the system with multiple instances of each resource types, but this is less efficient then the resource allocation graph algorithm.
- When a new process enters the system it must declare the maximum number of resources that it may need. This number may not exceed the total number of resources in the system. The system must determine that whether the allocation of the resources will leave the system in a safe state or not. If it is so resources are allocated else it should wait until the process release enough resources.
- Several data structures are used to implement the banker's algorithm. Let 'n' be the number of processes in the system and 'm' be the number of resources types. We need the following data structures:

x **Available**:-A vector of length m indicates the number of available resources. If Available[i]=k, then k instances of resource type Rj is available.
x **Max**:-An n*m matrix defines the maximum demand of each process if Max[i,j]=k, then Pi may request at most k instances of resource type Rj.
x **Allocation**:-An n*m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j]=k, then Pi is currently k instances of resource type Rj.
x **Need**:-An n*m matrix indicates the remaining resources need of each process. If Need[i,j]=k, then Pi may need k more instances of resource type Rj to compute its task. So Need[i,j]=Max[i,j]-Allocation[i]

Safety Algorithm:

- This algorithm is used to find out whether or not a system is in safe state or not. Step 1. Let work and finish be two vectors of length M and N respectively. Initialize work = available and Finish[i]=false for i=1,2,3,…….n

Step 2. Find i such that both Finish[i]=false Need i <= work
If no such i exist then go to step 4
Step 3. Work = work + Allocation Finish[i]=true Go to step 2
Step 4. If finish[i]=true for all i, then the system is in safe state. This algorithm may require an order of m*n*n operation to decide whether a state is safe.

Resource Request Algorithm:Let Request(i) be the request vector of process Pi. If Request(i)[j]=k, then process Pi wants K instances of the resource type Rj. When a request for resources is made by process Pi the following actions are taken.

x If Request(i) <= Need(i) go to step 2 otherwise raise an error condition since the process has exceeded its maximum claim. x If Request(i) <= Available go to step 3 otherwise Pi must wait. Since the resources are not available. x If the system want to allocate the requested resources to process Pi then modify the state as follows. Available = Available – Request(i) Allocation(i) = Allocation(i) + Request(i) Need(i) = Need(i) – Request(i)

x If the resulting resource allocation state is safe, the transaction is complete and Pi is allocated its resources. If the new state is unsafe then Pi must wait for Request(i) and old resource allocation state is restored.

4.7 DEADLOCK DETECTION

If a system does not employ either deadlock prevention or a deadlock avoidance algorithm then a deadlock situation may occur. In this environment the system may provide x An algorithm that examines the state of the system to determine whether a deadlock has occurred. x An algorithm to recover from the deadlock.

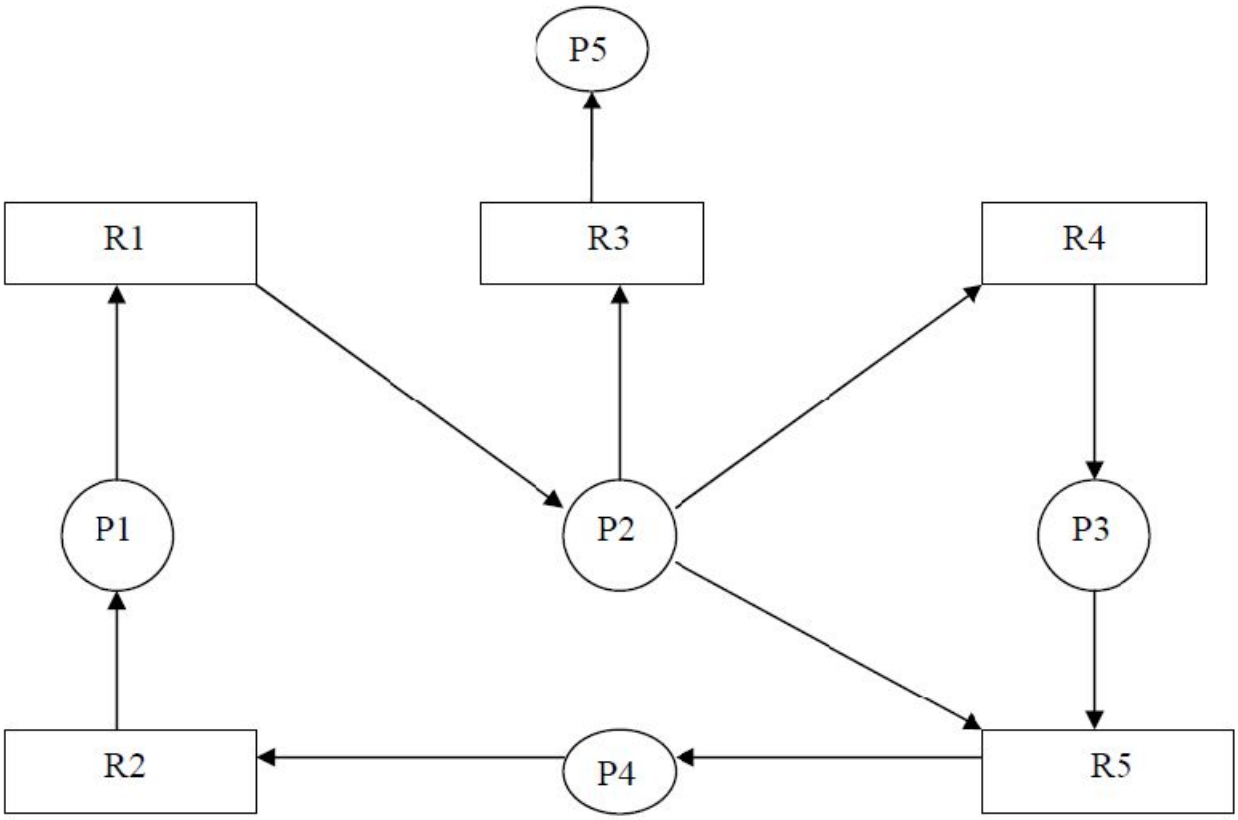
Single Instances of each Resource Type:

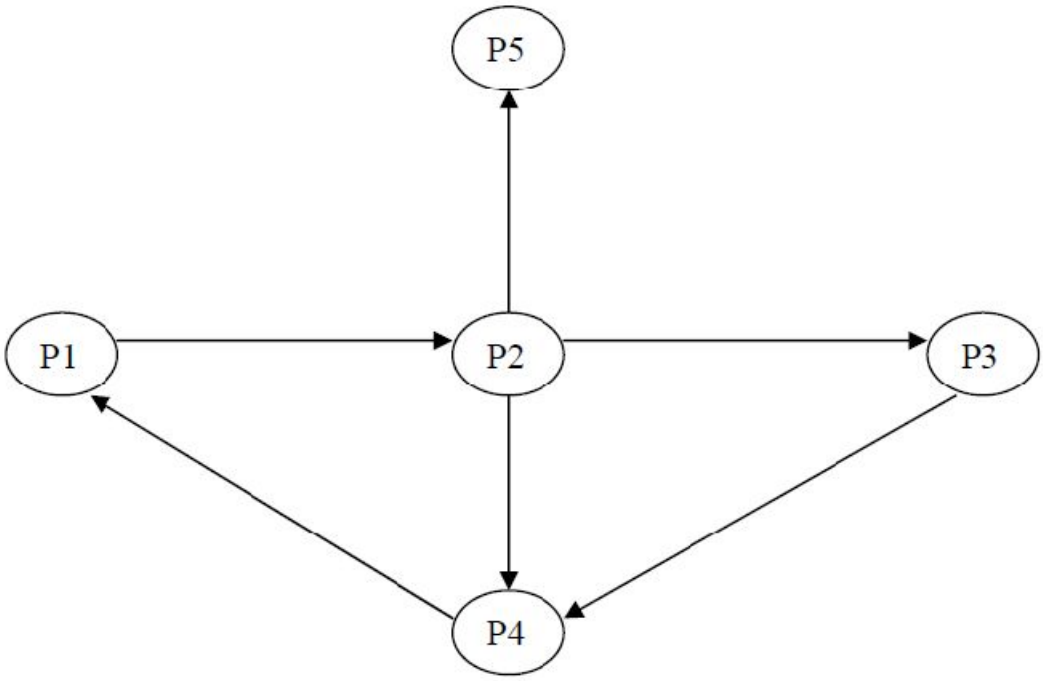
- If all the resources have only a single instance then we can define deadlock detection algorithm that uses a variant of resource allocation graph called a wait for graph. This graph is obtained by removing the nodes of type resources and removing appropriate edges.

- An edge from Pi to Pj in wait for graph implies that Pi is waiting for Pj to release a resource that Pi needs.

- An edge from Pi to Pj exists in wait for graph if and only if the corresponding resource allocation graph contains the edges Pi→Rq and Rq→Pj.

- Deadlock exists within the system if and only if there is a cycle. To detect deadlock the system needs an algorithm that searches for cycle in a graph.





4.8 RECOVERY FROM DEADLOCKS

Several Instances of a Resource Types:

- The wait for graph is applicable to only a single instance of a resource type. The following algorithm applies if there are several instances of a resource type. The following data structures are used:-

- o Available:-Is a vector of length m indicating the number of available resources of each type .

- o Allocation:-Is an m*n matrix which defines the number of resources of each type currently allocated to each process.

- o Request:-Is an m*n matrix indicating the current request of each process. If request[i,j]=k then Pi is requesting k more instances of resources type Rj.

Step 1. let work and finish be vectors of length m and n respectively. Initialize Work = available/expression
For i=0,1,2…….n if allocation(i)!=0 then Finish[i]=0
else Finish[i]=true
Step 2. Find an index(i) such that both Finish[i]
= false Request(i)<=work
If no such I exist go to step 4.

Step 3. Work = work + Allocation(i) Finish[i] = true Go to step 2. Step 4. If Finish[i] = false for some I where m>=i>=1. When a system is in a deadlock state. This algorithm needs an order of m*n square operations to detect whether the system is in deadlock state or not.

IMPORTANT QUESTIONS:

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

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

a. Calculate the need of each process?
b. To find safe sequence?

2. Consider the following snapshot of the system and answer the following questions using Banker's algorithm?
a. Find the need of the allocation?

b. Is the system is in safe state?
c. If the process P1 request (0,4,2,0) resources cam the request be granted immediately?

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

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

a. Is the current allocation is safe?
b. find need?
c. Can the request made by the process P1(1,1,0) can be granted?

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

4. Explain different methods to recover from deadlock?
5. Write advantage and disadvantage of deadlock avoidance and deadlock prevention?

UNIT 5 Storage Management

TOPICS

5.13 MEMORY MANAGEMENT STRATEGIES
5.14 BACKGROUND
5.15 SWAPPING
5.16 CONTIGUOUS MEMORY ALLOCATION
5.17 PAGING, STRUCTURE OF PAGE TABLE
5.18 SEGMENTATION
5.19 VIRTUAL MEMORY MANAGEMENT
5.20 BACKGROUND,DEMAND PAGING
5.21 COPY-ON-WRITE
5.22 PAGE REPLACEMENT
5.23 ALLOCATION OF FRAMES
5.24 THRASHING

5.1 MEMORY MANAGEMENT STRATGIES

x Memory management is concerned with managing the primary memory. x Memory consists of array of bytes or words each with their own address. x The instructions are fetched from the memory by the cpu based on the value program counter.

Functions of memory management:

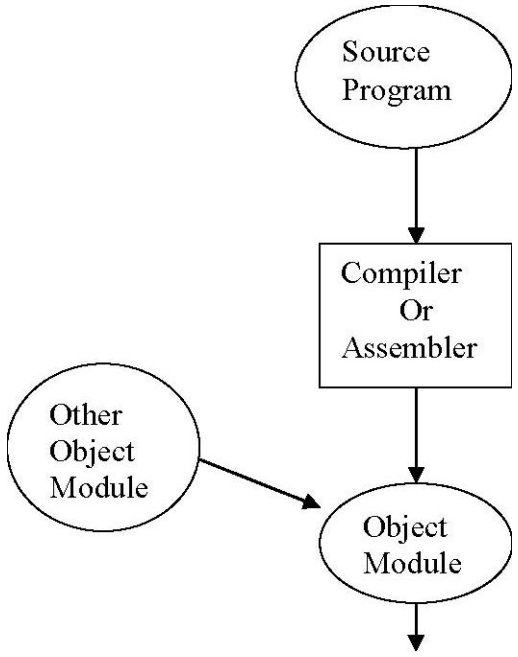
x Keeping track of status of each memory location.. x Determining the allocation policy. x Memory allocation technique. x De-allocation technique.

Address Binding:

x Programs are stored on the secondary storage disks as binary executable files. x When the programs are to be executed they are brought in to the main memory and placed within a process. x The collection of processes on the disk waiting to enter the main memory forms the input queue. x One of the processes which are to be executed is fetched from the queue and placed in the main memory. x During the execution it fetches instruction and data from main memory. After the process terminates it returns back the memory space. x During execution the process will go through different steps and in each step the address is represented in different ways. x In source program the address is symbolic. x The compiler converts the symbolic address to re-locatable address. x The loader will convert this re-locatable address to absolute address.

Binding of instructions and data can be done at any step along the way:

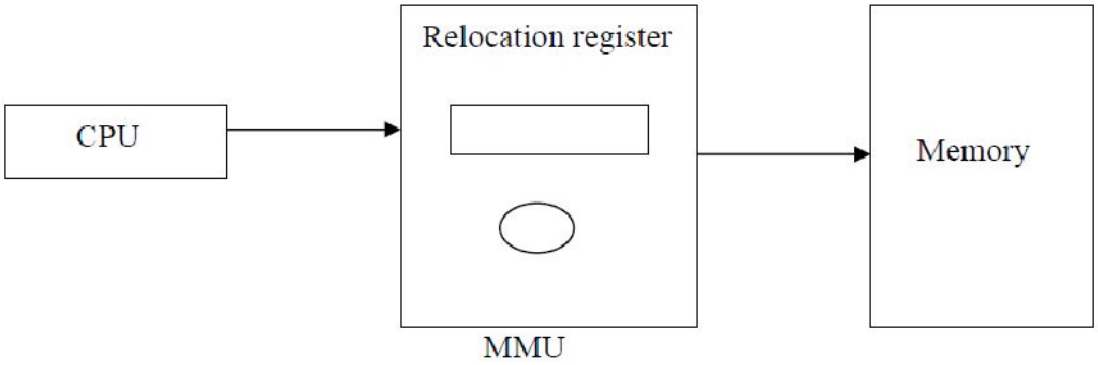
1. Compile time:-If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.
2. Load time:-If the compiler doesn't know whether the process resides in memory then it generates the re-locatable code. In this the binding is delayed until the load time.
3. Execution time:-If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.



5.2 BACKGROUND

Logical versus physical address:

x The address generated by the CPU is called logical address or virtual address. x The address seen by the memory unit i.e., the one loaded in to the memory register is called the physical address. x Compile time and load time address binding methods generate some logical and physical address. x The execution time addressing binding generate different logical and physical address. x Set of logical address space generated by the programs is the logical address space. x Set of physical address corresponding to these logical addresses is the physical address space. x The mapping of virtual address to physical address during run time is done by the hardware device called memory management unit (MMU). x The base register is also called re-location register. x Value of the re-location register is added to every address generated by the user process at the time it is sent to memory.



Dynamic re-location using a re-location registers

The above figure shows that dynamic re-location which implies mapping from virtual addresses space to physical address space and is performed by the hardware at run time.
Re-location is performed by the hardware and is invisible to the user dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.

Dynamic Loading:

x For a process to be executed it should be loaded in to the physical memory. The size of the process is limited to the size of the physical memory. x Dynamic loading is used to obtain better memory utilization. x In dynamic loading the routine or procedure will not be loaded until it is called. x Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not. If it is not loaded it cause the loader to load the desired program in to the memory and updates the programs address table to indicate the change and control is passed to newly called routine.

Advantage:x Gives better memory utilization. x Unused routine is never loaded. x Do not need special operating system support. x This method is useful when large amount of codes are needed to handle in frequently occurring cases.

Dynamic linking and Shared libraries:

x Some operating system supports only the static linking.
x In dynamic linking only the main program is loaded in to the memory. If the main program requests a procedure, the procedure is loaded and the link is established at the time of references. This linking is postponed until the execution time.
x With dynamic linking a "stub" is used in the image of each library referenced routine. A "stub" is a piece of code which is used to indicate how to locate the appropriate memory resident library routine or how to load library if the routine is not already present.
  x When "stub" is executed it checks whether the routine is present is memory or not. If not it loads the routine in to the memory.
  x This feature can be used to update libraries i.e., library is replaced by a new version and all the programs can make use of this library.
  x More than one version of the library can be loaded in memory at a time and each program uses its version of the library. Only the program that are compiled with the new version are affected by the changes incorporated in it. Other programs linked before new version is installed will continue using older libraries this type of system is called "shared library".

Overlays:

  x The size of the process is limited to the size of physical memory. If the size is more than the size of physical memory then a technique called overlays is used.
  x The idea is to load only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded in to memory apace that was previously occupied by the

instructions that are no longer needed. *Eg*:

Consider a 2-pass assembler where pass-1 generates a symbol table and pass-2 generates a machine code. Assume that the sizes of components are as follows:

Pass-1 = 70k Pass-2 = 80k Symbol table = 20k Common routine = 30k

To load everything at once, it requires 200k of memory. Suppose if 150k of memory is available, we can't run all the components at same time.

- Thus we define 2 overlays, overlay A which consist of symbol table, common routine and pass1 and overlay B which consists of symbol table, common routine and pass-2.
- We add an overlay driver and start overlay A in memory. When we finish pass-1 we jump to overlay driver, then the control is transferred to pass-2.
- Thus we can run assembler in 150k of memory.
- The code for overlays A and B are kept on disk as absolute memory images. Special re-location and linking algorithms are needed to construct the overlays. They can be implemented using simple file structures.

5.3 SWAPPING

- Swapping is a technique of temporarily removing inactive programs from the memory of the system.
- A process can be swapped temporarily out of the memory to a backing store and then brought back in to the memory for continuing the execution. This process is called swapping.

Eg:-In a multi-programming environment with a round robin CPU scheduling whenever the time quantum expires then the process that has just finished is swapped out and a new process swaps in to the memory for execution.

- A variation of swap is priority based scheduling. When a low priority is executing and if a high priority process arrives then a low priority will be swapped out and high priority is allowed for execution. This process is also called as Roll out and Roll in.
- Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously. This depends upon address binding.
- If the binding is done at load time, then the process is moved to same memory location.
- If the binding is done at run time, then the process is moved to different memory location. This is because the physical address is computed during run time.
- Swapping requires backing store and it should be large enough to accommodate the copies of all memory images.
- The system maintains a ready queue consisting of all the processes whose memory images are on the backing store or in memory that are ready to run.
- Swapping is constant by other factors:x To swap a process, it should be completely idle. x A process may be waiting for an i/o operation. If the i/o is asynchronously accessing the

user memory for i/o buffers, then the process cannot be swapped.

5.4 CONTIGUOUS MEMORY ALLOCATION

x One of the simplest method for memory allocation is to divide memory in to several fixed partition. Each partition contains exactly one process. The degree of multi-programming depends on the number of partitions.

x In multiple partition method, when a partition is free, process is selected from the input queue and is loaded in to free partition of memory. x When process terminates, the memory partition becomes available for another process. x Batch OS uses the fixed size partition scheme. x The OS keeps a table indicating which part of the memory is free and is occupied. x When the process enters the system it will be loaded in to the input queue. The OS keeps track of the memory requirement of each process and the amount of memory available and determines which process to allocate the memory. x When a process requests, the OS searches for large hole for this process, hole is a large block of free memory available. x If the hole is too large it is split in to two. One part is allocated to the requesting process and other is returned to the set of holes. x The set of holes are searched to determine which hole is best to allocate. There are three strategies to select a free hole:

- o First bit:-Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
- o Best bit:-It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
- o Worst fit:-It allocates the largest hole to the process request. It searches for the largest hole in the entire list.

• First fit and best fit are the most popular algorithms for dynamic memory allocation. First fit is generally faster. Best fit searches for the entire list to find the smallest hole i.e., large enough. Worst fit reduces the rate of production of smallest holes.
• All these algorithms suffer from fragmentation.

Memory Protection:

x Memory protection means protecting the OS from user process and protecting process from one another. x Memory protection is provided by using a re-location register, with a limit register. x Re-location register contains the values of smallest physical address and limit register contains range of logical addresses. (Re-location = 100040 and limit = 74600). x The logical address must be less than the limit register, the MMU maps the logical address dynamically by adding the value in re-location register. x When the CPU scheduler selects a process for execution, the dispatcher loads the re-location and limit register with correct values as a part of context switch. x Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.

Fragmentation:

• Memory fragmentation can be of two types:x Internal Fragmentation x External Fragmentation
• ➓ In Internal Fragmentation there is wasted space internal to a portion due to the fact that block of
• data loaded is smaller than the partition. *Eg*:-If there is a block of 50kb and if the process requests 40kb and if the block is allocated to the process then there will be 10kb of memory left.
• External Fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes.
• External Fragmentation may be either minor or a major problem.
• One solution for over-coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the relocation is static and is done at load time then compaction is not possible. Compaction is possible if the re-location is dynamic and done at execution time.
• Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non-contiguous, thus allowing the process to be allocated physical memory whenever the latter is available.

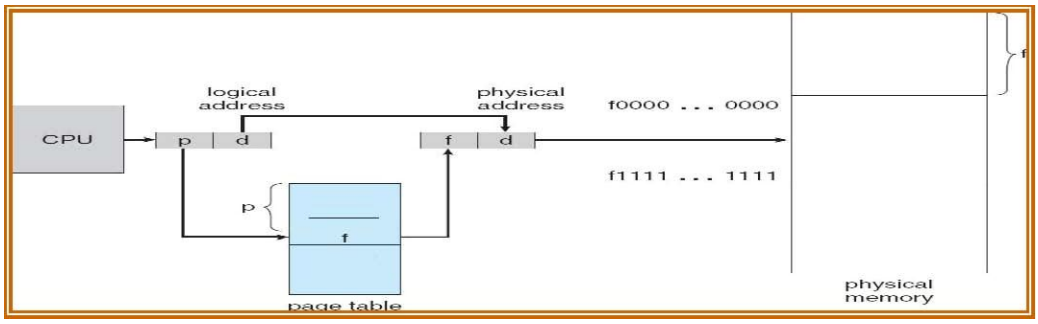
5.5 PAGING AND STRUCTURE OF PAGE TABLE

x Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware. x It is used to avoid external fragmentation. x Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store.

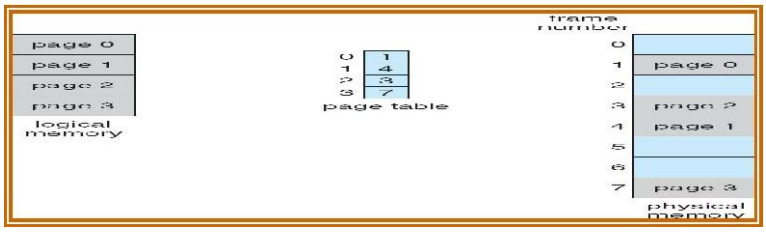
x When some code or date residing in main memory need to be swapped out, space must be found on backing store.

Basic Method:

x Physical memory is broken in to fixed sized blocks called frames (f). x Logical memory is broken in to blocks of same size called pages (p). x When a process is to be executed its pages are loaded in to available frames from backing store. x The blocking store is also divided in to fixed-sized blocks of same size as memory frames. x The following figure shows paging hardware:



x Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).
x The page number (p) is used as index to the page table. The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e., sent to the memory unit.
x
x The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page.
x If the size of logical address space is 2^m address unit and page size is 2^n, then high order m-n designates the page number and n low order bits represents page offset.



Eg:-To show how to map logical memory in to physical memory consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).

a. Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
b. Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].

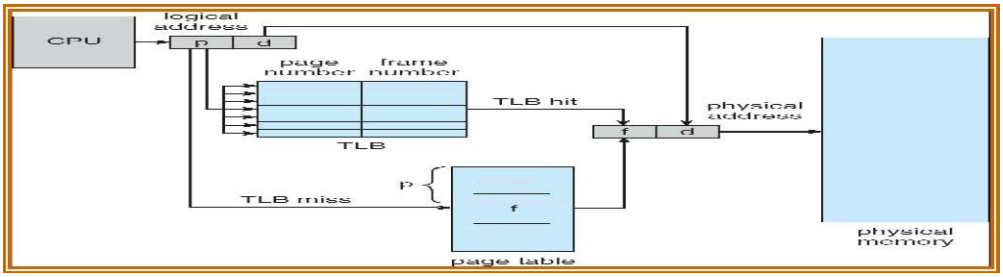
c. Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
d. Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

Hardware Support for Paging:

The hardware implementation of the page table can be done in several ways:

1. The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every accessed memory must go through paging map. The use of registers for page table is satisfactory if the page table is small.
2. If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a page table base register [PTBR] points to the page table. Changing the page table requires only one register which reduces the context switching type. The problem with this approach is the time required to access memory location. To access a location [i] first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.
3. The only solution is to use special, fast, lookup hardware cache called translation look aside buffer [TLB] or associative register.

TLB is built with associative register with high speed memory. Each register contains two paths a key and a value.



When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast.

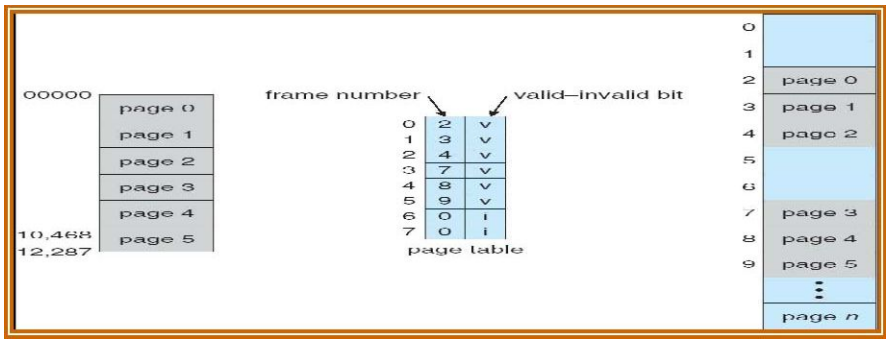
TLB is used with the page table as follows:x TLB contains only few page table entries. x When a logical address is generated by the CPU, its page number along with the frame number is added to TLB. If the page number is found its frame memory is used to access the actual memory. x If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained use can use it to access the memory. x If the TLB is full of entries the OS must select anyone for replacement. x Each time a new page table is selected the TLB must be flushed [erased] to ensure that next executing process do not use wrong information. x The percentage of time that a page number is found in the TLB is called HIT ratio.

Protection:

x

Memory protection in paged environment is done by protection bits that are associated with each frame these bits are kept in page table. x One bit can define a page to be read-write or read-only. x

To find the correct frame number every reference to the memory should go through page table. At the same time physical address is computed. x The protection bits can be checked to verify that no writers are made to read-only page. x Any attempt to write in to read-only page causes a hardware trap to the OS. x This approach can be used to provide protection to read-only, read-write or execute-only pages. x One more bit is generally added to each entry in the page table: a valid-invalid bit.



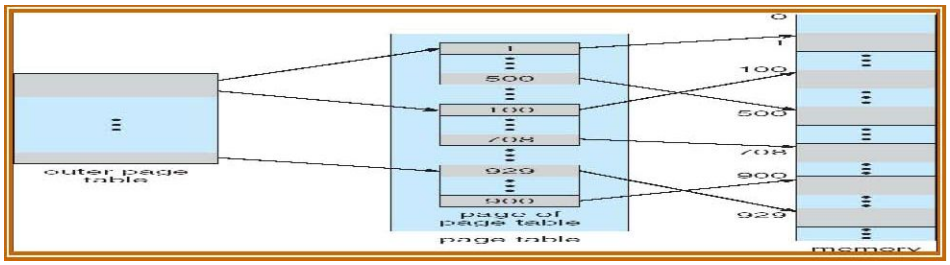
x A valid bit indicates that associated page is in the processes logical address space and thus it is a legal or valid page.
x If the bit is invalid, it indicates the page is not in the processes logical addressed space and illegal. Illegal addresses are trapped by using the valid-invalid bit.
x The OS sets this bit for each page to allow or disallow accesses to that page.

Structure of the Page Table

a. **Hierarchical paging:**

• Recent computer system support a large logical address apace from 2^32 to 2^64. In this system the page table becomes large. So it is very difficult to allocate contiguous main memory for page table. One simple solution to this problem is to divide page table in to smaller pieces. There are several ways to accomplish this division.

• One way is to use two-level paging algorithm in which the page table itself is also paged. *Eg*:-In a 32 bit machine with page size of 4kb. A logical address is divided in to a page number consisting of 20 bits and a page offset of 12 bit. The page table is further divided since the page table is paged, the page number is further divided in to 10 bit page number and a 10 bit offset. So the logical address is



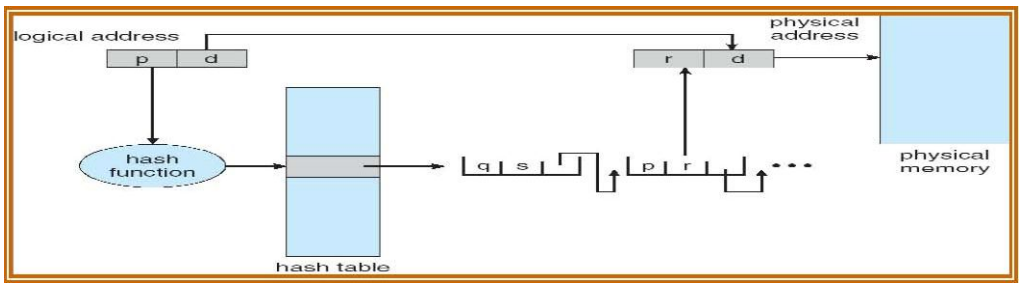
b. **Hashed page table:**

• Hashed page table handles the address space larger than 32 bit. The virtual page number is used as hashed value. Linked list is used in the hash table which contains a list of elements that hash to the same location.

• Each element in the hash table contains the following three fields:x Virtual page number x Mapped page frame value x Pointer to the next element in the linked list

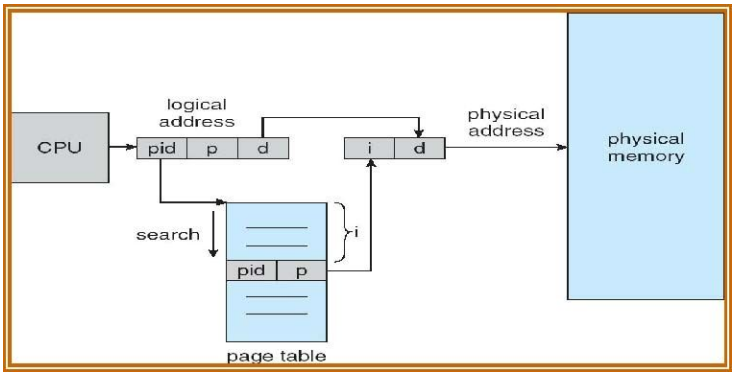
Working:

x Virtual page number is taken from virtual address.
x Virtual page number is hashed in to hash table.
x Virtual page number is compared with the first element of linked list.
x Both the values are matched, that value is (page frame) used for calculating the physical address. x If not match then entire linked list is searched for matching virtual page number. x Clustered pages are similar to hash table but one difference is that each entity in the hash table refer to several pages.



c. **Inverted Page Tables:**Since the address spaces have grown to 64 bits, the traditional page tables become a problem. Even with two level page tables. The table can be too large to handle. An inverted page table has only entry for each page in memory. Each entry consisted of virtual address of the page stored in that read-only location with information about the process that owns that page.

Each virtual address in the Inverted page table consists of triple <process-id , page number , offset >. The inverted page table entry is a pair <process-id , page number>. When a memory reference is made, the part of virtual address i.e., <process-id , page number> is presented in to memory sub-system. The inverted page table is searched for a match. If a match is found at entry I then the physical address <i , offset> is generated. If no match is found then an illegal address access has been attempted. This scheme decreases the amount of memory needed to store each page table, it increases the amount of time needed to search the table when a page reference occurs. If the whole table is to be searched it takes too long.



Advantage:

- x Eliminates fragmentation.
- x Support high degree of multiprogramming.
- x Increases memory and processor utilization.
- x Compaction overhead required for the re-locatable partition scheme is also eliminated.

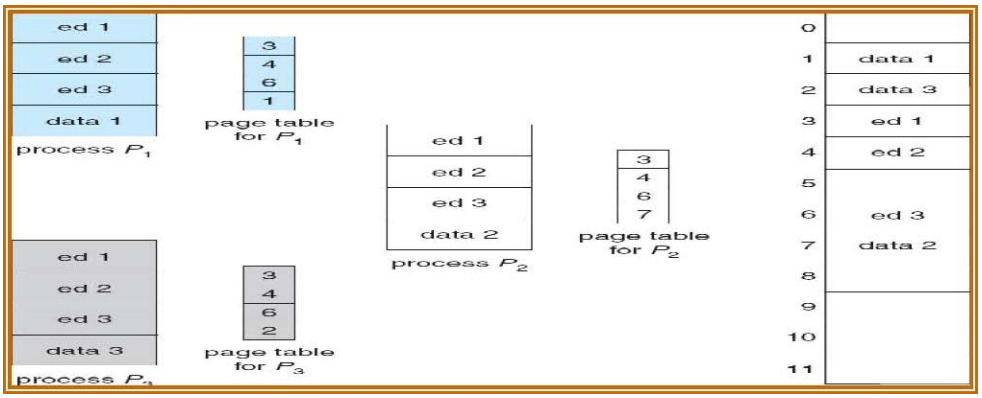
Disadvantage:

- x Page address mapping hardware increases the cost of the computer.
- x Memory must be used to store the various tables like page tables, memory map table etc.
- x Some memory will still be unused if the number of available block is not sufficient for the address space of the jobs to be run.

Shared Pages:

- ➓ Another advantage of paging is the possibility of sharing common code. This is useful in timesharing environment. *Eg*:-Consider a system with 40 users, each executing a text editor. If the text editor is of 150k

and data space is 50k, we need 8000k for 40 users. If the code is reentrant it can be shared. Consider the following figure



- If the code is reentrant then it never changes during execution. Thus two or more processes can execute same code at the same time. Each process has its own copy of registers and the data of two processes will vary.
- Only one copy of the editor is kept in physical memory. Each users page table maps to same physical copy of editor but date pages are mapped to different frames.
- So to support 40 users we need only one copy of editor (150k) plus 40 copies of 50k of data space i.e., only 2150k instead of 8000k.

5.6 SEGMENTATION

Basic method:

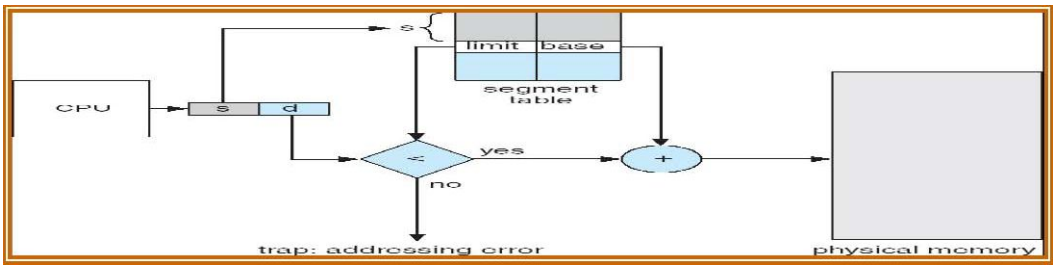
x Most users do not think memory as a linear array of bytes rather the users thinks memory as a collection of variable sized segments which are dedicated to a particular use such as code, data, stack, heap etc.

x A logical address is a collection of segments. Each segment has a name and length. The address specifies both the segment name and the offset within the segments.

x The users specifies address by using two quantities: a segment name and an offset. x For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.

Hardware support:x We must define an implementation to map 2D user defined address in to 1D physical address.

x This mapping is affected by a segment table. Each entry in the segment table has a segment base and segment limit. The segment base contains the starting physical address where the segment resides and limit specifies the length of the segment.



The use of segment table is shown in the above figure:x Logical address consists of two parts: segment number‘s’ and an offset‘d’
to that segment. x The segment number is used as an index to segment table. x The offset ’d’ must bi in between 0 and limit, if not an error is reported to OS. x If legal the offset is added to the base to generate the actual physical address. x The segment table is an array of base limit register pairs.

Protection and Sharing:x A particular advantage of segmentation is the association of protection with the segments.

- x The memory mapping hardware will check the protection bits associated with each segment table entry to prevent illegal access to memory like attempts to write in to read-only segment.
- x Another advantage of segmentation involves the sharing of code or data. Each process has a segment table associated with it. Segments are shared when the entries in the segment tables of two different processes points to same physical location.
- x Sharing occurs at the segment table. Any information can be shared at the segment level. Several segments can be shared so a program consisting of several segments can be shared.
- x We can also share parts of a program.

Advantages:x Eliminates fragmentation. x Provides virtual growth. x Allows dynamic segment growth. x Assist dynamic linking. x Segmentation is visible.

Differences between segmentation and paging:-

Segmentation:

x Program is divided in to variable sized segments. x User is responsible for dividing the program in to segments.

x Segmentation is slower than paging.

x Visible to user.

x Eliminates internal fragmentation.

x Suffers from external fragmentation.

x Process or user segment number, offset to calculate absolute address.

Paging:

x Programs are divided in to fixed size pages.

x Division is performed by the OS.

x Paging is faster than segmentation. x Invisible to user. x Suffers from internal fragmentation. x No external fragmentation. x Process or user page number, offset to calculate absolute address.

5.7 VIRTUAL MEMORY MANAGEMENT

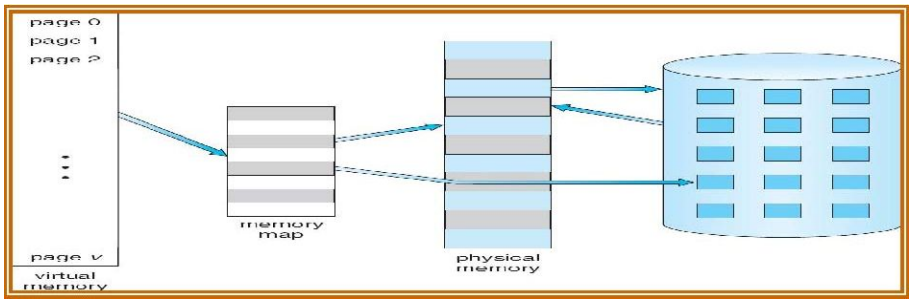
Virtual memory is a technique that allows for the execution of partially loaded process. There are many advantages of this:x A program will not be limited by the amount of physical memory that is available user can able to write in to large virtual space. x Since each program takes less amount of physical memory, more than one program could be run at the same time which can increase the throughput and CPU utilization. x Less i/o operation is needed to swap or load user program in to memory. So each user program could run faster.

- Virtual memory is the separation of users logical memory from physical memory. This separation allows an extremely large virtual memory to be provided when these is less physical memory.
- Separating logical memory from physical memory also allows files and memory to be shared by several different processes through page sharing.
- Virtual memory is implemented using Demand Paging.

5.8 BACKGROUND,DEMAND PAGING

- A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
- A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.

Basic concept: Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed.



The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory.

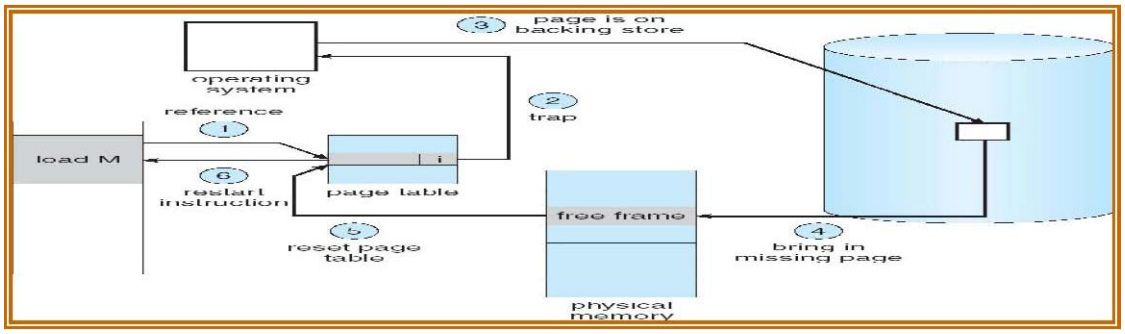
- o If the bit is valid then the page is both legal and is in memory.
- o If the bit is invalid then either page is not valid or is valid but is currently

on the disk. Marking a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

The step for handling page fault is straight forward and is given below:

1. We check the internal table of the process to determine whether the reference made is valid or invalid.
2. If invalid terminate the process,. If valid, then the page is not yet loaded and we now page it in.
3. We find a free frame.
4. We schedule disk operation to read the desired page in to newly allocated frame.
5. When disk reed is complete, we modify the internal table kept with the process to indicate that the page is now in memory.
6. We restart the instruction which was interrupted by illegal address trap. The process can

now access the page. In extreme cases, we start the process without pages in memory. When the OS points to the instruction of process it generates a page fault. After this page is brought in to memory the process continues to execute, faulting as necessary until every demand paging i.e., it never brings the page in to memory until it is required.



Hardware support:

For demand paging the same hardware is required as paging and swapping.

1. Page table:-Has the ability to mark an entry invalid through valid-invalid bit.
2. Secondary memory:-This holds the pages that are not present in main memory. I

Performance of demand paging:Demand paging can have significant effect on the performance of the computer system.

Let P be the probability of the page fault (0<=P<=1)

Effective access time = (1-P) * ma + P * page fault. Where P = page fault and ma = memory access time. Effective access time is directly proportional to page fault rate. It is important to keep page fault rate low in demand paging.

A page fault causes the following sequence to occur:

1. Trap to the OS.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Checks the page references were legal and determine the location of page on disk.
5. Issue a read from disk to a free frame.
6. If waiting, allocate the CPU to some other user.
7. Interrupt from the disk.
8. Save the registers and process states of other users.
9. Determine that the interrupt was from the disk.
10. Correct the page table and other table to show that the desired page is now in memory.

11. Wait for the CPU to be allocated to this process again.
12. Restore the user register process state and new page table then resume the interrupted instruction.

Comparison of demand paging with segmentation:-

Segmentation:

- Segment may of different size.
- Segment can be shared.
- Allows for dynamic growth of segments.
- Segment map table indicate the address of each segment in memory.
- Segments are allocated to the program while compilation.

Demand Paging:

- Pages are of same size.
- Pages can't be shared.
- Page size is fixed.
- Page table keeps track of pages in memory.
- Pages are allocated in memory on demand.

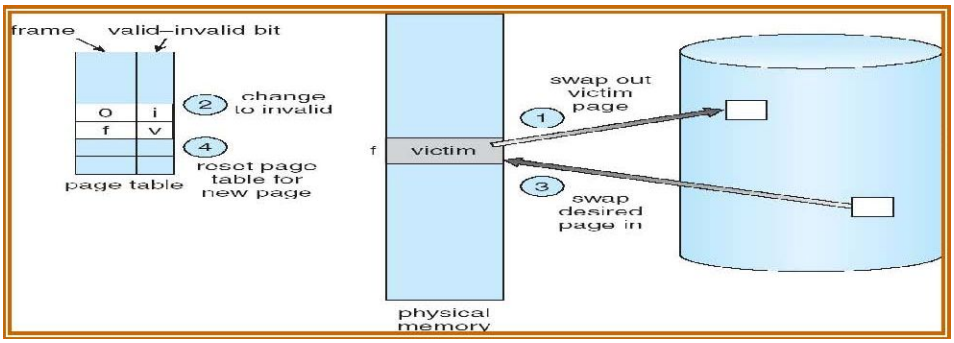
5.9 COPY-ON-WRITE

Demand paging is used when reading a file from disk in to memory. Fork () is used to create a process and it initially bypass the demand paging using a technique called page sharing. Page sharing provides rapid speed for process creation and reduces the number of pages allocated to the newly created process. Copy-on-write technique initially allows the parent and the child to share the same pages. These pages are marked as copy-on-write pages i.e., if either process writes to a shared page, a copy of shared page is created. *Eg*:-If a child process try to modify a page containing portions of the stack; the OS recognizes them as a copy-on-write page and create a copy of this page and maps it on to the address space of the child process. So the child process will modify its copied page and not the page belonging to parent. The new pages are obtained from the pool of free pages.

Memory Mapping: Standard system calls i.e., open (), read () and write () is used for sequential read of a file. Virtual memory is used for this. In memory mapping a file allows a part of the virtual address space to be logically associated with a file. Memory mapping a file is possible by mapping a disk block to page in memory.

5.10 PAGE REPLACEMENT

- Demand paging shares the I/O by not loading the pages that are never used.
- Demand paging also improves the degree of multiprogramming by allowing more process to run at the some time.
- Page replacement policy deals with the solution of pages in memory to be replaced by a new page that must be brought in. When a user process is executing a page fault occurs.
- The hardware traps to the operating system, which checks the internal table to see that this is a page fault and not an illegal memory access.
- The operating system determines where the derived page is residing on the disk, and this finds that thee are no free frames on the list of free frames.
- When all the frames are in main memory, it is necessary to bring a new page to satisfy the page fault, replacement policy is concerned with selecting a page currently in memory to be replaced.
- The page i,e to be removed should be the page i,e least likely to be referenced in future.



Working of Page Replacement Algorithm

1 Find the location of derived page on the disk.
2 Find a free frame x If there is a free frame, use it. x Otherwise, use a replacement algorithm to select a victim. x Write the victim page to the disk; change the page and frame tables accordingly.
3 Read the desired page into the free frame; change the page and frame tables.
4 Restart the user process.

Victim Page

The page that is supported out of physical memory is called victim page. x If no frames are free, the two page transforms come (out and one in) are read. This will see the effective access time.

- x Each page or frame may have a dirty (modify) bit associated with the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.
- x When we select the page for replacement, we check its modify bit. If the bit is set, then the page is modified since it was read from the disk.
- x If the bit was not set, the page has not been modified since it was read into memory. Therefore, if the copy of the page has not been modified we can avoid writing the memory page to the disk, if it is already there. Sum pages cannot be modified.

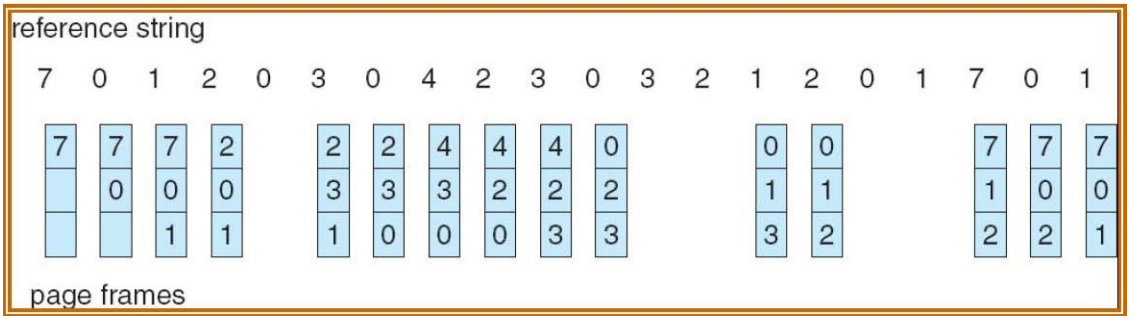
We must solve two major problems to implement demand paging: we must develop a frame allocation algorithm and a page replacement algorithm. If we have multiple processors in memory, we must decide how many frames to allocate and page replacement is needed.

Page replacement Algorithms

FIFO Algorithm:

- This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory.
- When a Page is to be replaced the oldest one is selected.
- We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

Example: Consider the following references string with frames initially empty.



- The first three references (7,0,1) cases page faults and are brought into the empty frames. The next references 2 replaces page 7 because the page 7 was brought in first. Since 0 is the next references and 0 is already in memory e has no page faults. The next references 3 results in page 0 being replaced so that the next references to 0 causer page fault. This will continue till the end of string. There are 15 faults all together.

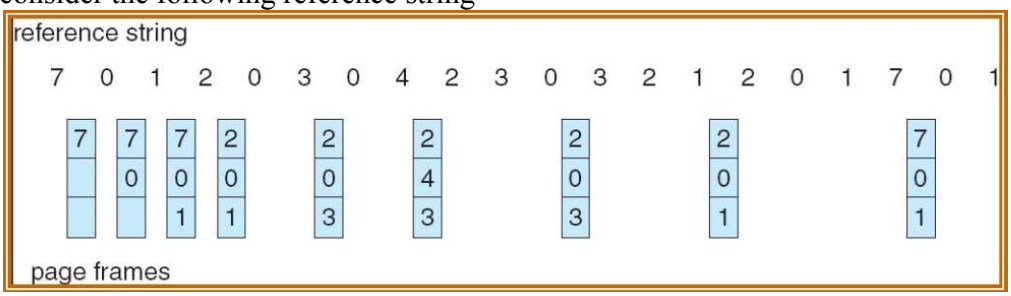
Belady's Anamoly

For some page replacement algorithm, the page fault may increase as the number of allocated frames increases. FIFO replacement algorithm may face this problem.

Optimal Algorithm

- Optimal page replacement algorithm is mainly to solve the problem of Belady's Anamoly. Optimal page replacement algorithm has the lowest page fault rate of all algorithms. An optimal page replacement algorithm exists and has been called OPT.

The working is simple "Replace the page that will not be used for the longest period of time" Example: consider the following reference string



- The first three references cause faults that fill the three empty frames.
- The references to page 2 replaces page 7, because 7 will not be used until reference 18.

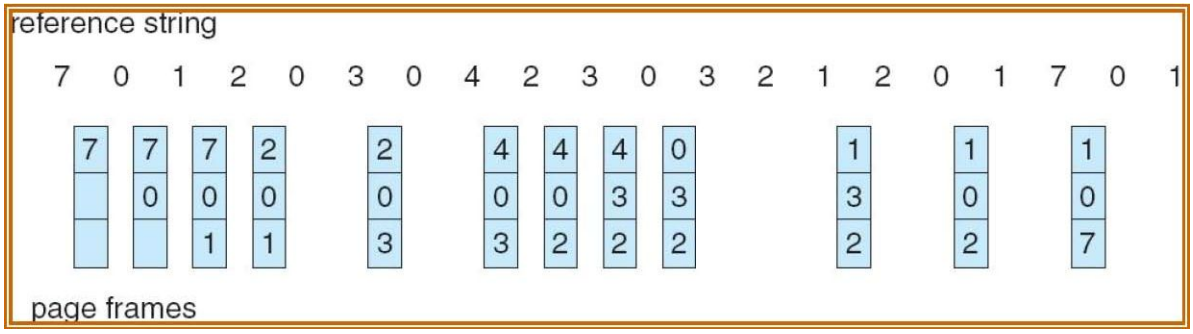
x The page 0 will be used at 5 and page 1 at 14.
x With only 9 page faults, optimal replacement is much better than a FIFO, which had 15 faults.
This algorithm is difficult t implement because it requires future knowledge of reference strings.

Least Recently Used (LRU) Algorithm

If the optimal algorithm is not feasible, an approximation to the optimal algorithm is possible. The main difference b/w OPTS and FIFO is that;

- FIFO algorithm uses the time when the pages was built in and OPT uses the time when a page is to be used.
- The LRU algorithm replaces the pages that have not been used for longest period of time. x The LRU associated its pages with the time of that pages last use. x This strategy is the optimal page replacement algorithm looking backward in time

rather than forward. Ex: consider the following reference string



x The first 5 faults are similar to optimal replacement. x

When reference to page 4 occurs, LRU sees that of the three frames, page 2 as used least recently. The most recently used page is page 0 and just before page 3 was used. The LRU policy is often used as a page replacement algorithm and
considered to be good. The main problem to how to implement LRU

is the LRU requires additional h/w assistance.
Two implementation are possible:

Counters: In this we associate each page table entry a time -of -use field, and add to the cpu a logical clock or counter. The clock is incremented for each memory reference. When 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 have the time of last reference to each page we replace the page with smallest time value. The time must also be maintained when page tables are changed.

Stack: Another approach to implement LRU replacement is to keep a stack of page numbers when a page is referenced it is removed from the stack and put on to the top of stack. In this way the top of stack is always the most recently used page and the bottom in least recently used page. Since the entries are removed from the stack it is best implement by a doubly linked list. With a head and tail pointer. Neither optimal replacement nor LRU replacement suffers from Belady's Anamoly. These are called stack algorithms.

LRU Approximation

- An LRU page replacement algorithm should update the page removal status information after every page reference updating is done by software, cost increases.
- But hardware LRU mechanism tend to degrade execution performance at the same time, then substantially increases the cost. For this reason, simple and efficient algorithm that approximation the LRU have been developed. With h/w support the reference bit was used. A reference bit associate with each memory block and this bit automatically set to 1 by the h/w whenever the page is referenced. The single reference bit per clock can be used to approximate LRU removal.
- The page removal s/w periodically resets the reference bit to 0, write the execution of the users job causes some reference bit to be set to 1.
- If the reference bit is 0 then the page has not been referenced since the last time the reference bit was set to 0.

Count Based Page Replacement

There is many other algorithms that can be used for page replacement, we can keep a counter of the number of references that has made to a page.

a) LFU (least frequently used) :
This causes the page with the smallest count to be replaced. The reason for this selection is that actively used page should have a large reference count.
This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process but never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

b) Most Frequently Used(MFU) : This is based on the principle that the page with the smallest count was probably just brought in and has yet to be used.

5.11 ALLOCATION OF FRAMES

- The allocation policy in a virtual memory controls the operating system decision regarding the amount of real memory to be allocated to each active process.
- In a paging system if more real pages are allocated, it reduces the page fault frequency and improved turnaround throughput.
- If too few pages are allocated to a process its page fault frequency and turnaround times may deteriorate to unacceptable levels.
- The minimum number of frames per process is defined by the architecture, and the maximum number of frames. This scheme is called equal allocation.
- With multiple processes competing for frames, we can classify page replacement into two broad

categories a) Local Replacement: requires that each process selects frames from only its own sets of allocated frame. b). Global Replacement: allows a process to select frame from the set of all frames. Even if the frame is currently allocated to some other process, one process can take a frame from another.

In local replacement the number of frames allocated to a process do not change but with global replacement number of frames allocated to a process do not change global replacement results in greater system throughput.

Other consideration

There is much other consideration for the selection of a replacement algorithm and allocation policy.

1) Preparing: This is an attempt to present high level of initial paging. This strategy is to bring into memory all the pages at one time. 2) TLB Reach: The TLB reach refers to the amount of memory accessible from the TLB and is simply the no of entries multiplied by page size.

3) Page Size: following parameters are considered a) page size us always power of 2 (from 512 to 16k) b) Internal fragmentation is reduced by a small page size. c) A large page size reduces the number of pages needed.

4) Invented Page table: This will reduces the amount of primary memory i,e. needed to track virtual to physical address translations. 5) Program Structure: Careful selection of data structure can increases the locality and hence lowers the page fault rate and the number of pages in working state.

6) Real time Processing: Real time system almost never has virtual memory. Virtual memory is the antithesis of real time computing, because it can introduce unexpected long term delay in the execution of a process.

5.12 THRASHING

- If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture then we suspend the process execution.
- A process is thrashing if it is spending more time in paging than executing.
- If the processes do not have enough number of frames, it will quickly page fault. During this it must replace some page that is not currently in use. Consequently it quickly faults again and again. The process continues to fault, replacing pages for which it then faults and brings back. This high paging activity is called thrashing. The phenomenon of excessively moving pages back and forth b/w memory and secondary has been called thrashing.

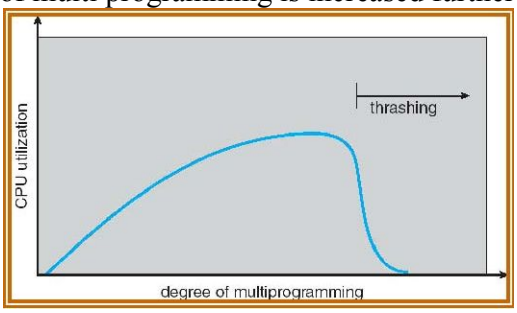
Cause of Thrashing x Thrashing results in severe performance problem. x

The operating system monitors the cpu utilization is low. We increase the degree of multi programming by introducing new process to the system.

x A global page replacement algorithm replaces pages with no regards to the process to which they belong.

The figure shows the thrashing

• As the degree of multi programming increases, more slowly until a maximum is reached. If the degree of multi programming is increased further thrashing sets in and the cpu utilization drops sharply.



• At this point, to increases CPU utilization and stop thrashing, we must increase degree of multi programming. We can limit the effect of thrashing by using a local replacement algorithm. To prevent thrashing, we must provide a process as many frames as it needs.

.

Locality of Reference: x As the process executes it moves from locality to locality. x A locality is a set of pages that are actively used. x A program may consist of several different localities, which may overlap. x Locality is caused by loops in code that find to reference arrays and other data structures by indices. The ordered list of page number accessed by a program is called reference string. Locality is of two types

1) spatial locality 2) temporal locality

Working set model

Working set model algorithm uses the current memory requirements to determine the number of page frames to allocate to the process, an informal definition is "the collection of pages that a process is working with and which must be resident if the process to avoid thrashing". The idea is to use the recent needs of a process to predict its future reader. The working set is an approximation of programs locality. Ex: given a sequence of memory reference, if the working set window size to memory references, then working set at time t1 is {1,2,5,6,7} and at t2 is changed to {3,4}

x At any given time, all pages referenced by a process in its last 4 seconds of execution are considered to compromise its working set.
x A process will never execute until its working set is resident in main memory.
x Pages outside the working set can be discarded at any movement.
Working sets are not enough and we must also introduce balance set.
a) If the sum of the working sets of all the run able process is greater than the size of memory the refuse some process for a while.
b) Divide the run able process into two groups, active and inactive. The collection of active set is called the balance set. When a process is made active its working set is loaded.
c) Some algorithm must be provided for moving process into and out of the balance set.
As a working set is changed, corresponding change is made to the balance set. Working set presents thrashing by keeping the degree of multi programming as high as possible. Thus if optimizes the CPU utilization. The main disadvantage of this is keeping track of the working set.

IMPORTANT QUESTIONS:

1. Name two differences between logical and physical addresses.
2. Explain the difference between internal and external fragmentation.
3. Describe the following allocation algorithms:
a. First fit
b. Best fit
c. Worst fit
4. Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?
5. Why are page sizes always powers of 2?
6. Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.
a. How many bits are there in the logical address?
b. How many bits are there in the physical address?
7. Why are segmentation and paging sometimes combined into one scheme?
8. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.
9. Which of the following programming techniques and structures are “good” for a demand paged environment ? Which are “not good”? Explain your answers.

a. Stack
b. Hashed symbol table
c. Sequential search
d. Binary search
e. Pure code
f. Vector operations
g. Indirection

10. Consider the following page-replacement algorithms. Rank these algorithms on a Five point scale from “bad” to “perfect” according to their page-fault rate. Separate those algorithms that suffer from Belady’s anomaly from those that do not.
a. LRU replacement
b. FIFO replacement

c. Optimal replacement d. Second-chance replacement

UNIT 6 File System Interface TOPICS

- 6 . 1 2 FILE SYSTEM: FILE CONCEPT
- 6 . 1 3 ACCESS METHODS
- 6 . 1 4 DIRECTORY STRUCTURE
- 6 . 1 5 FILE SYSTEM MOUNTING
- 6 . 1 6 FILE SHARING; PROTECTION.
- 6 . 1 7 IMPLEMENTING FILE SYSTEM
- 6 . 1 8 FILE SYSTEM STRUCTURE
- 6 . 1 9 FILE SYSTEM IMPLEMENTATION
- 6 . 2 0 DIRECTORY IMPLEMENTATION
- 6 . 2 1 ALLOCATION METHODS
- 6 . 2 2 FREE SPACE MANAGEMENT.

6 . 1 FILE SYSTEM: FILE CONCEPT

x A file is a collection of similar records. x The data can't be written on to the secondary storage unless they are within a file. x Files represent both the program and the data. Data can be numeric, alphanumeric, alphabetic or binary.

x Many different types of information can be stored on a file ---Source program, object programs, executable programs, numeric data, payroll recorder, graphic images, sound recordings and so on.

x A file has a certain defined structures according to its type:

1 Text file:-Text file is a sequence of characters organized in to lines.

2 Object file:-Object file is a sequence of bytes organized in to blocks understandable by the systems linker.

3 Executable file:-Executable file is a series of code section that the loader can bring in to memory and execute.

4 Source File:-Source file is a sequence of subroutine and function, each of which are further organized as declaration followed by executable statements.

File Attributes:-File attributes varies from one OS to other. The common file attributes are:

1 Name:-The symbolic file name is the only information kept in human readable form.

2 Identifier:-The unique tag, usually a number, identifies the file within the file system. It is the non-readable name for a file.

3 Type:-This information is needed for those systems that supports different types.

4 Location:-This information is a pointer to a device and to the location of the file on that device.

5 Size:-The current size of the file and possibly the maximum allowed size are included in this attribute.

6 Protection:-Access control information determines who can do reading, writing, execute and so on.

7 Time, data and User Identification:-This information must be kept for creation, last modification and last use. These data are useful for protection, security and usage monitoring.

File Operation:-File is an abstract data type. To define a file we need to consider the operation that can be performed on the file. Basic operations of files are:

1. Creating a file:-Two steps are necessary to create a file. First space in the file system for file is found. Second an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the file system.
2. Writing a file:-System call is mainly used for writing in to the file. System call specify the name of the file and the information i.e., to be written on to the file. Given the name the system search the entire directory for the file. The system must keep a write pointer to the location in the file where the next write to be taken place.
3. Reading a file:-To read a file system call is used. It requires the name of the file and the memory address. Again the directory is searched for the associated directory and system must maintain a read pointer to the location in the file where next read is to take place.
4. Delete a file:-System will search for the directory for which file to be deleted. If entry is found it releases all free space. That free space can be reused by another file.
5. Truncating the file:-User may want to erase the contents of the file but keep its attributes. Rather than forcing the user to delete a file and then recreate it, truncation allows all attributes to remain unchanged except for file length.

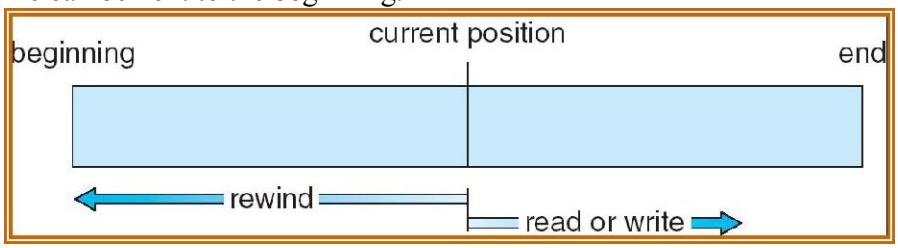
6. Repositioning within a file:-The directory is searched for appropriate entry and the current file position is set to a given value. Repositioning within a file does not need to involve actual i/o. The file operation is also known as file seeks.

In addition to this basis 6 operations the other two operations include appending new information to the end of the file and renaming the existing file. These primitives can be combined to perform other two operations. Most of the file operation involves searching the entire directory for the entry associated with the file. To avoid this OS keeps a small table containing information about an open file (the open table). When a file operation is requested, the file is specified via index in to this table. So searching is not required. Several piece of information are associated with an open file:

- x File pointer:-on systems that does not include offset an a part of the read and write system calls, the system must track the last read-write location as current file position pointer. This pointer is unique to each process operating on a file.
- x File open count:-As the files are closed, the OS must reuse its open file table entries, or it could run out of space in the table. Because multiple processes may open a file, the system must wait for the last file to close before removing the open file table entry. The counter tracks the number of copies of open and closes and reaches zero to last close.
- x Disk location of the file:-The information needed to locate the file on the disk is kept in memory to avoid having to read it from the disk for each operation. x Access rights:-Each process opens a file in an access mode. This information is stored on per-process table the OS can allow OS deny subsequent i/o request.

6.2 **ACCESS METHODS** The information in the file can be accessed in several ways. Different file access methods are:

1. **Sequential Access:**Sequential access is the simplest access method. Information in the file is processed in order, one record after another. Editors and compilers access the files in this fashion. Normally read and write operations are done on the files. A read operation reads the next portion of the file and automatically advances a file pointer, which track next i/I track. Write operation appends to the end of the file and such a file can be next to the beginning.



Sequential access depends on a tape model of a file.

2. **Direct Access:**

Direct access allows random access to any file block. This method is based on disk model of a file.A file is made up of fixed length logical records. It allows the program to read and write records rapidly in any order.A direct access file allows arbitrary blocks to be read or written. *Eg*:-User may need block 13, then read block 99 then write block 12. For searching the records in large amount of information with immediate result, the direct access method is suitable. Not all OS support sequential and direct access. Few OS use sequential access and some OS uses direct access. It is easy to simulate sequential access on a direct access but the reverse is extremely inefficient.

Indexing Method:x The index is like an index at the end of a book which contains pointers to various blocks. x To find a record in a file, we search the index and then use the pointer to access the file directly and to find the desired record. With large files index file itself can be very large to be kept in memory. One solution to create an index to the index files itself. The primary index file would contain pointer to secondary index files which would point to the actual data items. Two types of indexes can be used:

a. Exhaustive index:-Contain one entry for each of record in the main file. An index itself is organized as a sequential file.

b. Partial index:-Contains entries to records where the field of interest exists with records of variable length, soma record will not contain an fields. When a new record is added to the main file, all index files must be updated.

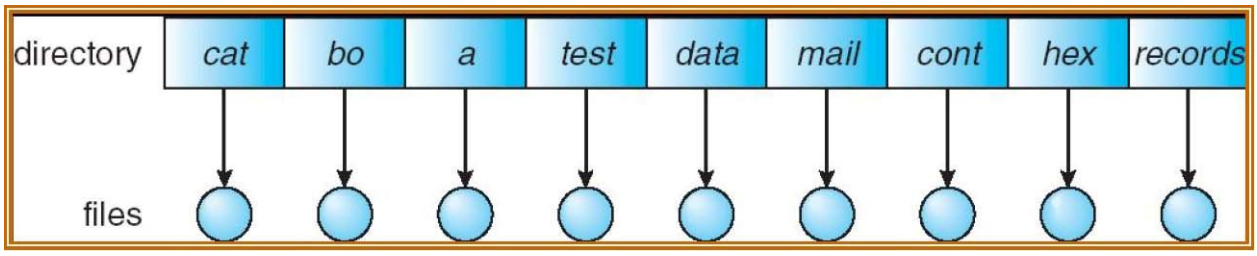
6 . 3 DIRECTORY STRUCTURE

The files systems can be very large. Some systems stores millions of files on the disk. To manage all this data we need to organize them. This organization is done in two parts:

1. Disks are split in to one or more partition also known as minidisks.
2. Each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents. The device directory or simple directory records information as name, location, size, type for all files on the partition. The directory can be viewed as a symbol table that translates the file names in to the directory entries. The directory itself can be organized in many ways. When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory.

x Search for a file:-Directory structure is searched for finding particular file in the directory. Files have symbolic name and similar name may indicate a relationship between files, we may want to be able to find all the files whose name match a particular pattern.
x Create a file:-New files can be created and added to the directory.
x Delete a file:-when a file is no longer needed, we can remove it from the directory.
x List a directory:-We need to be able to list the files in directory and the contents of the directory entry for each file in the list.
x Rename a file:-Name of the file must be changeable when the contents or use of the file is changed. Renaming allows the position within the directory structure to be changed.
x Traverse the file:-it is always good to keep the backup copy of the file so that or it can be used when the system gets fail or when the file system is not in use.

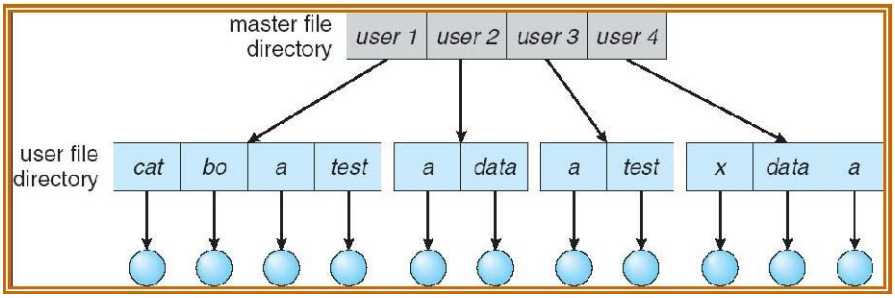
1. **Single-level directory**:This is the simplest directory structure. All the files are contained in the same directory which is easy to support and understand.



Disadvantage:x Not suitable for a large number of files and more than one user. x Because of single directory files, files require unique file names. x Difficult to remember names of all the files as the number of files
increases. MS-DOS OS allows only 11 character file name where as UNIX allows 255 character.

2. **Two-level directory**:x A single level directory often leads to the confusion of file names between different users. The solution here is to create separate directory or each user.
x In two level directories each user has its own directory. It is called User File Directory (UFD). Each UFD has a similar structure, but lists only the files of a single user.
x When a user job starts or users logs in, the systems Master File Directory (MFD) is searched. The MFD is indexed by the user name or account number and each entry points to the UFD for that user.
x When a user refers to a particular file, only his own UFD is searched. Thus different users may have files with the same name.

x To create a file for a user, OS searches only those users UFD to as certain whether another file of that name exists.
x To delete a file checks in the local UFD so that accidentally delete another user's file with the same name.



Although two-level directories solve the name collision problem but it still has some disadvantage. This structure isolates one user from another. This isolation is an advantage. When the users are independent but disadvantage, when some users want to co-operate on some table and to access one another file.

3. **Tree-structured directories**:MS-DOS use Tree structure directory. It allows users to create their own subdirectory and to organize their files accordingly. A subdirectory contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. The entire directory will have the same internal format. One bit in each entry defines the entry as a file (0) and as a subdirectory (1). Special system calls are used to create and delete directories. In normal use each uses has a current directory. Current directory should contain most of the files that are of the current interest of users. When a reference to a file is needed the current directory is searched. If file is needed i.e., not in the current directory to be the directory currently holding that file.
Path name can be of two types:

a. Absolute path name:-Begins at the root and follows a path down to the specified file, giving the directory names on the path.

b. Relative path name:-Defines a path from the current directory. One important policy in this structure is low to handle the deletion of a directory.

a. If a directory is empty, its entry can simply be deleted.
b. If a directory is not empty, one of the two approaches can be used.
   i. In MS-DOS, the directory is not deleted until it becomes empty.
   ii. In UNIX, RM command is used with some options for deleting directory.

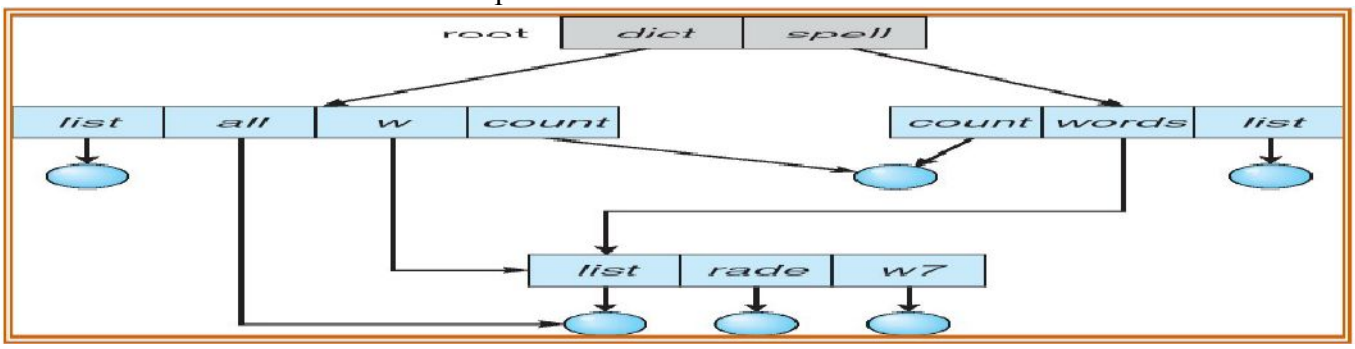
4. **Acyclic graph directories**:x It allows directories to have shared subdirectories and files. x

Same file or directory may be in two different directories.
x

A graph with no cycles is a generalization of the tree structure subdirectories scheme. x Shared files and subdirectories can be implemented by using links. x

A link is a pointer to another file or a subdirectory.

A link is implemented as absolute or relative path.
An acyclic graph directory structure is more flexible then is a simple tree structure but sometimes it is more complex.



6 . 4 FILE SYSTEM MOUNTING

The file system must be mounted before it can be available to processes on the system The procedure for mounting the file is:

a. The OS is given the name of the device and the location within the file structure at which to attach the file system (mount point).A mount point will be an empty directory at which the mounted file system will be attached.

Eg:-On UNIX a file system containing users home directory might be mounted as /home then to access the directory structure within that file system. We must precede the directory names as /home/jane.

b. Then OS verifies that the device contains this valid file system. OS uses device drivers for this verification.

c. Finally the OS mounts the file system at the specified mount point.

6.5 FILE SHARING,PROTECTION.

Disks provide bulk of secondary storage on which the file system is maintained. Disks have two characteristics:

a. They can be rewritten in place i.e., it is possible to read a block from the disk to modify the block and to write back in to same place.

b. They can access any given block of information on the disk. Thus it is simple to access any file either sequentially or randomly and switching from one file to another.

The lowest level is the i/o control consisting of device drivers and interrupt handless to transfer the information between memory and the disk system. The device driver is like a translator. Its input is a high level command and the o/p consists of low level hardware specific instructions, which are used by the hardware controllers which interface I/O device to the rest of the system. The basic file system needs only to issue generic commands to the appropriate device drivers to read and write physical blocks on the disk. The file organization module knows about files and their logical blocks as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file organization module can translate logical block address to the physical block address. Each logical block is numbered 0 to N. Since the physical blocks containing the data usually do not match the logical numbers, So a translation is needed to locate each block. The file allocation modules also include free space manager which tracks the unallocated blocks and provides these blocks when requested.

6.6 IMPLEMENTING FILE SYSTEM

x File system is implemented on the disk and the memory. x The implementation of the file system varies according to the OS and the file system, but there are some general principles. If the file system is implemented on the disk it contains the following information:

a. **Boot Control Block**:-can contain information needed by the system to boot an OS from that partition. If the disk has no OS, this block is empty. It is the first block of the partition. In UFS→boot block, In NTFS→partition boot sector.

b. **Partition control Block**:-contains partition details such as the number of blocks in partition, size of the blocks, number of free blocks, free block pointer, free FCB count and FCB pointers. In NTFS→master file tables, In UFS→super block.

c. Directory structure is used to organize the files.

d. An FCB contains many of the files details, including file permissions, ownership, size, location of the data blocks. In UFS→inode, In NTFS this information is actually stored within master file table.

Structure of the file system management in memory is as follows:

a. An in-memory partition table containing information about each mounted information.

b. An in-memory directory structure that holds the directory information of recently accessed directories.

c. The system wide open file table contains a copy of the FCB of each open file as well as other information.

d. The per-process open file table contains a pointer to the appropriate entry in the system wide open file table as well as other information.

6.7 FILE SYSTEM STRUCTURE

To provide efficient and convenient access to the disks, the OS provides the file system to allow the data to be stored, located and retrieved. A file system has two design problems:

a. How the file system should look to the user.

b. Selecting algorithms and data structures that must be created to map logical file

system on to the physical secondary storage devices. The file system itself is composed of different levels. Each level uses the feature of the lower levels to create new features for use by higher levels. The following structures shows an example of layered design

The lowest level is the i/o control consisting of device drivers and interrupt handless to transfer the information between memory and the disk system. The device driver is like a translator. Its input is a high level command and the o/p consists of low level hardware specific instructions, which are used by the hardware controllers which interface I/O device to the rest of the system. The basic file system needs only to issue generic commands to the appropriate device drivers to read and write physical blocks on the disk. The file organization module knows about files and their logical blocks as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file organization module can translate logical

block address to the physical block address. Each logical block is numbered 0 to N. Since the physical blocks containing the data usually do not match the logical numbers, So a translation is needed to locate each block. The file allocation modules also include free space manager which tracks the unallocated blocks and provides these blocks when requested. The logical file system uses the directory structure to provide the file organization module with the information, given a symbolic file name. The logical file system also responsible for protection and security. Logical file system manages metadata information. Metadata includes all the file system structures excluding the actual data. The file structure is maintained via file control block (FCB). FCB contains information about the file including the ownership permission and location of the file contents.

6.8 FILE SYSTEM IMPLEMENTATION

x File system is implemented on the disk and the memory. x The implementation of the file system varies according to the OS and the file system, but there are some general principles. If the file system is implemented on the disk it contains the following information:
e. **Boot Control Block**:-can contain information needed by the system to boot an OS from that partition. If the disk has no OS, this block is empty. It is the first block of the partition. In UFS→boot block, In NTFS→partition boot sector.
f. **Partition control Block**:-contains partition details such as the number of blocks in partition, size of the blocks, number of free blocks, free block pointer, free FCB count and FCB pointers. In NTFS→master file tables, In UFS→super block.
g. Directory structure is used to organize the files.
h. An FCB contains many of the files details, including file permissions, ownership, size, location of the data blocks. In UFS→inode, In NTFS this information is actually stored within master file table.

Structure of the file system management in memory is as follows:
e. An in-memory partition table containing information about each mounted information.
f. An in-memory directory structure that holds the directory information of recently accessed directories.
g. The system wide open file table contains a copy of the FCB of each open file as well as other information.
h. The per-process open file table contains a pointer to the appropriate entry in the

system wide open file table as well as other information. A typical file control blocks is shown below

File Permission
File dates (create, access, write)
File owner, group, acc
File size
File data blocks

Partition and Mounting:x A disk can be divided in to multiple partitions. Each partition can be either raw i.e., containing no file system and cooked i.e., containing a file system. x Raw disk is used where no file system is appropriate. UNIX swap space can use a raw partition and do not use file system.

x Some db uses raw disk and format the data to suit their needs. Raw disks can hold

information need by disk RAID (Redundant Array of Independent Disks) system.

x Boot information can be stored in a separate partition. Boot information will have their own format. At the booting time system does not load any device driver for the file system. Boot information is a sequential series of blocks, loaded as an image in to memory.

x Dual booting is also possible on some Pc's, more than one OS are loaded on a system. A boot loader understands multiple file system and multiple OS can occupy the boot space once loaded it can boot one of the OS available on the disk. The disks can have multiple portions each containing different types of file system and different types of OS. Root partition contains the OS kernel and is mounted at a boot time. Microsoft window based systems mount each partition in a separate name space denoted by a letter and a colon. On UNIS file system can be mounted at any directory.

6 . 9 DIRECTORY IMPLEMENTATION

Directory is implemented in two ways:

1. **Linear list:**x Linear list is a simplest method. x It uses a linear list of file names with pointers to the data blocks. x Linear list uses a linear search to find a particular entry. x Simple for programming but time consuming to execute. x For creating a new file, it searches the directory for the name whether same name already exists. x Linear search is the main disadvantage. x Directory implementation is used frequently and uses would notice a slow implementation of access to it.

2. **Hash table:**x Hash table decreases the directory search time. x Insertion and deletion are fairly straight forward. x Hash table takes the value computed from that file name. x Then it returns a pointer to the file name in the linear list. x Hash table uses fixed size.

6.10 ALLOCATION METHODS

The space allocation strategy is closely related to the efficiency of the file accessing and of logical to physical mapping of disk addresses. A good space allocation strategy must take in to consideration several factors such as:

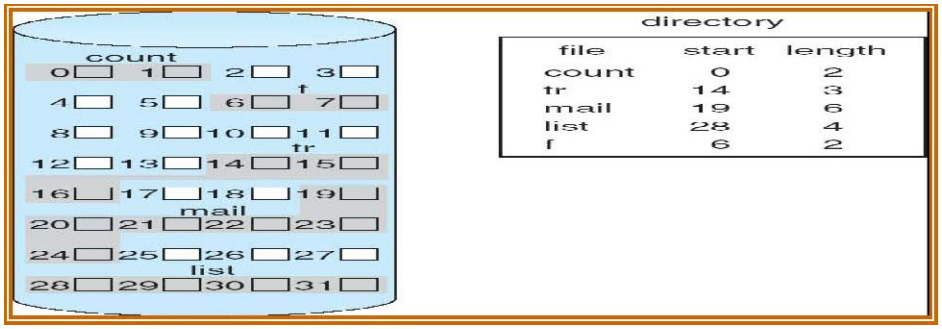
1. Processing speed of sequential access to files, random access to files and allocation and de-allocation of blocks.
2. Disk space utilization.
3. Ability to make multi-user and multi-track transfers.

4. Main memory requirement of a given algorithm.

Three major methods of allocating disk space is used.

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

file. The figure shows the contiguous allocation method.



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 file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation. Contiguous allocation algorithm suffers from external fragmentation. Depending on the amount of disk storage the external fragmentation can be a major or minor problem. Compaction is used to solve the problem of external fragmentation. The following figure shows the contiguous allocation of space after compaction. The original disk was then freed completely creating one large contiguous space. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2…….b+n

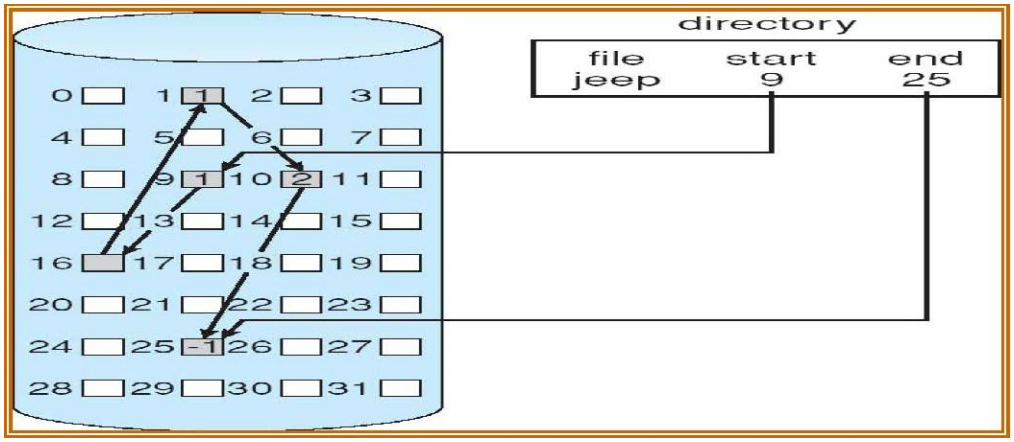
1. The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in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

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

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

Disadvantage:x Pre-allocation is required. x It suffers from external fragmentation.

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



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

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

Advantages:x No external fragmentation. x Compaction is never required. x Pre-allocation is not required.

Disadvantage:x Files are accessed sequentially. x

Space required for pointers.

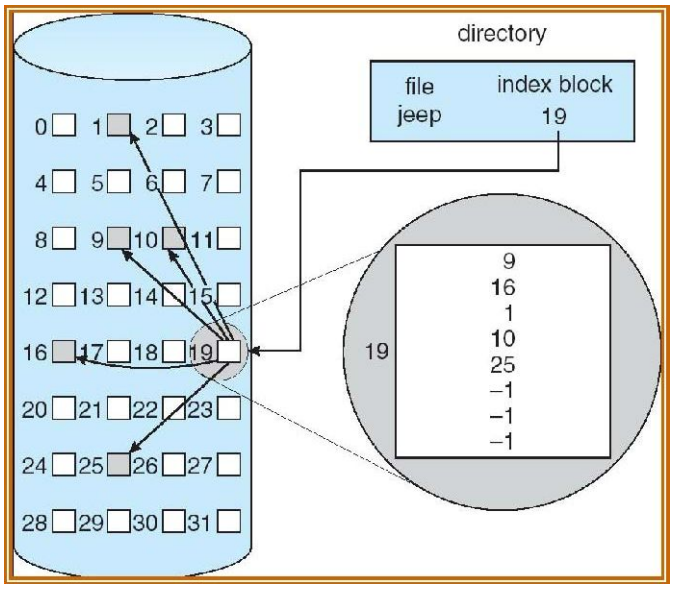
x

Reliability is not good.

x

Cannot support direct access.

3. **Indexed Allocation:**The file allocation table contains a separate one level index for each file. The index has one entry for each portion allocated to the file. The i th entry in the index block points to the i th block of the file. The following figure shows indexed allocation.



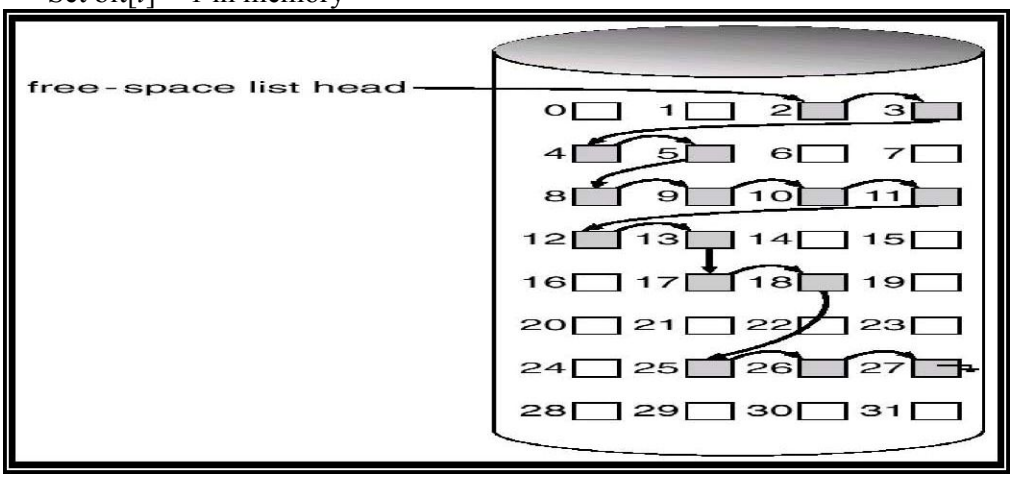
The indexes are not stored as a part of file allocation table rather than the index is kept as a separate block and the entry in the file allocation table points to that block. Allocation can be made on either fixed size blocks or variable size blocks. When the file is created all pointers in the index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

Advantages:-
Supports both sequential and direct access.
No external fragmentation. Faster then other two methods.
Supports fixed size and variable sized blocks.

Disadvantage:-
Suffers from wasted space.
Pointer overhead is generally greater

6.11 FREE SPACE MANAGEMENT

- Need to protect:
  - Pointer to free list
- Bit map
  - Must be kept on disk
  - Copy in memory and disk may differ.
  - Cannot allow for block[i] to have a situation where bit[i] = 1 in memory and bit[i] = 0 on disk.
- Solution:
  - Set bit[i] = 1 in disk.
  - Allocate block[i]
  - Set bit[i] = 1 in memory



- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

IMPORTANT QUESTIONS:

1. Explain the purpose of the open and close operations.
2. Give an example of an application in which data in a file should be accessed in the following order:

a. Sequentially
b. Randomly

3. In some systems, a subdirectory can be read and written by an authorized user, just as ordinary files can be.

a. Describe the protection problems that could arise.
b. Suggest a scheme for dealing with each of the protection problems you named in part a.

4. What is a file?
5. List sample file types, based on use, on the VAX under VMS.

6. What is a sequential file?
7. Can a direct access file be read sequentially? Explain.
8. List two types of system directories
9. List operations to be performed on directories
10. How do we overcome the disadvantages of the two-level directory?

UNIT 7 Mass Storage Structure

TOPICS

7.15 MASS STORAGE STRUCTURES
7.16 DISK STRUCTURE
7.17 DISK ATTACHMENT
7.18 DISK SCHEDULING
7.19 DISK MANAGEMENT
7.20 SWAP SPACE MANAGEMENT
7.21 PROTECTION: GOALS OF PROTECTION
7.22 PRINCIPLES OF PROTECTION
7.23 DOMAIN OF PROTECTION
7.24 ACCESS MATRIX
7.25 IMPLEMENTATION OF ACCESS MATRIX
7.26 ACCESS CONTROL
7.27 REVOCATION OF ACCESS RIGHTS
7.28 CAPABILITY-BASED SYSTEM

7.1 MASS STORAGE STRUCTURES

x Disks provide a bilk of secondary storage. Disks come in various sizes, speed and information can be stored optically or magnetically. x Magnetic tapes were used early as secondary storage but the access time is less than

disk. x Modern disks are organized as single one-dimensional array of logical blocks. x The actual details of disk i/o open depends on the computer system, OS, nature of i/o channels and disk controller hardware.

x The basic unit of information storage is a sector. The sectors are stored on flat, circular, media disk. This disk media spins against one or more read-write heads. The head can move from the inner portion of the disk to the outer portion.

x When the disk drive is operating the disks is rotating at a constant speed. x To read or write the head must be positioned at the desired track and at the beginning if the desired sector on that track. x Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed head system. x These characteristics are common to floppy disks, hard disks, CD-ROM and DVD.

7.2 DISK STRUCTURE

1. Seek Time:-Seek time is the time required to move the disk arm to the required track.
Seektimecanbegivenby Ts=m*n+s. Where Ts = seek time
n = number of track traversed. m = constant that depends on the disk drive s = startup time.
2. Rotational Latency:-Rotational latency is the additional addition time for waiting for the disk to rotate to the desired sector to the disk head.

3. Rotational Delay:-Disks other than the floppy disk rotate at 3600 rpm which is one revolution per 16.7ms.

4. Disk Bandwidth:-Disk bandwidth is the total number of bytes transferred divided by

total time between the first request for service and the completion of last transfer. Transfer time = T =b / rN Where b = number of bytes transferred.
T = transfer time. r = rotational speed in RpS. N = number of bytes on the track.
Average access time = Ta = Ts + 1/ 2r + b/ rN
Where Ts = seek time.

4. Total capacity of the disk:-It is calculated by using following formula. Number of cylinders * number of heads * number of sector/track * number of bytes/sector.

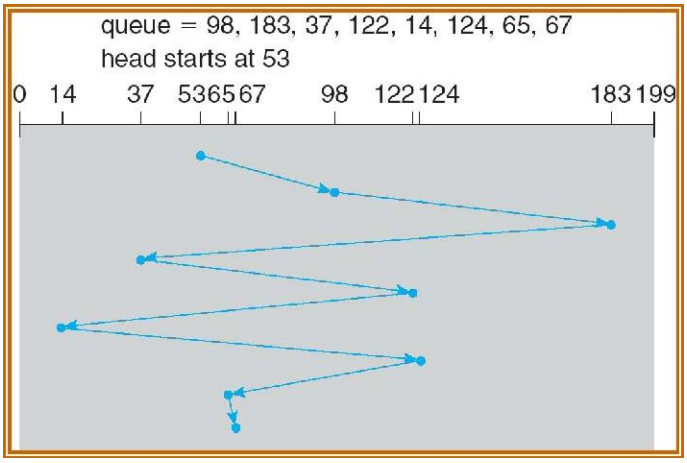
7.3 DISK ATTACHMENT

The amount of head movement needed to satisfy a series of i/o request can affect the performance. If the desired drive and the controller are available the request can be serviced immediately. If the device or controller is busy any new requests for service will be placed on the queue of pending requests for that drive when one request is complete the OS chooses which pending request to service next.

7.4 DISK SCHEDULING

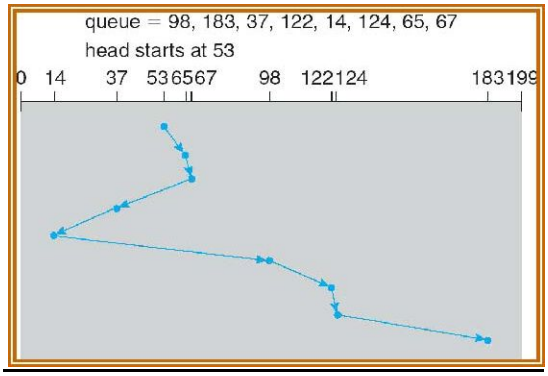
Different types of scheduling algorithms are as follows:

1. **FCFS scheduling algorithm**:This is the simplest form of disk scheduling algorithm. This services the request in the order they are received. This algorithm is fair but do not provide fastest service. It takes no special time to minimize the overall seek time. *Eg:*-consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67



If the disk head is initially at 53, it will first move from 53 to 98 then to 183 and then to 37, 122, 14, 124, 65, 67 for a total head movement of 640 cylinders. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together before or after 122 and 124 the total head movement could be decreased substantially and performance could be improved.

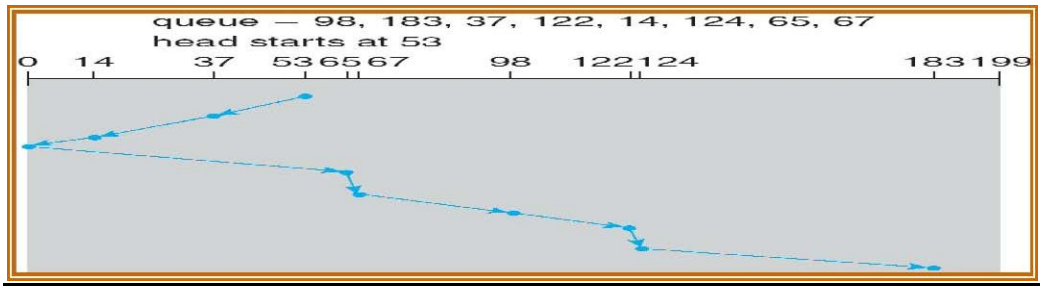
2. **SSTF** (Shortest seek time first) **algorithm**:This selects the request with minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by head, SSTF chooses the pending request closest to the current head position. *Eg*:-*:*-consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67



If the disk head is initially at 53, the closest is at cylinder 65, then 67, then 37 is closer then 98 to 67. So it services 37, continuing we service 14, 98, 122, 124 and finally 183. The total head movement is only 236 cylinders. SSTF is essentially a form of SJF and it may cause starvation of some requests. SSTF is a

substantial improvement over FCFS, it is not optimal.

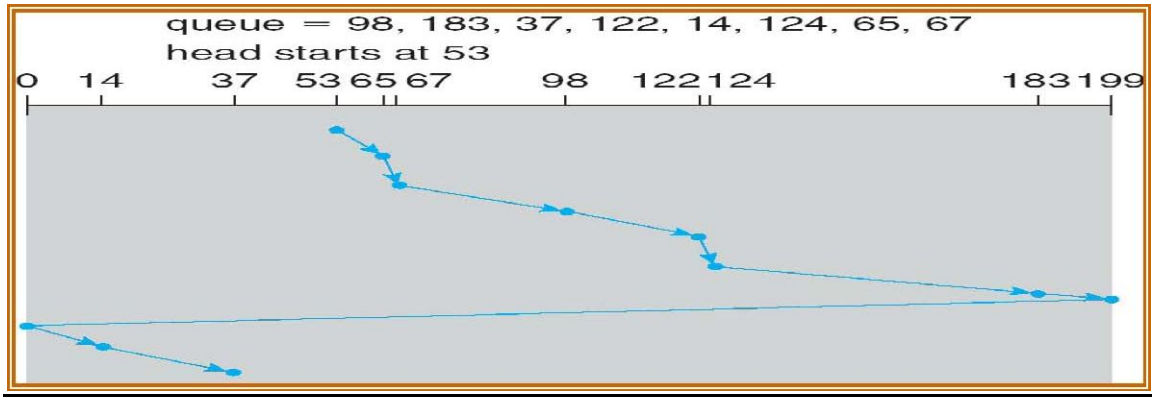
3. **SCAN algorithm**:In this the disk arm starts at one end of the disk and moves towards the other end, servicing the request as it reaches each cylinder until it gets to the other end of the disk. At the other end, the direction of the head movement is reversed and servicing continues. *Eg*:-:-consider a disk queue with request for i/o to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67



If the disk head is initially at 53 and if the head is moving towards 0, it services 37 and then 14. At cylinder 0 the arm will reverse and will move towards the other end of the disk servicing 65, 67, 98, 122, 124 and 183. If a request arrives just in from of head, it will be serviced immediately and the request just behind the head will have to wait until the arms reach other end and reverses direction. The SCAN is also called as elevator algorithm.

4. **C-SCAN** (Circular scan) **algorithm**:

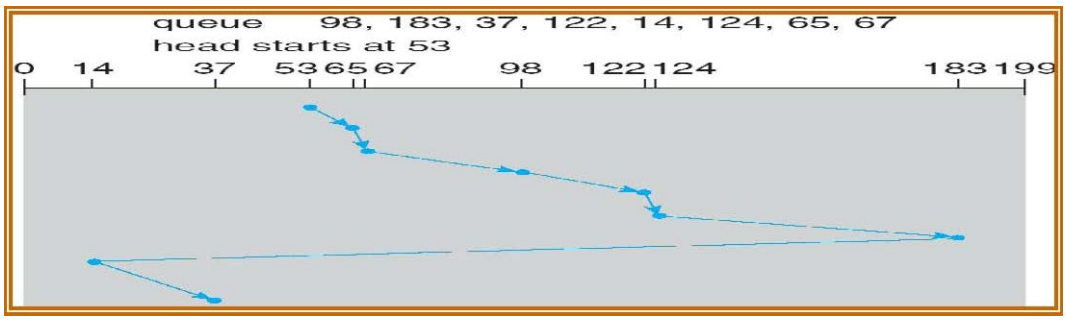
C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from end of the disk to the other servicing the request along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any request on the return. The C-SCAN treats the cylinders as circular list that wraps around from the final cylinder to the first one. *Eg*:-



5.Look Scheduling algorithm:

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In practice neither of the algorithms is implemented in this way. The arm goes only as far as the final request in each direction. Then it reverses, without going all the way to the end of the disk. These versions of SCAN

and CSCAN are called Look and C-Look scheduling because they look for a request before continuing to move in a given direction. *Eg*:



Selection of Disk Scheduling Algorithm:

1. SSTF is common and it increases performance over FCFS.
2. SCAN and C-SCAN algorithm is better for a heavy load on disk.
3. SCAN and C-SCAN have less starvation problem.
4. SSTF or Look is a reasonable choice for a default algorithm.

7.4 DISK MANAGEMENT

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

7.5 SWAP SPACE MANAGEMENT

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

RAID Structure

➓ Disk striping uses a group of disks as one storage unit schemes improve performance and improve the reliability of the storage system by storing redundant data

- Mirroring or shadowing (RAID 1) keeps duplicate of each disk
- Striped mirrors (RAID 1+0) or mirrored stripes (RAID 0+1) provides high performance and high reliability
- Block interleaved parity (RAID 4, 5, 6) uses much less redundancy

- RAID within a storage array can still fail if the array fails, so automatic replication of the data between arrays is common.Frequently, a small number of hot-spare disks are left unallocated, automatically replacing a failed disk and having

data rebuilt onto them

- RAID – multiple disk drives provides reliability via redundancy. It is arranged into six different levels.Several improvements in disk-use techniques involve the use of multiple disks working cooperatively.Disk striping uses a group of disks as one storage unit.
- ➓ RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
  - Mirroring or shadowing keeps duplicate of each disk.
  - Block interleaved parity uses much less redundancy.
- RAID alone does not prevent or detect data corruption or other errors, just disk failures .Solaris ZFS adds checksums of all data and metadata Checksums kept with pointer to object, to detect if object is the right one and whether it changed can detect and correct data and metadata corruptionZFS also removes volumes, partititions .Disks allocated in pools
- Filesystems with a pool share that pool, use and release space like “malloc” and “free” memory allocate / release calls

7.6 PROTECTION: GOALS OF PROTECTION

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access.Examine capability and language-based protection systems
- Operating system consists of a collection of objects, hardware or software Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem -ensure that each object is accessed correctly and only by those processes that are allowed to do so

7.6 PRINCIPLES OF PROTECTION

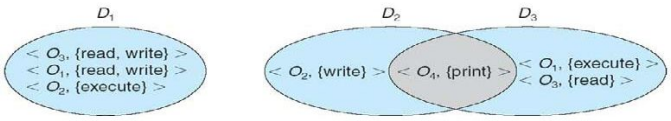
- Guiding principle principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks **Disk Attachment :** Stable-Storage Implementation

- Write-ahead log scheme requires stable storage
- • To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery

7.8 DOMAIN OF PROTECTION

Domain Structure

Access-right=<object-name,rights-set> where *rights-set* is a subset of all valid operations that can be performed on the object.



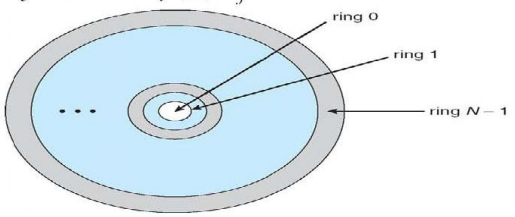
Domain Implementation (UNIX)

- System consists of 2 domains:
  - User
  - Supervisor
- • UNIX
  - Domain = user-id
  - Domain switch accomplished via file system

- Each file has associated with it a domain bit (setuid bit)
- When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset •

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < I \Rightarrow D_i \subseteq D_j$



7.9 ACCESS MATRIX

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- *Access(i, j)* is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$

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

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- • Can be expanded to dynamic protection
- – Operations to add, delete access rights
- – Special access rights: • owner of O_i
  - copy op from O_i to O_j
  - control – D_i can modify D_j access rights
  - transfer – switch from domain D_i to D_j
- • Access matrix design separates mechanism from policy
- – Mechanism
  - Operating system provides access-matrix + rules
  - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
- – Policy
  - User dictates policy
  - Who can access what object and in what mode

•

7.10 IMPLEMENTATION OF ACCESS MATRIX

• Each column = Access-control list for one object Defines who can perform what operation.

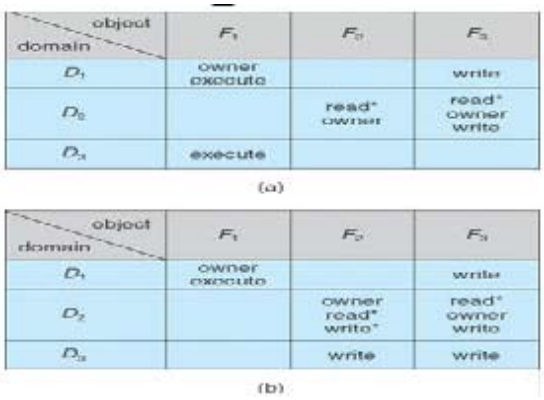
Domain 1 = Read, Write, Domain 2 Read Domain 3 Read

• Each Row = Capability List (like a key)

Fore each domain, what operations allowed on what objects. Object 1 – Read Object 4 – Read, Write, Execute
Object 5 – Read, Write, Delete, Copy

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

ACCESS MATRIX OF FIGURE A WITH DOMAINS AS OBJECTS



ACCESS MATRIX WITH *COPY* RIGHTS

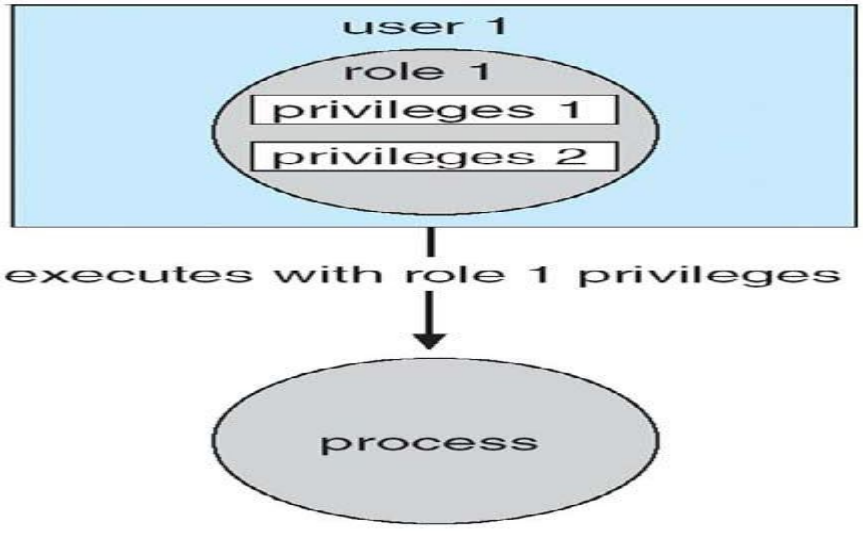
object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

MODIFIED ACCESS MATRIX OF FIGURE B

7.11 ACCESS CONTROL

Protection can be applied to non-file resources. Solaris 10 provides role-based access control **(**RBAC**)** to implement least privilege.Privilege is right to execute system call or use an option within a system call Can be assigned to processes. Users assigned roles granting access to privileges and Programs. Revocation of Access Rights

• Access List – Delete access rights from access list
- Simple
- Immediate
- • Capability List – Scheme required to locate capability in the system before capability can be revoked
- Reacquisition
- Back-pointers
- Indirection
- Keys



- Low cost is the defining characteristic of tertiary storage
- Generally, tertiary storage is built using removable media

- Common examples of removable media are floppy disks and CD-ROMs; other types are available

Removable Disks

• Floppy disk — thin flexible disk coated with magnetic material, enclosed in a protective plastic case
  - Most floppies hold about 1 MB; similar technology is used for removable disks that hold more than 1 GB
  - Removable magnetic disks can be nearly as fast as hard disks, but they are at a greater risk of damage from exposure
- • A magneto-optic disk records data on a rigid platter coated with magnetic material
  - Laser heat is used to amplify a large, weak magnetic field to record a bit
  - Laser light is also used to read data (Kerr effect)
  - The magneto-optic head flies much farther from the disk surface than a magnetic disk head, and the magnetic material is covered with a protective layer of plastic or glass; resistant to head crashes
- Optical disks do not use magnetism; they employ special materials that are altered by laser light

WORM Disks

- The data on read-write disks can be modified over and over
- WORM ("Write Once, Read Many Times") disks can be written only once
- Thin aluminum film sandwiched between two glass or plastic platters
- To write a bit, the drive uses a laser light to burn a small hole through the aluminum; information can be destroyed by not altered
- Very durable and reliable
- Read-only disks, such ad CD-ROM and DVD, com from the factory with the data pre-recorded Tapes
- Compared to a disk, a tape is less expensive and holds more data, but random access is much slower
- Tape is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data
- • Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library
  - stacker – library that holds a few tapes
  - silo – library that holds thousands of tapes
- A disk-resident file can be archived to tape for low cost storage; the computer can stage it back into disk storage for active use

Application Interface

- Most OSs handle removable disks almost exactly like fixed disks — a new cartridge is formatted and an empty file system is generated on the disk
- Tapes are presented as a raw storage medium, i.e., and application does not not open a file on the tape, it opens the whole tape drive as a raw device
- Usually the tape drive is reserved for the exclusive use of that application
- Since the OS does not provide file system services, the application must decide how to use the array of blocks
- Since every application makes up its own rules for how to organize a tape, a tape full of data can generally only be used by the program that created it

7.12 CAPABILITY-BASED SYSTEM

• Hydra
- Fixed set of access rights known to and interpreted by the system
- Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights

• • Cambridge CAP System
- Data capability -provides standard read, write, execute of individual storage segments associated with object
- Software capability -interpretation left to the subsystem, through its protected procedures

• Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
• Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable

• Protection in Java 2
• Protection is handled by the Java Virtual Machine (JVM)
• A class is assigned a protection domain when it is loaded by the JVM
• The protection domain indicates what operations the class can (and cannot) perform
• If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library
• Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

untrusted applet	URL loader	networking
none	*.lucent.com:80, connect	any

gui:
...
get(url);
open(addr);
...
	get(URL u):
	...
	doPrivileged {
	open('proxy.lucent.com:80');
	}
	<request u from proxy>
	...
		open(Addr a):
		...
		checkPermission
		(a, connect);
		connect (a);
		...

IMPORTANT QUESTIONS

What are the main differences between capability lists and access lists?

a. What hardware features are needed for efficient capability manipulation? Can these be used for memory protection? What protection problems may arise if a shared stack is used for parameter passing?
b. What is the need-to-know principle? Why is it important for a protection systemto adhere to this principle? Why is it difficult to protect a system in which users are allowed to do their own I/O?
c. Describe how the Java protection model would be sacrificed if a Java program were allowed to directly alter the annotations of its stack frame. What are two advantages of encrypting data stored in the computer system.

UNIT 8 Linux system

TOPICS

8.1 LINUX HISTORY

- To explore the history of the UNIX operating system from which Linux is derived and the principles which Linux is designed upon
- To examine the Linux process model and illustrate how Linux schedules processes and provides interprocess communication
- To look at memory management in Linux
- To explore how Linux implements file systems and manages I/O devices

History

- Linux is a modern, free operating system based on UNIX standards
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility
- Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet
- It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platform.

The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code. The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.

☐ Version 0.01 (May 1991) had no networking, ran only on 80386compatible Intel pr Many, varying **Linux Distributions** including the kernel, applications, and management tools
ocessors and on PC hardware, had extremely limited device-drive support, and supported only the Minix file system
Linux 1.0 (March 1994) included these new features:

Support for UNIX's standard TCP/IP networking protocols

- BSD-compatible socket interface for networking programming
- Device-driver support for running IP over an Ethernet
- Enhanced file system
- Support for a range of SCSI controllers for high-performance disk access

Version 1.2 (March 1995) was the final PC-only Linux kernel

☐ Released in June 1996, 2.0 added two major new capabilities:

☐ Support for multiple architectures, including a fully 64-bit native Alpha port

☐ Support for multiprocessor architectur
☐ Other new features included:

- Improved memory-management code
- Improved TCP/IP performance
- Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand

Standardized configuration interface

☐ Available for Motorola 68000-series processors, Sun Sparc systems, and for PC and PowerMac systems.2.4 and 2.6 increased SMP support, added journaling file system, preemptive kernel, 64-bit memory support

8.2 DESIGN PRINCIPLES Linux is a multiuser, multitasking system with a full set of UNIX-

compatible tools

- Its file system adheres to traditional UNIX semantics, and it fully

implements the standard UNIX networking model

- Main design goals are speed, efficiency, and standardization
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification
- The Linux programming interface adheres to the SVR4 UNIX

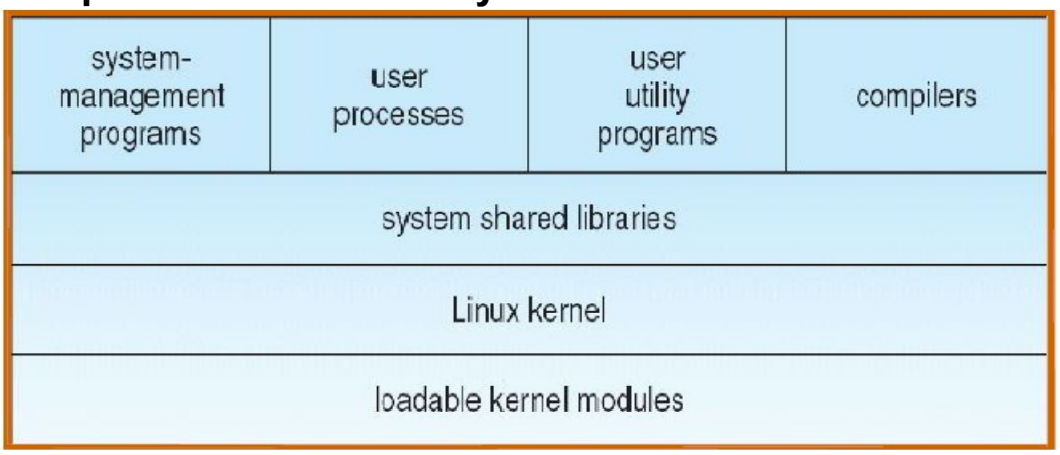
semantics, rather than to BSD behavior Like most UNIX implementations, Linux is composed

of three main bodies of code; the most important distinction between the kernel and all other components

x Like most UNIX implementations, Linux is composed of three main bodies of code; the most important distinction between the kernel and all other components x The **kernel** is responsible for maintaining the important abstractions of the operating system

- o Kernel code executes in *kernel mode* with full access to all the physical resources of the computer
- o All kernel code and data structures are kept in the same single address space

Components of a Linux System



x The **system libraries** define a standard set of functions through which applications interact with the kernel, and which implement much of the operating-system functionality that does not need the full privileges of kernel code

x The **system utilities** perform individual specialized management tasks x Sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel

8.3 KERNEL MODULES

x A kernel module may typically implement a device driver, a file system, or a networking protocol

x The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL

x Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in

x Three components to Linux module support:

- o module management
- o driver registration
- o conflict resolution
- o MODULE MANAGEMENT
- o Supports loading modules into memory and letting them talk to the rest of the kernel

- o Module loading is split into two separate sections:
- o Managing sections of module code in kernel memory
- o Handling symbols that modules are allowed to reference
- o The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed
- o **Driver register**
- o Allows modules to tell the rest of the kernel that a new driver has become available
- o The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to be added to or removed from these tables at any time
- o Registration tables include the following items:
- o Device drivers
- o File systems
- o Network protocols
- o Binary format
- o **Conflict Resolution**
- o A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver
- o The conflict resolution module aims to:
- o Prevent modules from clashing over access to hardware resources
- o Prevent *autoprobes* from interfering with existing device drivers
- o Resolve conflicts with multiple drivers trying to access the same hardware

8.4 PROCESS MANAGEMENT

- o UNIX process management separates the creation of processes and the running of a new program into two distinct operations.
- o The **fork** system call creates a new process
- o A new program is run after a call to **execve**
- o Under UNIX, a process encompasses all the information that the operating system must maintain *t* track the context of a single execution of a single program
- o Under Linux, process properties fall into three groups: the process's identity, environment, and context
- o **Process Identity**
- o Process ID (PID). The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process
- o Credentials. Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files
- o Personality. Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls
- o Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX
- o **Process Environment**
- o The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
- o The argument vector lists the command-line arguments used to invoke the running program;

conventionally starts with the name of the program itself

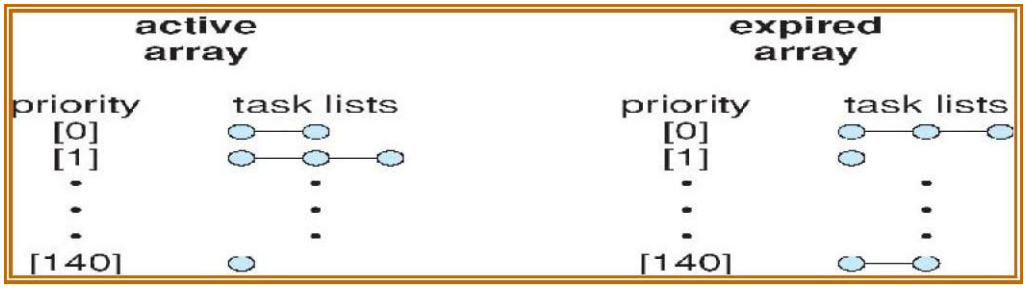
- The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values
- Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software
- The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole
- **Process Context**
- The (constantly changing) state of a running program at any point in time
- The **scheduling context** is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process
- The kernel maintains **accounting** information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far
- The **file table** is an array of pointers to kernel file structures
- When making file I/O system calls, processes refer to files by their index into this table
- Whereas the file table lists the existing open files, the **file-system context** applies to requests to open new files
- The current root and default directories to be used for new file searches are stored here
- The **signal-handler table** defines the routine in the process's address space to be called when specific signals arrive
- The **virtual-memory context** of a process describes the full contents of the its private address space
- **Processes and Threads**
- Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent
- A distinction is only made when a new thread is created by the **clone** system call
- **fork** creates a new process with its own entirely new process context
- **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent
- Using **clone** gives an application fine-grained control over exactly what is shared between two threads

8.5 SCHEDULING

- The job of allocating CPU time to different tasks within an operating system
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks
- Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver

Relationship Between Priorities and Time-slice Length

List of Tasks Indexed by Priority



Kernel Synchronization

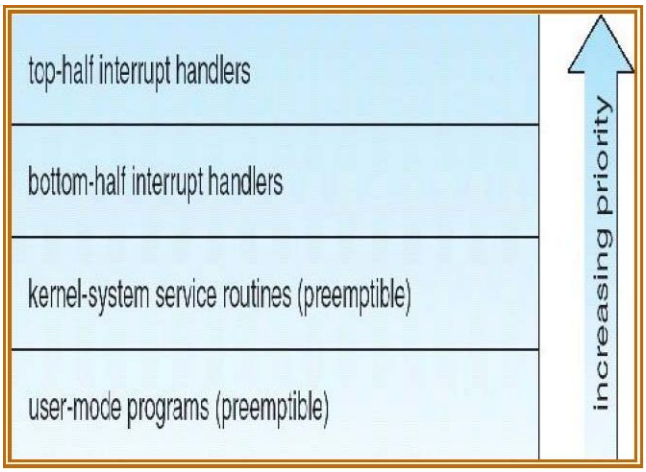
- A request for kernel-mode execution can occur in two ways:
  - A running program may request an operating system service, either explicitly via a system call, or implicitly, for example, when a page fault occurs
  - A device driver may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt
- Kernel synchronization requires a framework that will allow the kernel's critical sections to run without interruption by another critical section
- Linux uses two techniques to protect critical sections:
- ➓ 1. Normal kernel code is nonpreemptible (until 2.4)
- – when a time interrupt is received while a process is executing a kernel system service routine, the kernel's **need_resched** flag is set so that the scheduler will run once the system call has completed and control is about to be returned to user mode
- The second technique applies to critical sections that occur in an interrupt service routines

- By using the processor's interrupt control hardware to disable interrupts during a critical section, the kernel guarantees that it can proceed without the risk of concurrent access of shared data structures

.

- To avoid performance penalties, Linux's kernel uses a synchronization architecture that allows long critical sections to run without having interrupts disabled for the critical section's entire duration
- Interrupt service routines are separated into a *top half* and a *bottom half.*
  - The top half is a normal interrupt service routine, and runs with recursive interrupts disabled
  - The bottom half is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves
  - This architecture is completed by a mechanism for disabling selected bottom halves while executing normal, foreground kernel code

- **Interrupt Protection Levels**



- Each level may be interrupted by code running at a higher level, but will never be interrupted by code running at the same or a lower level

- User processes can always be preempted by another process when a time-sharing scheduling interrupt occurs

PROCESS SCHEDULING

- Linux uses two process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling between multiple processes
  - A real-time algorithm for tasks where absolute priorities are more important than fairness
- A process's scheduling class defines which algorithm to apply
- For time-sharing processes, Linux uses a prioritized, credit based algorithm
- ❿ The crediting rule

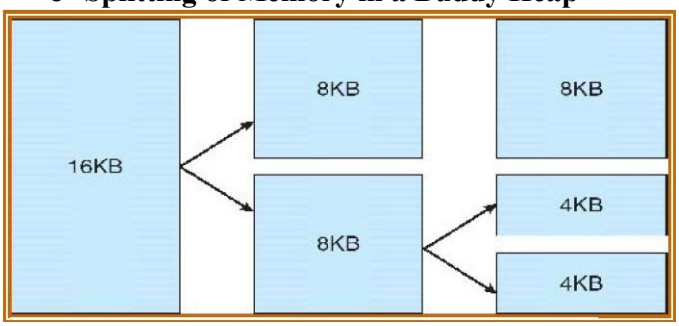
- factors in both the process's history and its priority

- This crediting system automatically prioritizes interactive or I/Obound processes

- **Symmetric Multiprocessing**
- Linux 2.0 was the first Linux kernel to support SMP hardware; separate processes or threads can execute in parallel on separate processors
- To preserve the kernel's nonpreemptible synchronization requirements, SMP imposes the restriction, via a single kernel spinlock, that only one processor at a time may execute kernel-mode code

8.6 **MEMORY MANAGEMENT**

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes
- Splits memory into 3 different **zones** due to hardware characteristics
- **Splitting of Memory in a Buddy Heap**



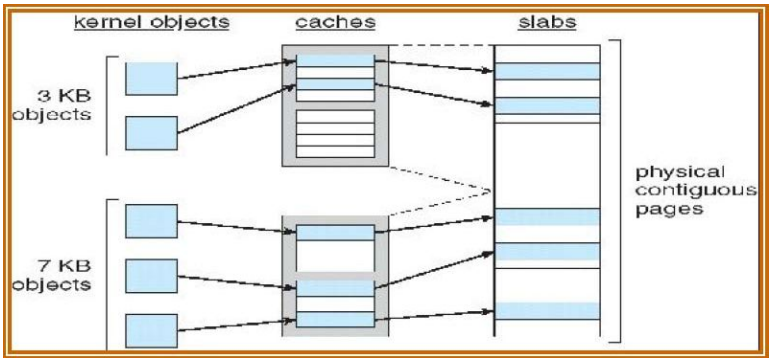
- o **Managing Physical Memory**
- o The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request
- o The allocator uses a buddy-heap algorithm to keep track of available physical pages
  - Each allocatable memory region is paired with an adjacent partner
  - Whenever two allocated partner regions are both freed up they are combined to form a larger region
  - If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request

- o Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator)
- o Also uses **slab allocator** for kernel memory

VIRTUAL MEMORY

- o The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required
- o The VM manager maintains two separate views of a process's address space:
- ❿ A logical view describing instructions concerning the layout of the address space
- x The address space consists of a set of nonoverlapping regions, each representing a continuous, page-aligned subset of the address space
  - A physical view of each address space which is stored in the hardware page tables for the process
- o Virtual memory regions are characterized by:
  - The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (*demand-zero* memory)
  - The region's reaction to writes (page sharing or copy-on-write)
- o The kernel creates a new virtual address space
  - o When a process runs a new program with the **exec** system call
  - o Upon creation of a new process by the **fork** system call

On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions



o Creating a new process with **fork** involves creating a complete copy of the existing process's virtual address space

- The kernel copies the parent process's VMA descriptors, then creates a new set of page tables for the child
- The parent's page tables are copied directly into the child's, with the reference count of each page covered being incremented
- After the fork, the parent and child share the same physical pages of memory in their address spaces

- o The VM paging system relocates pages of memory from physical memory out

to disk when the memory is needed for something else. .o The VM paging system can be divided into two sections:

- The pageout-policy algorithm decides which pages to write out to disk, and when
- The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed

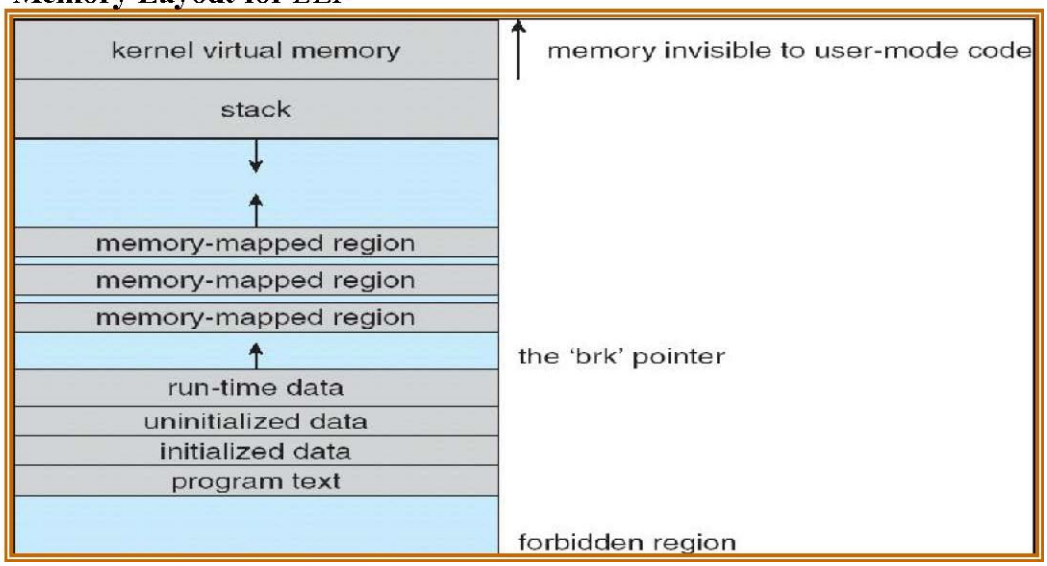
- o The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use
- o This kernel virtual-memory area contains two regions:
  - A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code
  - The reminder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory

- o **Executing and Loading User Programs**
- o Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made
- o The registration of multiple loader routines allows Linux to support both the ELF and **a.out** binary formats
- o Initially, binary-file pages are mapped into virtual memory

  - Only when a program tries to access a given page will a page fault result in that page being loaded into physical memory

- An ELF-format binary file consists of a header followed by several page-aligned sections
  - The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory

- **Memory Layout for** ELF



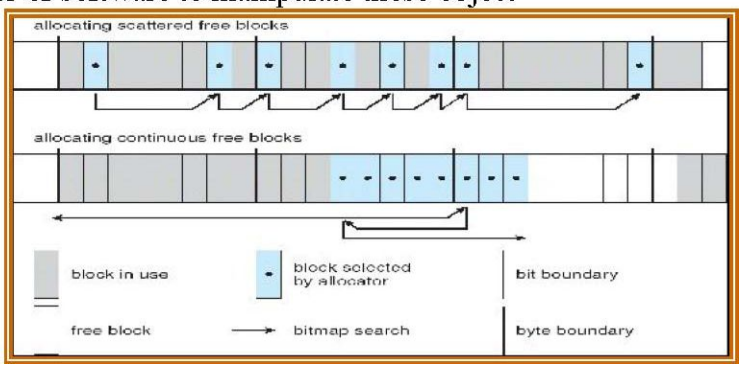
- **Static and Dynamic Linking**
- A program whose necessary library functions are embedded directly in the program's executable binary file is *statically* linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions
- *Dynamic* linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once

8.7 FILE SYSTEMS

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the *virtual file system (VFS)*
- The Linux VFS is designed around object-oriented principles and is composed of two components:

  - A set of definitions that define what a file object is allowed to look like x The *inode-object* and the *file-object* structures represent individual files
    x the *file system object* represents an entire file system

▪A layer of software to manipulate those object



8.INPUT AND OUTPUT

- The Linux device-oriented file system accesses disk storage through two caches:
  - Data is cached in the page cache, which is unified with the virtual memory system
  - Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block
- Linux splits all devices into three classes:
  - *block devices* allow random access to completely independent, fixed size blocks of data
  - *character devices* include most other devices; they don't need to support the functionality of regular files
  - *network devices* are interfaced via the kernel's networking subsystem

- **Block Devices**
- Provide the main interface to all disk devices in a system
- The *block buffer* cache serves two main purposes:
  - it acts as a pool of buffers for active I/O
  - it serves as a cache for completed I/O

- The *request manager* manages the reading and writing of buffer contents to and from a block device driver
- **Character Devices**
- A device driver which does not offer random access to fixed blocks of data
- A character device driver must register a set of functions which implement the driver's various file I/O operations
- The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device
- The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface

8.9INTERPROCESS COMMUNICATION

- Like UNIX, Linux informs processes that an event has occurred via signals
- There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process.
- The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait.queue** structures

- o **Passing Data Between Processes**
- o The pipe mechanism allows a child process to inherit a communication channel
- to its parent, data written to one end of the pipe can be read a the other
- o Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
- o To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism
- o **Shared Memory Object**
- o The shared-memory object acts as a backing store for shared-memory regions in the same way as a file can act as backing store for a memory-mapped memory region
- o Shared-memory mappings direct page faults to map in pages from a persistent shared-memory object
- o Shared-memory objects remember their contents even if no processes are currently mapping them into virtual memory
- o **Network Structure**
- o Networking is a key area of functionality for Linux.
- o It supports the standard Internet protocols for UNIX to UNIX communications
- o It also implements protocols native to nonUNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- o Internally, networking in the Linux kernel is implemented by three layers of software:
- o The socket interface
- o Protocol drivers
- o Network device drivers
- o **Security**
- o The *pluggable authentication modules (PAM)* system is available under Linux
- o PAM is based on a shared library that can be used by any system component that needs to authenticate users
- o Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (**uid** and **gid**)
- o Access control is performed by assigning objects a *protections mask*, which specifies which access modes—read, write, or execute—are to be granted to processes with owner, group, or world access

IMPORTANT QUESTIONS:

1. Describe three different ways that threads could be implemented. Explain how these ways compare to the Linux **clone** mechanism
2. What are the extra costs incurred by the creation and scheduling of a process, as compared to the cost of a cloned thread?
3. The Linux scheduler implements *soft* real-time scheduling. What features are missing thatare necessary for some real-time programming tasks?
4. How might they be added to thekernel?.What effect does thisrestriction have on the kernel's design?
5. What are two advantages and two disadvantages of this design decision?
6. What is the advantage of keeping this functionality out of the kernel? Are there any drawbacks?
7. What are three advantages of dynamic (shared) linkage of libraries compared to static linkage? What are two cases where static linkage is preferable.