UNIT I OPERATING SYSTEMS OVERVIEW	
Computer System Overview-Basic Elements, Instruction Execution, Interrupts, Memory Hierarchy, Cache Memory, Direct Memory Access, Multiprocessor and Multicore Organization. Operating system overview-objectives and functions, Evolution of Operating System.- Computer System Organization-Operating System Structure and Operations- System Calls, System Programs, OS Generation and System Boot.	
PART-A	
1	**Define Operating System?** An Operating System is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.
2	**What are the three main purposes of an operating system? (May/June 2013)** To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner. To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible. As a control program it serves two major functions: supervision of the execution of user programs to prevent errors and improper use of the computer Management of the operation and control of I/O devices.
3	**What are the four components of a computer system?** The hardware, Operating system, System and application programs example: Compiler, Assembler, Text editor, Database system, users.
4	**What is the purpose of interrupts? How does an interrupt differ from a trap? (Nov 2016)**

	An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.
5	**What is the need for DMA?** DMA, or Direct Memory Access, is a sub controller that can access memory in sequential order without intervention from the processor. It is used to avoid programmed I/O for large data movement , It bypasses CPU to transfer data directly between I/O device and memory, It can access the data items in primary and secondary cache
6	**Define mainframe Systems?** Main frame operating systems are designed primarily to optimize utilization of hardware. They were the first computers used to tackle many commercial and scientific applications. They can be namely as, Batch Systems, Multi programmed Systems and time-sharing systems.
7	**Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?** Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systems where more than one process may be accessing a datum. A component may be

	eliminated by an equal-sized cache, but only if: (a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.
8	**Define multiprocessor system and give down the advantages of Multiprocessor Systems?** Multiprocessor systems (also known as parallel systems or multi core systems or tightly coupled systems) have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral. Advantages are: Increased throughput, Economy of scale and increased reliability.
9	**Define Symmetric and asymmetric multiprocessing?** Symmetric: That all processors are peers; no master-slave relationship exists between processors. Each processor concurrently runs a copy of the operating system. Asymmetric: In which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed
10	**Define Client- server systems with its types?** User-interface functionality that used to be handled directly by the centralized systems is increasingly being handled by the PCs. As a result, centralized systems today act as server systems to satisfy requests generated by client systems. The two types of client – server systems are, Compute-server, File-server systems.
11	**Define Clustered systems with its types?** Another type of multiprocessor system is a clustered system, which gathers together multiple CPUs to accomplish computational work. In asymmetric clustering, one machine is in hot-stand by mode while the other is running the applications. The hot standby host does nothing but

	monitor the active server. In symmetric mode, two or more hosts are running applications, and they are monitoring each other.
12	**Some CPUs provide for more than two modes of operation. What are the possible uses of these multiple modes?** CPUs that support virtualization frequently have a separate mode to indicate when the virtual machine manager (VMM)—and the virtualization management software—are in control of the system.
13	**How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) system?** User mode and monitor mode (supervisor, system, privileged) the dual mode of operation provides the means for protecting the operating system from errant users—and errant users from one another. This protection is accomplished by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.
14	**What is a trap?** A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating- system service be performed.
15	**What are the activities performed by the process management?** Creating and deleting both user and system processes, Suspending and resuming process, Providing mechanisms for process synchronization, Providing mechanisms for process communication, Providing mechanisms for deadlock handling
16	**What are the activities performed by the main-memory management?** Keeping track of which parts of memory are currently being used and by whom, Deciding which processes are to be loaded into memory when memory space becomes available., Allocating and deal locating memory

	space as needed.
17	**What is cache coherency?** In a multiprocessor environment addition internal registers are maintained, each of the CPUs also contains a local cache. In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute in parallel, we must make sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called cache coherency, and it is usually a hardware issue.
18	**What are the services given by the operating system?** Program execution, I/o operations, File-system manipulation, Communications, Error detection
19	**Define system call? What is the purpose of system calls? (April 2018)** It provides the interface between a process and the operating system. Its categories are process control, file management, device management, information maintenance, and communications. System calls allow user-level processes to request services of the operating system.
20	**Why do some systems store the operating system in firmware, while others store it on disk?** For certain devices, for handheld PDA's and cellular telephones, a disk with a file system may be not being available for the device. In this situation the OS must be stored in firmware.
21	**Why is the separation of mechanism and policy desirable?** Mechanism and policy must be separate to ensure that systems are easy to modify. No two system installations are the same, so each installation may want to tune the operating system to suit its needs. With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged.
22	**What is bitmap?** A bitmap is a string of n binary digits that can be used to represent the status of n items. For example, suppose we have several resources, and the

	availability of each resource is indicated by the value of a binary digit: 0 means that the resource is available, while 1 indicates that it is unavailable (or vice-versa).
23	**What is system boot?** The procedure of starting a computer by loading the kernel is known as booting the system. On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution. Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.
24	**Define control statements.** When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically
25	**Do timesharing differs from Multiprogramming? If so, How? (April 2015)** Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. Timesharing (or multitasking) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.
26	**Why API's need to be used rather than system calls? (April 2015)** System calls differ from platform to platform. By using a stable API, it is easier to migrate your software to different platforms. The API usually provides more useful functionality than the system call directly. If you make the system call directly, you'll typically have to replicate the pre-call and post-call code that's already implemented by the API. The API can support multiple versions of the operating system and detect which version it needs to use at run time. If you call the

	system directly, you either need to replicate this code or you can only support limited versions.	
27	**Compare and contrast DMA and Cache Memory (Nov 2015)**	
	Direct Memory Access	**Cache Memory**
	It's a module in computer system which helps the devices in system to read/write in the Main memory without the intervention of CPU	It's a small storage present within CPU or between CPU and Main memory and stores most frequently used copy of data
	Advantage: It fastens bulk transfer of data from devices to main memory, as there is no need of CPU for every transfer	**Advantage:** It fastens the memory reference of required data, therefore fetching time reduced and execution is faster
	Disadvantage: Slows down processor due to Cycle stealing	**Disadvantage:** Maintaining Cache coherence and updating in copies of main memory and secondary storage is an overhead
28	**Write the difference between Batch Systems and Time Sharing Systems.(Nov 2015)**	
	Batch Systems	**Time Sharing Systems**
	Batch systems focuses to improve the CPU Utilization	Time Sharing Systems focuses to improve multi user interactive environment
	Batches the jobs (programs) together and executes job in sequence. Thus reduces idle time of CPU	Handle multiple-interactive jobs, by sharing processor time among multiple users. Multiple users access via terminals
	Eg IBM SYS, IBM OS for 7090/ 7094	Eg: Compatible time sharing systems (CTSS)
29	**What are the advantages of peer to peer systems over client server systems? (April 2016)**	

	It is easy to install and so is the configuration of computers on this network All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources. P2P is more reliable as central dependency is eliminated. Failure of one peer doesn't affect the functioning of other peers. In case of Client –Server network, if server own. There is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources. The over-all cost of building, maintaining this type of network is comparatively very less Security : Rules defining security and access rights can be defined at the time of set-up of server.
30	**What is the purpose of system programs? (April 2016)** System programs can be thought of as bundles of useful system calls. They provide basic functionality to users so that users do not need to write their own programs to solve common problems.
31	**What are the disadvantages of multiprocessor system (Nov 2016)** Overhead—The time wasted in achieving the required communications and control status prior to actually beginning the client's processing request Latency—The time delay between initiating a control command, or sending the command message, and when the processors receive it and begin initiating the appropriate actions Determinism—The degree to which the processing events are precisely executed Skew—A measurement of how far apart events occur in different processors, when they should occur simultaneously
32	**Consider a memory system with a cache access time of 10 ns and a memory access time of 110 ns – assume the memory access time includes**

	the time to check the cache. If the effective access time is 10% greater than the cache access time, what is the hit ratio H ? effective access time =cache hit ratio*cache access time+ cache miss ratio *(cache access time +main memory access time) effective access time =10% greater the cache access time ==>11 let cache hit ratio be h. 11 = h*10ns +(1-h) (110) 11= 10h+110-110h 100h= 99 ie, h= 99/100= 99%`
33	**What are the objectives of operating systems?** To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner. To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible. As a control program it serves two major functions:(1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) Management of the operation and control of I/O devices.
34	**What is difference between trap and interrupt?(Apr/May 2018)** An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.
35	**Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems?(Nov/Dec '18)** An operating system for a machine of this type would need to remain in control (or monitor mode) at all times. This could be accomplished by two methods:a. Software interpretation of all user programs (like some BASIC, Java, and LISP systems, for example). The software interpreter would provide, in software, what the hardware does not provide.b. Require meant that all programs be written in high-level languages so that

	all object code is compiler-produced. The compiler would generate (either in-line or by function calls) the protection checks that the hardware is missing.
36	***Can traps be generated intentionally by a user program? If so, for what purpose.***(Nov/Dec '18) A trap can be generated intentionally by a user program. It can be used to call operating system routines or to catch arithmetic errors.
37	**under what circumstances would a user be better off using a timesharing system rather than a PC or single -user workstation.(Nov 2018)** Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.A workstation is a special computer designed for technical or scientific applications. Intended primarily to be used by one person at a time, they are commonly connected to a local area network and run multi-user operating systems.

PART-B

1	***Discuss about various system components in detail (or) Explain about various managements of operating system and responsibilities*** *An operating system (OS) is a program that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. Following are the various components of OS* ***1 Process Management()*** *Program is a set of instruction that solves a purpose/task. A **process** is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.* *CPU executes one instruction of process after another until process completes* *Resources are given to process when its created or allocated while it is executing.* *Process termination requires reclaim of any reusable resources.* *System consists of collection of processes, some of which are operating system processes(that execute system code) and rest are user processes(that execute user code)* *All these processes can be executed concurrently by multiplexing CPU among them.* *Role of OS in process management:* *Creating and deleting both user and system processes*

Suspending and resuming processes
Providing mechanisms for process synchronization
Providing mechanisms for process communication
Providing mechanisms for deadlock handling

(2) Main-Memory Management

Main-Memory -Large array of words or bytes.
It's a repository of quickly accessible data shared by I/O devices.
All instructions must be in main memory in order to execute.
For the CPU to process data from disk, data must first transfer to main memory by CPU-generated I/O calls.
For a program to be executed, it must be mapped to absolute address & loaded into memory.
When program terminates, memory space is declared available & next program can be loaded and executed.
Efficient memory management improves CPU utilization and computer response to users.

Role of OS in Memory management
Keeping track of which parts of memory are currently being used and by whom
Deciding which processes (or parts thereof) and data to move into and out of memory.
Allocating and deallocating memory space as needed.

(3)File Management

Magnetic tape, disk and optical disk – used to store information
Each storage media has own characteristics and physical organization.
OS provides uniform, logical view of information storage
Abstracts physical properties to logical storage unit - file
Each medium is controlled by device (i.e., disk drive, tape drive)
OS maps files in to physical media & accesses these files via the storage devices
File à collection of related information defined by its creator
represents programs and data
data files , text files

File-System management

Files usually organized into directories

Access control on most systems to determine who can access what

Role of OS in File Management

Creating and deleting files and directories

Primitives to manipulate files and directories

Mapping files onto secondary storage

Backup files onto stable (non-volatile) storage media

(4) I/O System Management

One purpose of OS is to hide internal working (peculiarities) of hardware devices from the user

In UNIX, such peculiarities of I/O devices are hidden from the bulk of OS itself by the IO subsystem

I/O subsystem responsible for

Memory management of I/O including **buffering** (storing data temporarily while it is being transferred), **caching** (storing parts of data in faster storage for performance), **spooling** (the overlapping of output of one job with input of other jobs)

General device-driver interface

Drivers for specific hardware devices

Only device driver knows the peculiarities of the specific device to which it is assigned

(5) Secondary-Storage Management

Usually disks used to store data that does not fit in main memory / **primary storage** or data that must be kept for a "long" period of time.

Secondary storage –To back up main memory

Role of OS in disk management

Free-space management

Storage allocation

Disk scheduling

(6) Protection System

Protection is any mechanism for controlling access of processes or users to resources defined by the OS

Security – defense of the system against internal and external attacks

Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

Systems generally first distinguish among users, to determine who can do what

User identities (user IDs, security IDs) include name and associated number, one per user

User ID then associated with all files, processes of that user to determine access control

Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file

Privilege escalation allows user to change to effective ID with more rights

(7) Command-Interpreter System

Command line interpreter is an important system program that interfaces between user and OS

Resides In kernel of some OS

But in MS-DOS and UNIX – Command Interpreter is a special program that is running when a job is initiated or when a user first logs in.

Control Statements- commands given to OS

Control card interpreter or command-line interpreter or shell àprogram that reads and interprets control statements, gets next command statement and execute it

User friendly interface is mouse-based window and menu system, icons eg. MS Windows and Macintosh

MS-DOS and UNIX à shells operate as command-line interpreter

2	**Discuss the following a) Operating System service b) Cache Memory c) Direct Memory (Nov 2015)**

(a) Operating-System Services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs.

These operating-system services are provided for the convenience of the programmer, to make the programming task easier.

(i) User interface

User Interface is the space where interactions between humans and machines occur. Almost all operating systems have a user interface (UI).This interface can take several forms.

One is a command-line interface(CLI), which uses text commands and a method for entering them (say, a program to allow entering and editing of commands).

Another is a batch interface, in which commands and directives to control those commands are entered into files, and those files are executed.

Most commonly/ a graphical user interface (GUI) is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.

(ii) Program execution

The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

(ii) I/O operations

A running program may require I/O, which may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

(iii) File-system manipulation

Programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Some programs include permissions management to allow or deny access to files or directories based on file ownership.

(iv)Communications

There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.

Communications may be implemented via shared memory or through message passing, in which packets of information are moved between processes by the operating system.

(v) Error Detection

The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (lack of paper in the printer), and in the user program (too-great use of CPU time).

For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Debugging facilities can greatly enhance the user's and programmer's abilities to use the system efficiently.

(vi) Resource Allocation

When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system.

Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. Various CPU scheduling routines are used to manage the resource allocation efficiently.

(vii) Accounting

We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

	(viii) Protection and security

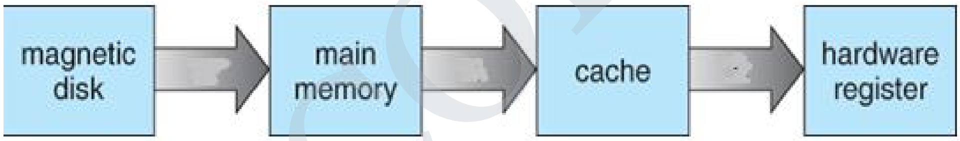
Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important.

Security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources. It extends to defending external I/O devices, including modems and network adapters, from invalid access attempts and to recording all such connections for detection of break-ins.

(b) Cache Memory

A cache is a **small high-speed storage space** or **temporary** storage that stores **most frequently needed** portions of larger slower storage areas

Access to the cached copy is more efficient than access to the main memory. The Cache Memory is the nearest memory to the CPU, all the recent instructions are Stored into the Cache Memory. The instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches.



When CPU requests contents of memory location

Check cache for the data

If data found in cache the its called as "**Cache Hit**", the data is transferred to CPU for processing

If data is not found in cache it is called as "**Cache Miss**", the required data is then searched in main memory or secondary memory and transferred to the cache later.

Multi-level Caches: multi level caches are used to overcome the disadvantages of longer latency in single larger caches. Multi-level caches generally operate by checking the fastest, level 1 (L1) cache first; if it hits, the processor proceeds at high speed. If that smaller cache misses, the next fastest cache (level 2, L2) is checked, and so on, before external memory is checked.

Cache Coherence: In a shared memory multiprocessor with a separate cache memory for each processor , it is possible to have many copies of any one instruction operand. Cache coherence ensures that changes in the values of shared operands are updated throughout the system.

Cache vs Buffer: The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes.

(c) Direct Memory Access

Programmed I/O (PIO) : expensive general-purpose processor used to watch status bits and to feed data into a controller register one byte at a time . Many computers avoid burdening the main CPU with PIO by offloading some of this work to a special-purpose processor called a **direct-memory-access (DMA)** controller.

To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.

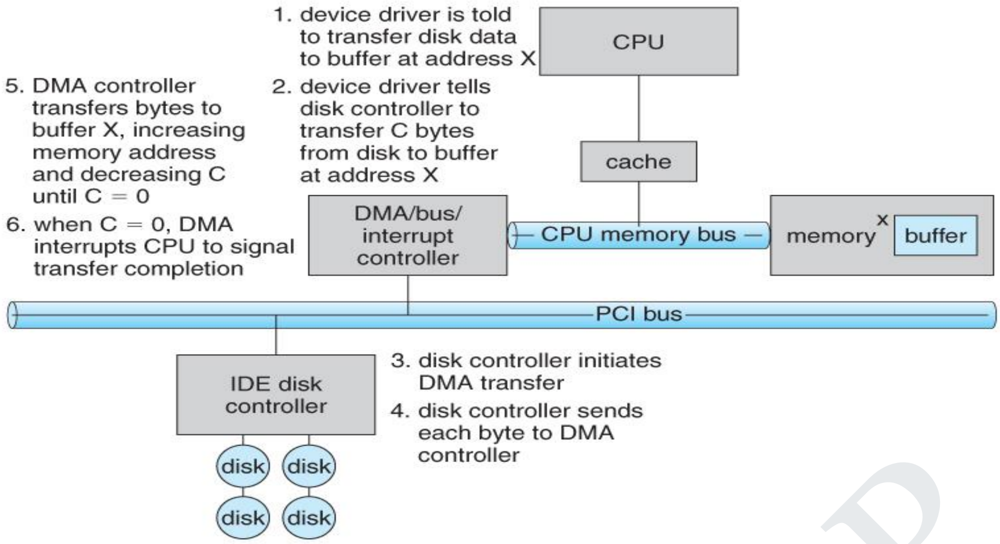
The CPU writes the address of this command block to the DMA controller, then goes on with other work.

The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU.

A simple DMA controller is a standard component in PCs, and **bus-mastering** I/O boards for the PC usually contain their own high-speed DMA hardware.

Handshaking between the DMA controller and the device controller is performed via a pair of wires **called DMA-request and DMA-acknowledge.**

The device controller places a signal on the DMA-request wire when a word of data is available for transfer. This signal causes the DMA controller to seize the memory bus, to place the desired, address on the memory-address wires, and to place a signal on the DMA-acknowledge wire.

	 DMA Transfer Diagram When the device controller receives the DMA-acknowledge signal, it transfers the word of data to memory and removes the DMA-request signal. When the entire transfer is finished, the DMA controller interrupts the CPU. This process is depicted in Figure above. While the DMA transfer is going on the CPU does not have access to the PCI bus (including main memory), but it does have access to its internal registers and primary and secondary caches Although this **cycle stealing** can slow down the CPU computation, DMA controller generally improves the total system performance. Some computer architectures use physical memory addresses for DMA, but others perform **direct virtual memory access (DVMA)**, using virtual addresses that undergo translation to physical addresses. DVMA can perform a transfer between two memory-mapped devices without the intervention of the CPU or the use of main memory.
3	***Give the details about hardware protection and dual mode protection.*** **Hardware Protection:** Multiprogramming put several programs in memory at the same time. With sharing, many processes could be adversely affected by a bug in one program. Without protection against these sorts of errors, either the computer must execute only one process at a time, or all output must be suspect. A properly designed operating system must ensure that an incorrect (or

malicious) program cannot cause other programs to execute incorrectly.
These errors are normally handled by the operating system. If a user program fails in someway—such as makes an attempt either to execute an illegal instruction, or to access memory that is not in the user's address space—then the **hardware will trap to the operating system**. The **trap** transfers control through the interrupt vector to the OS just like the interrupt.
Whenever a program error occurs, the operating system must **abnormally terminate the program**.
Thus via hardware, the effects of erroneous programs are prevented.

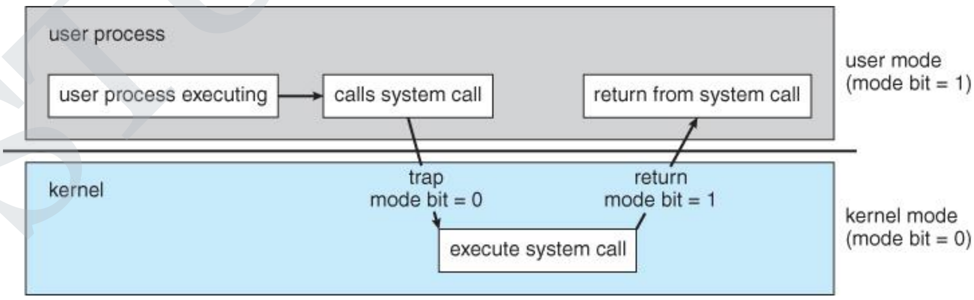
Dual Mode Operation:

To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program. we need two separate *modes* of operation:

user mode

kernel mode(supervisor mode/ system mode/ privileged mode)

A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: **kernel (0) or user (1)**.
With the mode bit, we are able to distinguish between an execution that is done on behalf of the operating system, and one that is done on behalf of the user.



Transition from user to kernel mode

At **system boot** time, the hardware starts in **kernel mode.** The operating system is then loaded, and starts **user processes** in **user mode**. Whenever **a trap or interrupt occurs**, the hardware switches from user mode to **kernel mode**(that is, changes the state of the mode bit to 0).
Thus, whenever the operating system gains control of the computer, it is in

	kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program. The dual mode of operation provides us with the means for protecting the operating system from errant users, and errant users from one another. This protection by designating some of the machine instructions that may cause harm as **privileged instructions.** The hardware allows privileged instructions to be executed in only kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction, but rather treats the instruction as illegal and traps to the operating system. The lack of a hardware-supported dual mode can cause serious shortcomings in an operating system **Example:** More recent and advanced versions of the Intel CPU, such as the **80486**, provide dual-mode operation. As a result, more recent operating systems, such as **Microsoft Windows NT**, and **IBM OS/2**,take advantage of this feature and provide greater protection for the operating system
4	**List and explain the classification of system calls and its significance(NOV/DEC 18)** **System calls** provide the **interface** between a **running program** and the **operating system**. Generally available as assembly-language instructions. Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++) **Five Categories of system calls** Process control File management Device management Information maintenance Communications **Process control** end, abort load, execute create process, terminate process get process attributes, set process attributes

wait for time

wait event, signal event

allocate and free memory

Significance: Using these system calls process can be loaded, aborted and allocated memory. Attributes and important details regarding a process can be obtained.

File management

create file, delete file

open, close

read, write, reposition

get file attributes, set file attributes

Significance: File management system calls are used to perform all basic operations concerning a file like file open, close, read, write and to obtain file attributes and details.

Device management

request device, release device

read, write, reposition

get device attributes, set device attributes

logically attach or detach devices

Significance: A program, as it is running, may need additional resources/devices like memory, tape drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user program; otherwise, the program will have to wait until sufficient resources are available. Such device request, connections and release can be instrumented using these system calls.

Information maintenance

get time or date, set time or date

get system data, set system data

get process, file, or device attributes

set process, file, or device attributes

Significance: Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date, number of current users, the version number of the operating system,

	the amount of free memory or disk space, and so on. **Communications** create, delete communication connection send, receive messages transfer status information attach or detach remote devices Significance: Communication system calls are used to set up communication links, transfer messages and report transfer status. In the shared-memory model, processes use map memory system calls to gain access to regions of memory owned by other processes. In the message-passing model, information is exchanged through an interprocess-communication facility provided by the operating system.
5	**How could a system be designed to allow a choice of operating system from which to boot? What would the boot strap program need to do?** The procedure of starting a computer by loading the kernel is known as booting the system. On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel,loads it into main memory, and starts its execution. Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel. Consider a system that would like to run both Windows XP and three different distributions of Linux (e.g., RedHat, Debian, and Mandrake). Each operating system will be stored on disk. During system boot-up, a special program (which we will call the boot manager) will determine which operating system to boot into. This means that rather initially booting to an operating system, the boot manager will first run during system startup. It is this boot manager that is responsible for determining which system to boot into. Typically boot managers must be stored at certain locations of the hard disk to be recognized during system startup. So, rather than booting to an operating system, the boot manager will first run during system startup. The boot manager is responsible for determining which system to boot into. Often, boot managers provide the

user with a selection of systems to boot into. Example boot manager is Grub, LILO which come from Linux environment.

Boot managers often provide the user with a selection of systems to boot into; boot managers are also typically designed to boot into a default operating system if no choice is selected by the user.

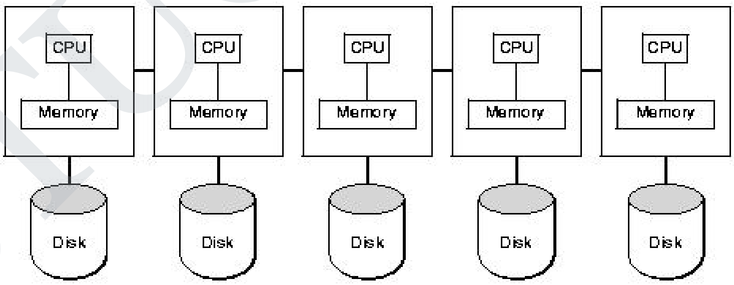
How do clustered systems differ from multi-processor systems? What is required for two machines belonging to cluster to cooperate to provide a highly available service?

Clustered systems are typically constructed by combining multiple computers into a single system to perform a computational task distributed across the cluster.

Multiprocessor systems on the other hand could be a single physical entity comprising of multiple CPUs.

A clustered system is less tightly coupled than a multiprocessor system. Clustered systems communicate using messages, while processors in a multiprocessor system could communicate using shared memory.

Clustering is usually used to provide high-availability service; that is,service will continue even if one or more systems in the cluster fail. High availability is generally obtained by adding a level of redundancy in the system. A layer of cluster software runs on the cluster nodes.



Clustered System

Each node can monitor one or more of the others (over the LAN). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. The users and clients of the applications see only a brief interruption of service.

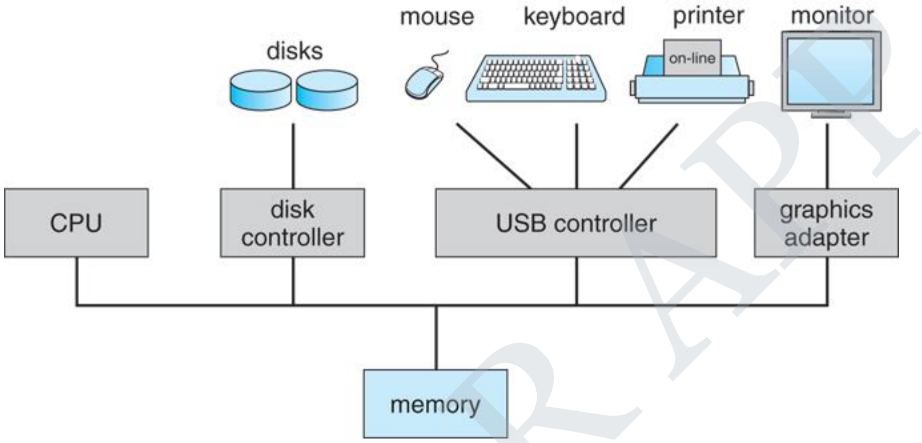
Therefore in order for two machines to provide a highly available service, the state on the two machines should be replicated and should be consistently updated.

When one of the machines fail, the other could then take-over the functionality of the failed machine.

6 **Explain in detail computer system organization.**

Computer-System Operation

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.



A Modern Computer System

Each device controller is in-charge of a specific type of device (for example, disk drives, audio devices, and video displays).

The CPU and the device controllers can execute concurrently, competing for memory cycles.

To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or **bootstrap program** is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term **firmware,** within the computer hardware.

It initializes all aspects of the system, from CPU registers to device controllers to memory contents.

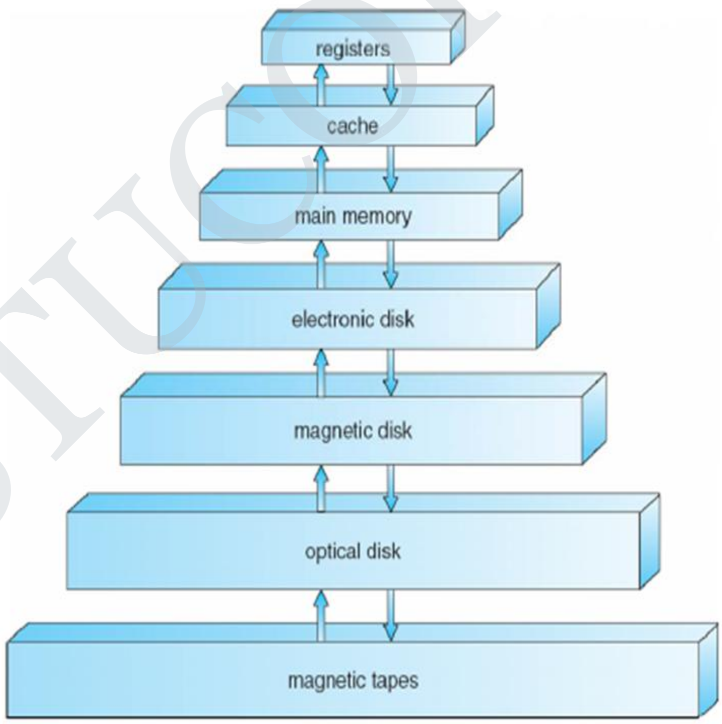
The bootstrap program must know how to load the operating system and to start executing that system. To accomplish this goal, the bootstrap program must locate and load into memory the operating system kernel. The operating system then starts executing the first process, such as "init," and waits for some event to occur.

Interrupts are an important part of a computer architecture.The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a **system call** (also called a **kernel call**).

Storage Structure

Computer programs must be in main memory (also called **random-access memory** or **RAM**) to be executed. Main memory is the only large storage area(millions to billions of bytes) that the processor can access directly.

RAM is commonly implemented in a semiconductor technology called **dynamic random-access memory (DRAM)**, which forms an array of memory words main memory.



Storage Memory Hierarchy

The main requirement for secondary storage is that it be able to hold large quantities of data permanently. The most common secondary-storage device is a **magnetic disk**, which provides storage for both programs and data. Most programs (web browsers, compilers, word processors, spreadsheets, and so on) are stored on a disk until they are loaded into memory.

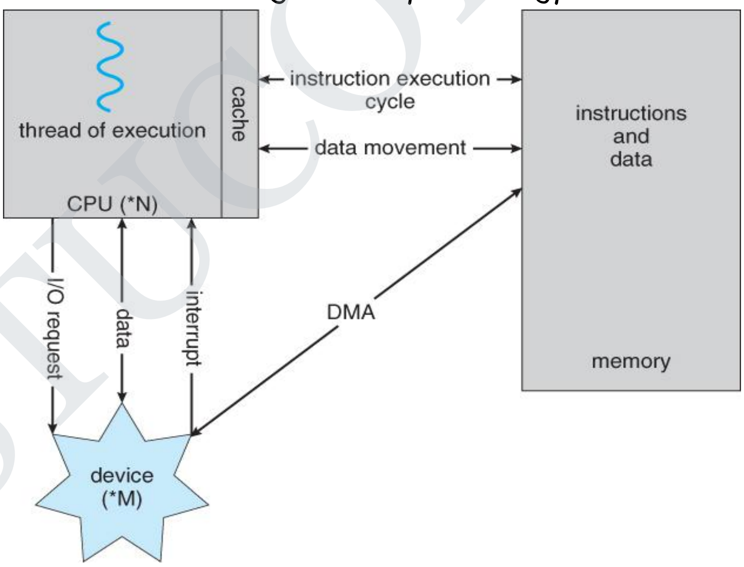
Volatile storage (RAM) loses its contents when the power to the device is removed. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage (Secondary Memory)** for safekeeping.

Cache memory is small temporary high speed storage between CPU and RAM that stores frequently used data to reduce the latency in data access.

Input/Output (I/O) Structure

A large portion of operating system code is dedicated to managing I/O, because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device.

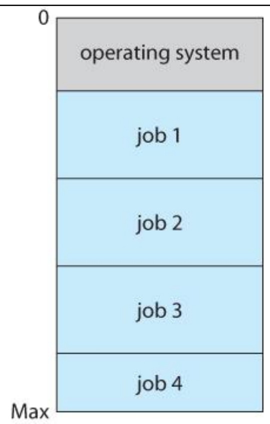


Working of Modern Computer System

Depending on the controller, there may be more than one attached device. For instance, seven or more devices can be attached to the **small computer-systems interface (SCSI)** controller.

A ***device controller*** maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and

	its local buffer storage. Operating systems have a **device driver** for each device controller. This device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system. This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem, **direct memory access (DMA)** is used. After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. While the device controller is performing these operations, the CPU is available to accomplish other work.
7	**State the operating system structure. Describe the operating system operations in detail. Justify the reasons why the lack of a hardware supported dual mode can cause serious shortcoming in an operating system?(April 2018)** An operating system provides the environment within which programs are executed. One of the most important aspects of operating systems is the ability to multiprogram. **Multiprogramming** increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. Eg: The operating system keeps several jobs in memory simultaneously. This set of jobs can be a subset of the jobs kept in the **job pool** that contains all jobs that enter the system. The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another job. As long as at least one job needs to execute, the CPU is never idle.



Memory Layout of Multiprogramming System

Time sharing (or **multitasking**) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

Time sharing requires an **interactive** (or **hands-on**) **computer system,** which provides direct communication between the user and the system.

The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the **response time** should be short—typically less than one second.

A time-shared operating system allows many users to share the computer simultaneously. Time-shared operating system uses CPU scheduling and multiprogramming

Time-sharing and multiprogramming require several jobs to be kept simultaneously in memory. Since in general main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the **job pool.**

This pool consists of all processes residing on disk awaiting allocation of main memory.

If several jobs are ready to be brought into memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is **job scheduling,** if several jobs are ready to run at the same time, the system must choose among them. Making this decision is **CPU scheduling,**

In a time-sharing system, the operating system must ensure reasonable response time, which is sometimes accomplished through **swapping,** where processes are swapped in and out of main memory to the disk.

A more common method for achieving this goal is **virtual memory,** a technique that allows the execution of a process that is not completely in memory

The main advantage of the virtual-memory scheme is that it enables users to run programs that are larger than actual **physical memory**, it abstracts main memory into a large, uniform array of storage, separating **logical memory** as viewed by the user from physical memory.

Operating-System Operations

Modern operating systems are **interrupt driven.** Events are almost always signaled by the occurrence of an interrupt or a trap. A **trap (or** an **exception)** is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed.

The interrupt-driven nature of an operating system defines that system's general structure. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt

A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

(i)Dual-Mode Operation

To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program. we need two separate *modes* of operation:

user mode

kernel mode(supervisor mode/ system mode/ privileged mode)

A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: **kernel (0) or user (1**).

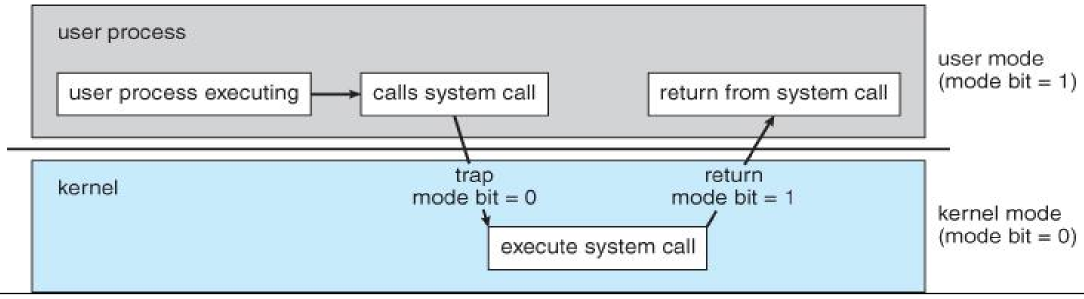
With the mode bit, we are able to distinguish between an execution that is done on behalf of the operating system, and one that is done on behalf of the user.

At **system boot** time, the hardware starts in **kernel mode.** The operating system is then loaded, and starts **user processes** in **user mode**. Whenever **a trap or interrupt occurs**, the hardware switches from user mode to **kernel mode**(that is, changes the state of the mode bit to 0).

Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users, and errant users from one another.

This protection by designating some of the machine instructions that may cause harm as **privileged instructions.**



8 **Discuss about the evolution of Virtual Machines. Also explain how virtualization could be implemented in operating systems. (April 2015)**

The Invention of the Virtual Machine

In the Early 1960's IBM had a wide range of systems; each generation of which was substantially different from the previous. This made it difficult for customers to keep up with the changes and requirements of each new system. Also, computers could only do one thing at a time. If you had two tasks to accomplish, you had to run the processes in batches. This Batch processing requirement wasn't too big of a deal to IBM since most of their users were in the Scientific Community and up until this time Batch processing seemed to have met the customers needs.

The main advantages of using virtual machines vs a time sharing operating system was more efficient use of the system since virtual machines were able to share the overall resources of the mainframe, instead of having the resources split equally between all users. There was better security since each users was running in a completely separate operating system. And it was more reliable since no one user could crash the entire system; only their own operating system.

In 1990, Sun Microsystems began a project known as "Stealth". Stealth was a project run by Engineers who had become frustrated with Sun's use of C/C++ API's and felt there was a better way to write and run applications

In 1994 Java was targeted towards the Worldwide web since Sun saw this as a major growth opportunity. The Internet is a large network of computers running on different operating systems and at the time had no way of running rich applications universally, Java was the answer to this problem. In January 1996. the Java Development Kit (JDK) was released, allowing developers to write applications for the Java Platform.

Mainstream Adoption of Hardware Virtualization

As was covered in the Invention of the Virtual Machine section, IBM was the first to bring the concept of Virtual Machines to the commercial environment. Virtual Machines as they were on IBM's Mainframes are still in use today, however most companies don't use mainframes. In January of 1987, Insignia Solutions demonstrated a software emulator called SoftPC. SoftPC allowed users to run Dos applications on their Unix workstations.

Discuss about the functionality of System boot with respect to operating system (April 2015)(NOV/DEC 2018)

When you turn on your computer, chances are that the operating system has been set up to boot (load into RAM) automatically in this sequence:
As soon as the computer is turned on, the basic input-output system (BIOS) on your system's read-only memory (ROM) chip is "woken up" and takes charge. BIOS is already loaded because it's built-in to the ROM chip and, unlike random access memory (RAM), ROM contents don't get erased when the computer is turned off.

BIOS first does a power-on self test (POST) to make sure all the computer's components are operational. Then the BIOS's boot program looks for the special boot programs that will actually load the operating system onto the hard disk.

First, it looks on drive A (unless you've set it up some other way or there is no diskette drive) at a specific place where operating system boot files are located. If there is a diskette in drive A but it's not a system disk, BIOS will send you a message that drive A doesn't contain a system disk. If there is no diskette in drive A (which is the most common case), BIOS looks for the system files at a specific place on your hard drive.

Having identified the drive where boot files are located, BIOS next looks at the first sector (a 512-byte area) and copies information from it into specific locations in RAM. This information is known as the *boot record* or Master Boot Record.

	It then loads the boot record into a specific place (hexadecimal address 7C00) in RAM. The boot record contains a program that BIOS now branches to, giving the boot record control of the computer. The boot record loads the initial system file (for example, for DOS systems, IO.SYS) into RAM from the diskette or hard disk. The initial file (for example, IO.SYS, which includes a program called SYSINIT) then loads the rest of the operating system into RAM. (At this point, the boot record is no longer needed and can be overlaid by other data.) The initial file (for example, SYSINIT) loads a system file (for example, MSDOS.SYS) that knows how to work with the BIOS. One of the first operating system files that is loaded is a system configuration file (for DOS, it's called CONFIG.SYS). Information in the configuration file tells the loading program which specific operating system files need to be loaded (for example, specific device driver. Another special file that is loaded is one that tells which specific applications or commands the user wants to have included or performed as part of the boot process. In DOS, this file is named AUTOEXEC.BAT. In Windows, it's called WIN.INI. After all operating system files have been loaded, the operating system is given control of the computer and performs requested initial commands and then waits for the first interactive user input.
9	**With neat sketch discuss Computer System Overview (Nov 2015)** **Computer System Overview** Computer is an electronic machine that accepts user input, process the data and outputs the desired result. **Basic Elements** The basic elements of computer like processor, memory and

input/output (I/O) components and modules are interconnected to achieve main function of computer. The four basic structural elements are:

Processor:

often referred as **Central Processing Unit**

Controls **operation** of computer

performs **data processing** functions

Main Memory

Also called as **real/primary memory**

Stores data and programs

volatile (contents of memory lost when system is shut down)

I/O Modules

Moves data between **computer** and **external environment**

External environment- devices like disks(secondary memory), terminals, communication equipments

System Bus

Provides **communication** among processors, main memory and I/O modules

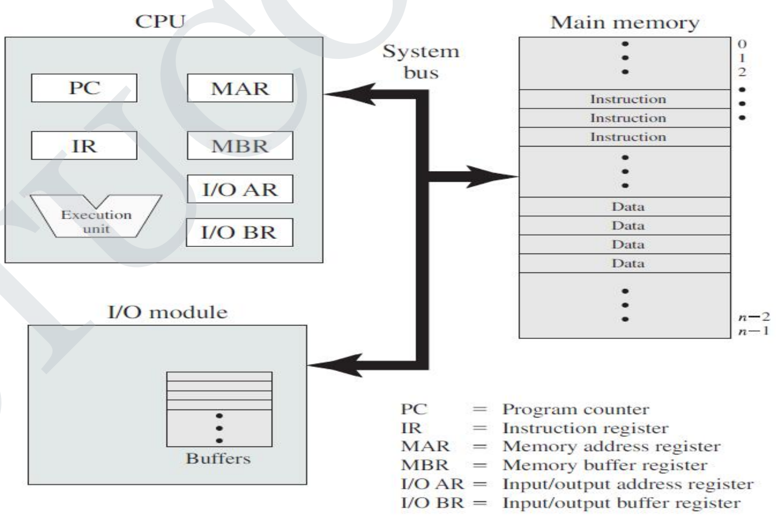


Figure 1.1 Computer Components-Top Level View

Figure 1.1 depicts the top level components of a computer. The processor exchanges data with the memory using two registers MAR(Memory Address Register-specifies memory address) and MBR (Memory Buffer Register-contains data). I/O AR specifies particular I/O device, I/O BR used to exchange data between processor and I/O device.

Instruction Execution

A **program** consists of **set of instructions** and is stored in memory, which is **executed** by the **processor** later.

Processing required for single instruction is called as **instruction cycle**

Processing of an instruction(instruction cycle) : **2 stages**

fetch stage

execute stage

Fetch stage:

At beginning of each instruction cycle processor fetches instruction **from memory**.

Program Counter(PC) holds **address of next instruction** to be fetched. PC incremented after every instruction fetch.

Execute stage:

Fetched instruction is loaded into **instruction register(IR)**.Instruction contains bits to specify action to be taken.

Interrupts

Interrupt is a **signal** emitted by **hardware or software** to the **processor** in-order alert the processor of higher priority event so that **current instruction** execution is **interrupted**.

Interrupts improve **processor utilization.**

Interrupts	Traps
Signal emitted by hardware to the processor	Signal Emitted by a software or a user program
Instances when h/w interrupts generated are : completion of a DMA , I/O request etc	It is caused by an **error** (for example, division by zero or invalid memory access)or by a **specific request** from a use program for an operating-system servic
Hardware interrupts are **asynchronous** since they are not associated with a specific instruction or process.	Traps(software interrupts) are **synchronous** since they are associate with a specific instruction and process.

Memory Hierarchy

Memory is a storage place where various data, instructions, program, documents are stored. Memory hierarchy can be determined by following factors:

Access time

Frequency of access by processor

Capacity

Cost

Factors\Memory	**Registers**	**Cache**	**Main Memory**	**Seco Mem**
Access time	very less	lesser	high	high
Cost	Costliest	Costlier	cheaper	chea
Capacity	very less	less	high	high
Frequency of access	Highest	higher	high	low

Register

The register is a **high speed storage memory** present **within CPU**, therefore the access time is very fast in case of registers

Cache

It's a **small fast storage** placed **between** the **CPU** and **Main memory**.

Frequently accessed data are stored in cache to improve access of data and reduce the access time

Main Memory

It has **more capacity** compared to cache.

There are two parts :**Random Access Memory**(RAM) and **Read Only Memory** (ROM).

RAM-**volatile** memory-data **temporarily** stored, programs are **loaded before** being executed in CPU

ROM-**non volatile**, stores important **system programs**

Secondary Memory

It's the largest and slowest memory

Made of **magnetic disks** and **tapes**

It's a **non-volatile** memory-data permanently stored

All **back up data** stored in secondary memory

Direct Memory Access

DMA Completely takes care of I/O operation **without processor intervention** to read/write data from memory

DMA function – may be present as **a module in s/m bus** or **inbuilt in I/O module**

Advantage: Efficient technique to move large volumes of data.

DMA uses **CPU-Memory bus** for data transfer between I/O and main memory

Processor cannot access bus during DMA transfer. **Processor waits** for bus, which is called as cycle stealing.

Cycle Stealing : DMA grabs the machine cycles from the processor to directly access the memory to complete I/O operations

Processor pauses for one bus cycle(not stopped)

Disadvantage: Processor **gets slower** on DMA transfers

Multiprocessor

Single entity or system having **multiple CPU/processors**

Communicate with each other using **shared memory**

Concept: **Parallelism by replicating processors**

Motivation: **Fast processing** and high performance

Advantages of SMP over Uniprocessor

Performance

Availability

Incremental growth

Scaling.

Multicore Organization

Consists of **2 or more processors (cores) on a single piece** of silicon(die) <u>chip</u>

Also called as **chip multiprocessor**

Each core consists of registers, ALU,CU,Caches etc

Motivation:

Parallelism provide more performance

Increase miniaturization to improve processing speed

Example: Intel Core i7

Enumerate different operating system structures and explain with neat sketch (Nov 2015) (April 2017)

OPERATING-SYSTEM STRUCTURE

<u>1.Simple Structure</u>

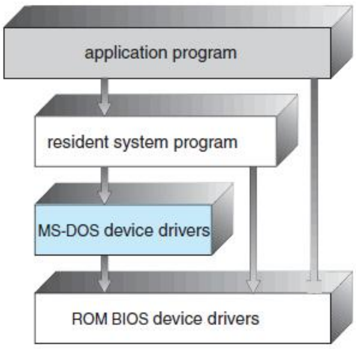


Figure 2.11 MS-DOS layer structure.

Operating systems were started as small, simple, and limited systems.

The interfaces and levels of functionality are not well separated.Application programs are able to access the basic I/O routines

Such freedom leaves systems (eg-MS-DOS) vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail.

Advantage: simple design and Easy construction

Disadvantage: difficult to debug, prone to frequent system failure

Example MS-DOS , UNIX operating system.

2. Layered Approach

With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate

The operating system can then retain much greater control over the computer and over the applications that make use of that computer.

Operating system is broken into a number of layers(levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface. The mid layers mostly comprising the OS system and application programs This layering structure is depicted in Figure 2.13.

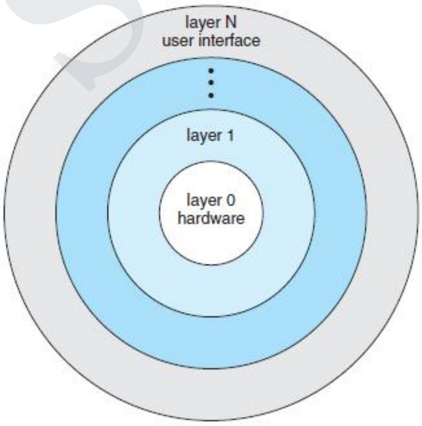


Figure 2.13 A layered operating system.

Advantage

Simplicity of construction and debugging

The first layer can be debugged without any concern for the rest of the system

A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware-level layers.

Disadvantage

Careful planning on design is necessary

less efficient than other types. Each layer adds overhead to the system call. The net result is a system call that takes longer than does oneon a nonlayered system.

Example VAX/VMS, Multics

3. Microkernels

Researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach.

Only essential Operating system functions like thread management, address space management, inter-process communication are built in kernel

All nonessential components from the kernel are removed and implemented as system and user-level programs.

The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.

Communication is provided through **message passing,**

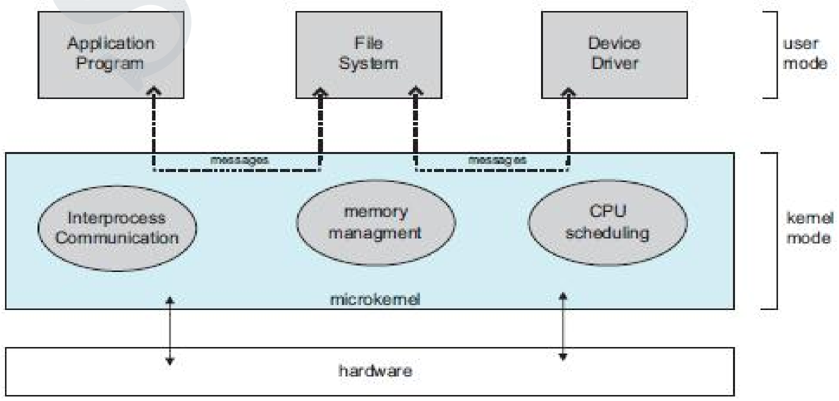
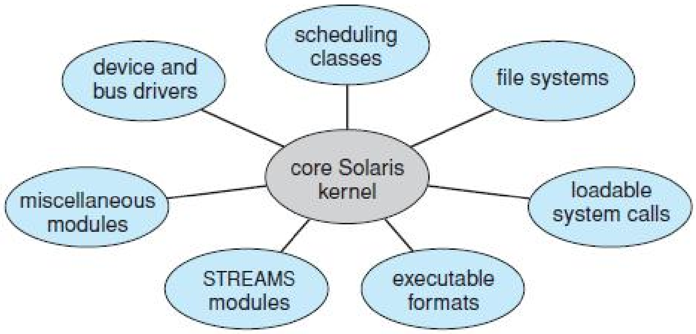
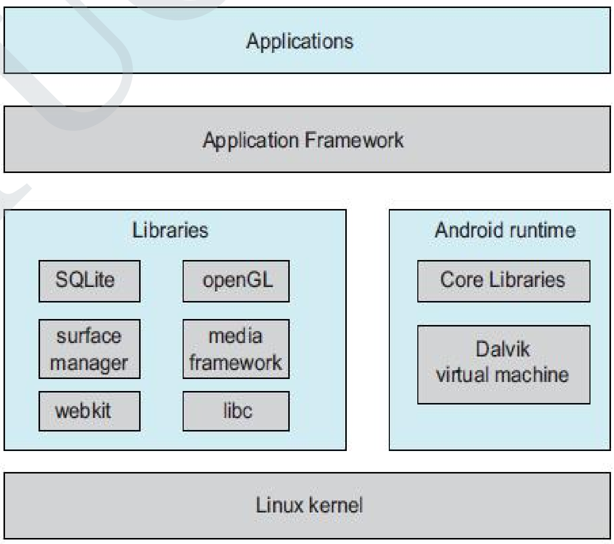


Figure 2.14 Architecture of a typical microkernel.

Advantage

	It makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel. Its easier to port from one hardware design to another. Provides more security and reliability, since most services are running as user process—rather than kernel process If a service fails, the rest of the operating system remains untouched **Disadvantage** The performance of microkernels can suffer due to increased system-function overhead. **Example:** Mac OS X kernel (also known as **Darwin**) based onMach microkernel **<u>4. Modules Approach</u>** The best current methodology for operating-system design involves using **loadable kernel modules** The kernel has a set of core components and links in additional services via modules, either at boot time or during run time. This type of design is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X, as well as Windows. The idea of the design is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running Linking services dynamically is preferable to adding new features directly to the kernel, which would require recompiling the kernel every time a change was made. **Advantages** Has defined, protected interfaces; More flexible than a layered system More efficient, because modules do not need to invoke message passing in order to communicate. **Example:**Solaris

	 **Figure 2.15** Solaris loadable modules. **5. Hybrid Systems Approach** Combine different structures, resulting in hybrid systems that address performance, security, and usability issues. Three hybrid systems: the Apple Mac OS X operating system and the two most prominent mobile operating systems—iOS and Android. **Android** The Android operating system was designed by the Open Handset Alliance (led primarily by Google) and was developed for Android smartphones and tablet computers. Android runs on a variety of mobile platforms and is open-sourced, Android is similar to iOS in that it is a layered stack of software  **Figure 2.18** Architecture of Google's Android.
10	**Describe System Calls, System Programs in detail with neat sketch (April 2017) and OS Generation (Nov 2015)**

System calls

They are routines/codes that provide interface between programs and services made available by the OS. They are routines written in C, C++, low level tasks in assembly language. Even Simple programs heavily use OS by calling thousands of system calls per second. The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.

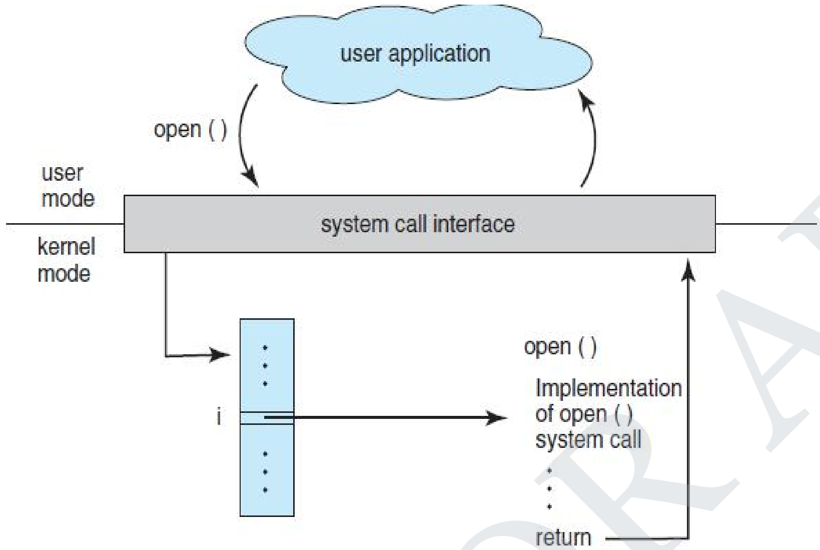


Figure 2.6 The handling of a user application invoking the open() system call.

System Programs

Also called as System Utilities

Provide a convenient environment for program development and execution

Some are user interfaces to s/m calls or bundle of useful s/m calls

Application Programs

System Programs

Operating System

Hardware

System programs can be divided into these categories:

File Management

Programs to create, delete, print, copy, list, manipulate files and directories etc

	Status Information Maintains and provides information on system Date, time, disk/memory space, logging, performance, debugging, registry File Modification Text editors create, modify and search file contents Programming language Support Softwares like Compilers, assemblers, debuggers, interpreters for common programming languages (C,C++,Java) Program Loading & Execution Programs of Absolute/relocatable loaders, linkage editors Communication These programs provide Virtual connections, remote login, file transfer, web browsing, email communication Background Services -They are constantly running programs Service/subs-systems/daemons **OPERATING-SYSTEM GENERATION** **The process of configuring or generating a system for a specific computer site based on its purpose is known as System Generation SYSGEN** SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system Following are kinds of information must be determined by SYSGEN: Type of CPU and options available in it Format of the boot disk-sections, partitions Memory Space available in the system Reference memory locations to find the illegal address in the memory Devices available in the system Address of the device, Device interrupt number, Device type and model and specific characteristics of each device Operating System options desired Type of scheduling algorithm used

	Number of buffers and their sizes Maximum number of processes supported by the system etc. Once this information is determined, it can be used in several ways. (1)**Manual Approach**: A system administrator can use it to modify a copy of the source code of the operating system. (2)**Modular Approach** : Second option is creation of tables and the selection of modules from a precompiled library. These modules are linked together to form the generated operating system. (3)**Tabular Approach:** It is possible to construct a system that is completely table driven. System generation involves simply creating the appropriate tables to describe the system.
11	**Distinguish between the client server and peer to peer models of distributed systems (April 2016)** Distributed systems use multiple central processors to serve multiple real time application and multiple users. Data processing jobs are distributed among the processors accordingly to which one can perform each job most efficiently. The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers and so on. The advantages of distributed systems are following. With resource sharing facility user at one site may be able to use the resources available at another. Speedup the exchange of data with one another via electronic mail. If one site fails in a distributed system, the remaining sites can potentially continue operating. Better service to the customers.

Reduction of the load on the host computer.

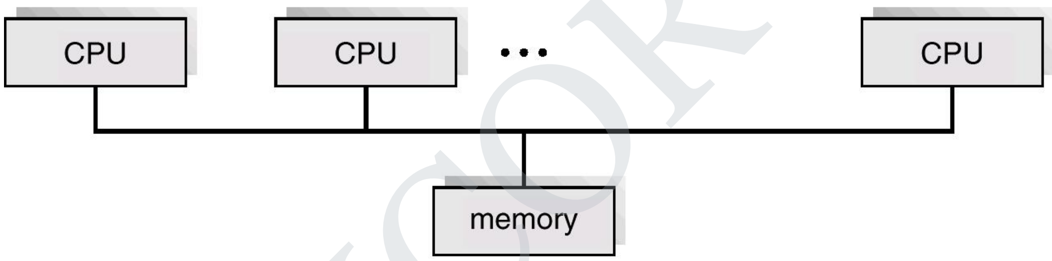
Reduction of delays in data processing.

Client-Server

The client-server model is probably the most popular paradigm. The server is responsible for accepting, processing, and replying to requests. It is the producer. The client is purely the consumer. It requests the services of the server and accepts the results.

The basic web follows the client-server model. Your browser is the client. It requests web pages from a server (e.g., google.com), waits for results, and displays them for the user.

In some cases, a web server may also act as a client. For example, it may act as a client of DNS or may request other web pages.



Peer to Peer

The peer-to-peer model assumes that each entity in the network has equivalent functionality. In essence, it can play the role of a client or a server. Ideally, this reduces bottlenecks and enables each entity to contribute resources to the system. Unfortunately, it doesn't always work that way.

In addition, enabling communication in such a system is challenging. First, peers must locate other peers in order to participate in the system. This is rarely done in a truly distributed or peer-to-peer fashion. Searching for content or other resources is the second big challenging in implementing peer-to-peer systems. It can be very inefficient to locate resources in a peer-to-peer system and a hybrid, or partially centralized, solution is often employed.

Hierarchical or super peer systems, like Skype, are also widely used. In these systems, peers are organized in a tree-like structure. Typically, more capable peers are elected to become superpeers (or supernodes). Superpeers act on behalf of downstream peers and can reduce communication overhead.

Describe differences between symmetric and asymmetric multiprocessing. What are the three advantages and disadvantages of multiprocessor system (April 2016)

Multiprocessor Systems

also known as parallel systems or tightly coupled systems

more than one processor in close communication, sharing the computer bus, the clock, memory and peripheral devices

There are 2 types of multi-processing : symmetric and asymmetric multiprocessing

(i) Symmetric multiprocessing (SMP)

- Each processor runs an identical copy of OS concurrently and these copies communicate with one another as needed

- All processors are peers, no master-slave relationship

- Carefully control i/o to ensure that the data reach the appropriate processor

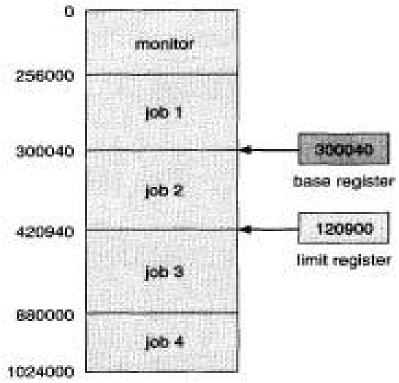
- Since CPUs are separate, one may be sitting idle while another is overloaded, resulting inefficiencies. This can be avoided if processors share certain data structures

Benefit – N processes can run simultaneously if there are N CPUs

	without causing the s ignificant deterioration of performance **(ii) Asymmetric Multiprocessing** - Each processor is assigned a specific task - Master processor – controls the system, others either look to the master for instruction of have predefined task - Master processor schedules and allocates work to the slave processors àmaster slave relationship **Advantages:** **1. Increased throughput** - More processors, more work done in less time - Speedup ratio with N-processors is not N, it is less than N - When multiple processors cooperate on a task – overhead is incurred in keeping all the parts working correctly **2. Economy of Scale** - Can save more money than multiple single-processor systems, because they can share peripherals, mass storage and power supplies - All the processors can share the data if several programs operate on same data **3. Increased reliability** - Failure of one processor will not halt the system, only slows it down - This ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation** - Systems designed for graceful degradation are also called **fault**

	tolerant **Disadvantages** Additional complexity in Operating system and possibly application software The Master CPU can become bottlenecked. The differences between symmetric and asymmetric multiprocessing is that in asymmetric there is one processor assigned as the boss and schedules and allocates work For the remaining processors, while in symmetric multiprocessing they work as peers with each of their own registers and private cache but sharing physical memory. The three main advantages for multiprocessor systems is increased throughput (more work done in less time), economy of scale (can cost less than equivalent), and increased reliability (if one processor fails the rest can still continue). A disadvantage, more specifically with symmetric multiprocessing, is that a single CPU can be sitting idle while another is overloaded, causing inefficiencies. This can be prevented with careful programming.
12	**Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs (Nov 2016)** To separate each program's memory space, we need the ability to determine the range of legal addresses that the program may access, and to protect the memory outside that space. We can provide this protection by using two registers, usually a base and a limit, as illustrated in the Figure. The base register holds the smallest legal physical memory address; the limit register contains the size of the range. For example, if the base register holds 300040 and limit register is 120900, then the program can legally access all addresses from 300040 through 420940 inclusive. This protection is accomplished by the CPU hardware comparing every address

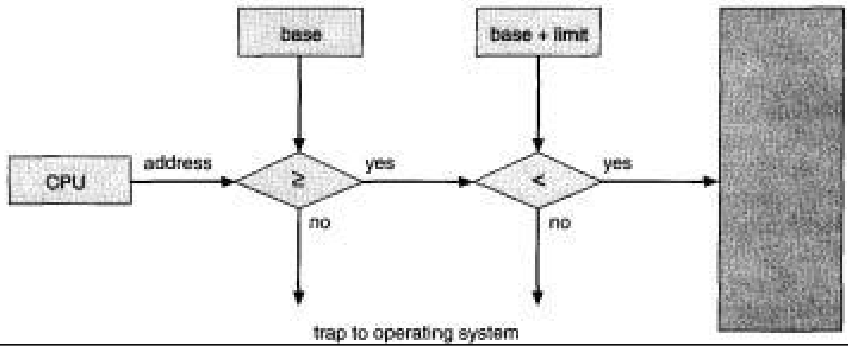
generated in user mode with the registers. Any attempt by a program executing in user mode to access monitor



memory or other users' memory results in a trap to the monitor, which treats the attempt as a fatal error (Figure 2.10). This scheme prevents the user program from (accidentally or deliberately) modifying the code or data structures of either the operating system or other users.

The base and limit registers can be loaded by only the operating system,which uses a special privileged instruction. Since privileged instructions can be executed in only monitor mode, and since only the operating system executes in monitor mode, only the operating system can load the base and limit registers. This scheme allows the monitor to change the value of the registers, but prevents user programs from changing the registers' contents.

The operating system, executing in monitor mode, is given unrestricted access to both monitor and users' memory. This provision allows the operating system to load users' programs into users' memory, to dump out those programs in case of errors, to access and modify parameters of system calls, and so on.



	What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices? (Nov 2016) __Advantages__ Each device can be accessed as though it was a file in the file system. - Since most of the kernel deals with devices through this file interface, it is easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. - This benefits the development of both a. User program code, which can be written to access devices and files in the same manner, and b. Device-driver code, which can be written to support a well-defined API. __Disadvantages__ It might be difficult to capture the functionality of certain devices within the context of the file access API, thereby resulting in either:- - Loss of functionality or a loss of performance. - Some of this could be overcome by the use of the ioctl operation that provides a general-purpose interface for processes to invoke operations on devices.
13	**State and explain the major activities of an operating system with regard to file management? (Nov 2016)** File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic tape, magnetic disk, and optical disk are the most common media. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random). For convenient use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating

system maps files onto physical media, and accesses these files via the storage devices.

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, or alphanumeric. Files may be free-form (for example, text files), or may be formatted rigidly (for example, fixed fields).

A file consists of a sequence of bits, bytes, lines, or records whose meanings are defined by their creators. The concept of a file is an extremely general one. The operating system implements the abstract concept of a file by managing mass storage media, such as disks and tapes, and the devices that control them. Also, files are normally organized into directories to ease their use.

Finally, when multiple users have access to files, we may want to control by whom and in what ways (for example, read, write, append) files may be accessed.

The operating system is responsible for the following activities in connection
with file management:

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

Discuss the different multiprocessor organizations with block diagrams (Nov 2016)

Multiple Processor Organization

Single instruction, single data stream – SISD

Single processor

Single instruction stream

Data stored in single memory

Uni-processor

Single instruction, multiple data stream – SIMD

Single machine instruction

Controls simultaneous execution

Number of processing elements

Lockstep basis

Each processing element has associated data memory

Each instruction executed on different set of data by different processors

Vector and array processors

Multiple instruction, single data stream – MISD

Sequence of data

Transmitted to set of processors

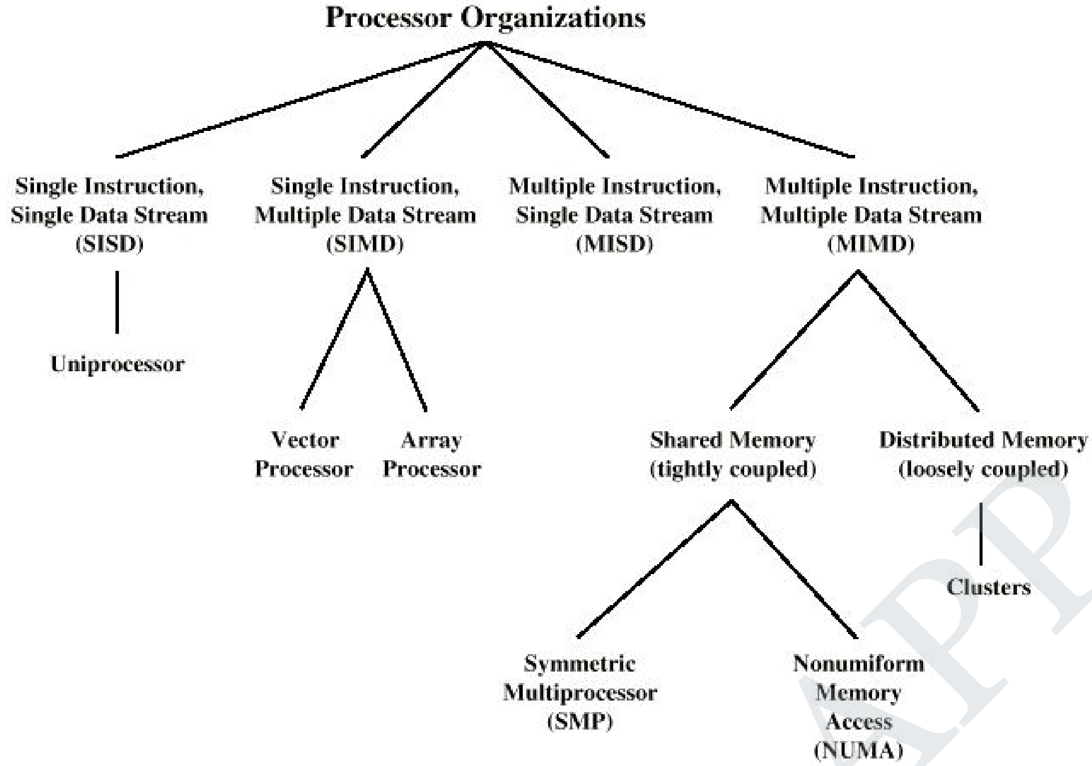
Each processor executes different instruction sequence never been implemented

Multiple instruction, multiple data stream- MIMD

Set of processors

Simultaneously execute different instruction sequences

Different sets of data SMPs, clusters and NUMA systems



Tightly Coupled – SMP

Processors share memory

Communicate via that shared memory

Symmetric Multiprocessor (SMP)

Share single memory or pool

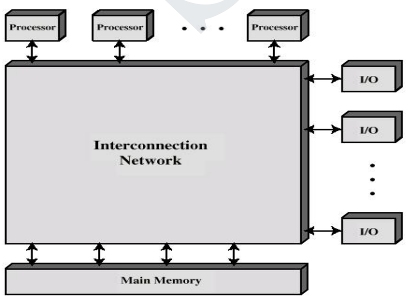
Shared bus to access memory

Memory access time to given area of memory is approximately the same for each processor

Tightly Coupled – NUMA

Nonuniform memory access

Access times to different regions of memory may differ



Loosely Coupled – Clusters

Collection of independent uniprocessors or SMPs

	Interconnected to form a cluster Communication via fixed path or network connections
14	**Explain the concept of multiprocessor and Multicore organization. (April 2017)** **Multiprocessor Systems** also known as parallel systems or tightly coupled systems more than one processor in close communication, sharing the computer bus, the clock, memory and peripheral devices There are 2 types of multi-processing : symmetric and asymmetric multiprocessing **Symmetric multiprocessing (SMP)** – Each processor runs an identical copy of OS concurrently and these copies communicate with one another as needed – All processors are peers, no master-slave relationship – Carefully control i/o to ensure that the data reach the appropriate processor – Since CPUs are separate, one may be sitting idle while another is overloaded, resulting inefficiencies. This can be avoided if processors share certain data structures Benefit – N processes can run simultaneously if there are N CPUs without causing the s ignificant deterioration of performance **Asymmetric Multiprocessing** – Each processor is assigned a specific task – Master processor – controls the system, others either look to the master for instruction of have predefined task – Master processor schedules and allocates work to the slave processors **Advantages:** 1. Increased throughput – More processors, more work done in less time – Speedup ratio with N-processors is not N, it is less than N – When multiple processors cooperate on a task – overhead is incurred in

keeping all the parts working correctly

2. Economy of Scale

- Can save more money than multiple single-processor systems, because they can share peripherals, mass storage and power supplies

- All the processors can share the data if several programs operate on same data

3. Increased reliability

- Failure of one processor will not halt the system, only slows it down

- This ability to continue providing service proportional to the level of surviving hardware is called graceful degradation

- Systems designed for graceful degradation are also called fault tolerant

Disadvantages

Additional complexity in Operating system and possibly application software

The Master CPU can become bottlenecked.

The differences between symmetric and asymmetric multiprocessing is that in asymmetric there is one processor assigned as the boss and schedules and allocates work For the remaining processors, while in symmetric multiprocessing they work as peers with each of their own registers and private cache but sharing physical memory.

The three main advantages for multiprocessor systems is increased throughput (more work done in less time), economy of scale (can cost less than equivalent), and increased reliability (if one processor fails the rest can still continue).

A disadvantage, more specifically with symmetric multiprocessing, is that a single CPU can be sitting idle while another is overloaded, causing inefficiencies. This can be prevented with careful programming.

Multicore Organization

In multicore architecture, all processors are on the same chip.

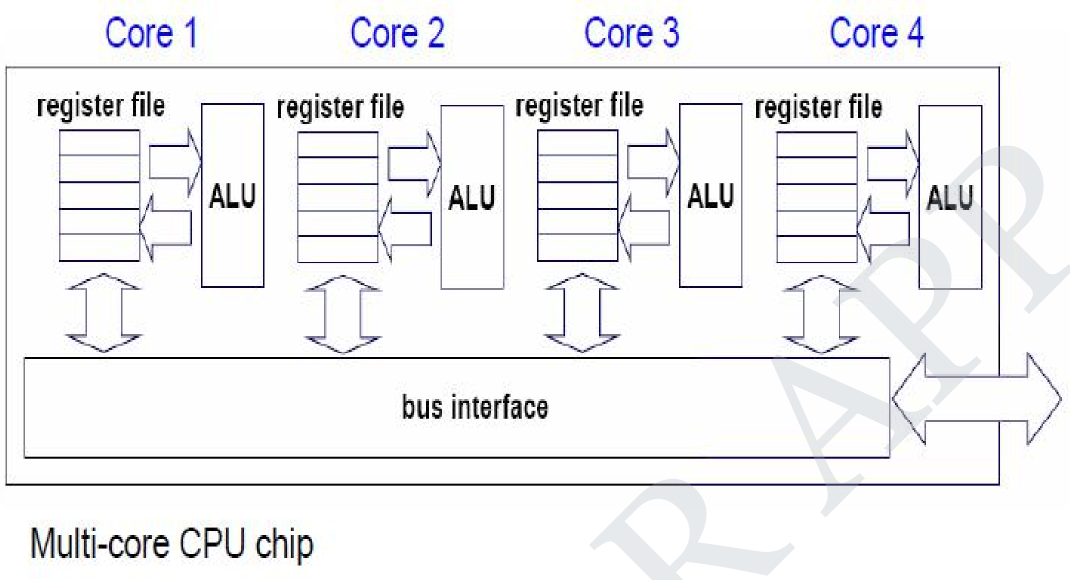
Multicore processors are MIMD

Different core execute different threads

(Multiple instructions), operating on different parts of the memory.

Multicore is a shared memory multiprocessor

All cores share the same memory.



Explain about direct memory access. **(April 2017)**

Programmed I/O (PIO) : expensive general-purpose processor used to watch status bits and to feed data into a controller register one byte at a time . Many computers avoid burdening the main CPU with PIO by offloading some of this work to a special-purpose processor called a direct-memory-access (DMA) controller.

To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred.

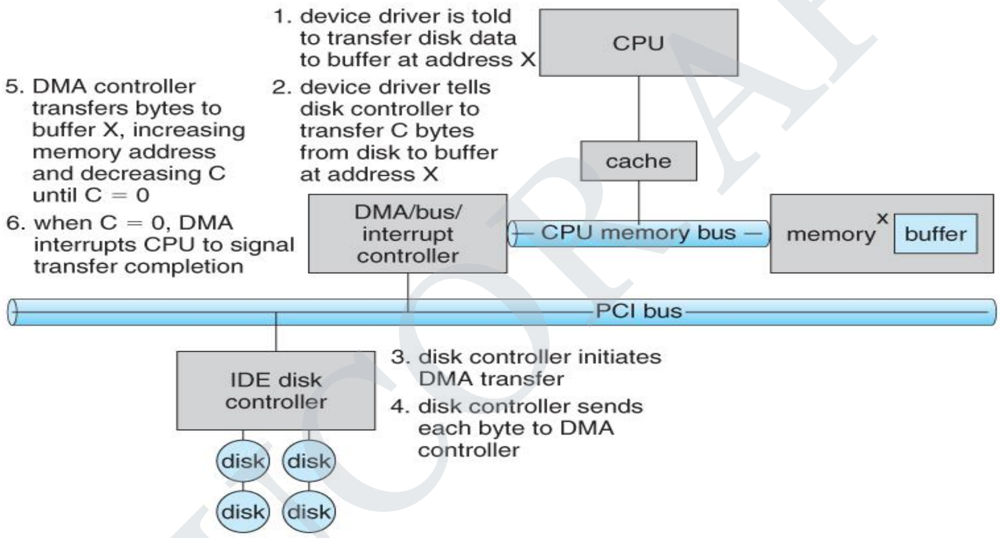
The CPU writes the address of this command block to the DMA controller, then goes on with other work.

The DMA controller proceeds to operate the memory bus directly, placing addresses on the bus to perform transfers without the help of the main CPU.

A simple DMA controller is a standard component in PCs, and bus-mastering I/O boards for the PC usually contain their own high-speed DMA hardware.

Handshaking between the DMA controller and the device controller is performed via a pair of wires called DMA-request and DMA-acknowledge.

The device controller places a signal on the DMA-request wire when a word of data is available for transfer. This signal causes the DMA controller to seize the memory bus, to place the desired, address on the memory-address wires, and to place a signal on the DMA-acknowledge wire.



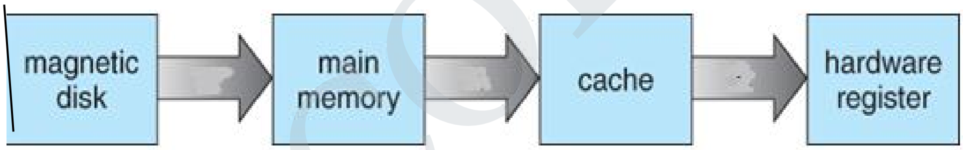
DMA Transfer Diagram

When the device controller receives the DMA-acknowledge signal, it transfers the word of data to memory and removes the DMA-request signal.

When the entire transfer is finished, the DMA controller interrupts the CPU. This process is depicted in Figure above.

While the DMA transfer is going on the CPU does not have access to the PCI bus (including main memory) but it does have access to its internal registers and primary and secondary caches

Although this cycle stealing can slow down the CPU computation, DMA

	controller generally improves the total system performance. Some computer architectures use physical memory addresses for DMA, but others perform direct virtual memory access (DVMA), using virtual addresses that undergo translation to physical addresses. DVMA can perform a transfer between two memory-mapped devices without the intervention of the CPU or the use of main memory.
15	**Give reason why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching why not make it that large and eliminate the device?(April 2018)** A cache is a **small high-speed storage space** or **temporary** storage that stores **most frequently needed** portions of larger slower storage areas Access to the cached copy is more efficient than access to the main memory. The Cache Memory is the nearest memory to the CPU, all the recent instructions are Stored into the Cache Memory. The instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches.  When CPU requests contents of memory location Check cache for the data If data found in cache the its called as "**Cache Hit**", the data is transferred to CPU for processing If data is not found in cache it is called as "**Cache Miss**", the required data is then searched in main memory or secondary memory and transferred to the cache later. **Multi-level Caches:** multi level caches are used to overcome the disadvantages of longer latency in single larger caches. Multi-level caches generally operate by checking the fastest, level 1 (L1) cache first; if it hits, the processor proceeds at high speed. If that smaller cache misses, the next fastest cache (level 2, L2) is checked, and so on, before external memory is checked. **Cache Coherence**: In a shared memory multiprocessor with a separate cache memory for each processor , it is possible to have many copies of any one instruction operand. Cache coherence ensures that changes in the values of shared

	operands are updated throughout the system. **Cache vs Buffer:** The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes. ii) Describe the major activities of operating system with regards to the file management. Magnetic tape, disk and optical disk – used to store information Each storage media has own characteristics and physical organization. OS provides uniform, logical view of information storage Abstracts physical properties to logical storage unit - file Each medium is controlled by device (i.e., disk drive, tape drive) OS maps files in to physical media & accesses these files via the storage devices File à collection of related information defined by its creator represents programs and data data files , text files File-System management Files usually organized into directories Access control on most systems to determine who can access what Role of OS in File Management Creating and deleting files and directories Primitives to manipulate files and directories Mapping files onto secondary storage Backup files onto stable (non-volatile) storage media
16	Discuss the essential properties of the following types of systems (i) time sharing systems. (ii) multiprocessor systems (iii) distributed systems (i)**Time-sharing** is a technique which enables many people, located at various terminals, to use a particular computer **system** at the same **time**. **Time-sharing** or multitasking is a logical extension of multiprogramming.

Processor's **time** which is **shared** among multiple users simultaneously is termed as **time-sharing**.

(ii) multiprocessor systems

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer **system**. The term also refers to the ability of a **system** to support more than one processor or the ability to allocate tasks between them.

(iii) distributed systems

Distributed computing is a field of computer science that studies **distributed systems**. A **distributed system** is a **system** whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to one another. Characteristics of distributed system

Resource sharing Resource: hardware - disks and printers software - files, windows, and data objects Hardware sharing for: convenience ,reduction of cost ,Data sharing for: consistency - compilers and libraries ,exchange of information - database .cooperative work - groupware

Openess

Open or closed with respect to hardware or software

Open - published specifications and interfaces - standardization of interfaces

UNIX was a relatively open operating system C language readily available System calls documented New hardware drivers were easy to add

Applications were hardware independent IPC allowed extension of services and resource

Concurrency

Multi-programming Multi-processing

Parallel executions in distributed systems

1. Many users using the same resources, application interactions

2. Many servers responding to client requests

Scalability

Small system - two computers and a file server on a single network

	Large system – current Internet Scalability – software should not change to support growth – research area – for large, high-performance networks Avoid centralization to support scalability Choose your naming or numbering scheme carefully Handle timing problems with caching and data replication **Fault Tolerance** Computers fail therefore we need: hardware redundancy ,software recovery Increase in availability for services Network is not normally redundant Program recovery via the process group **Transparency:** ccess , location , concurrency, replication , failure , migration, performance ,scaling
UNIT II PROCESS MANAGEMENT Processes-Process Concept, Process Scheduling, Operations on Processes, Interprocess Communication; Threads- Overview, Multicore Programming, Multithreading Models; Windows 7 - Thread and SMP Management. Process Synchronization - Critical Section Problem, Mutex Locks, Semaphores, Monitors; CPU Scheduling and Deadlocks.	
PART-A	
1	**Define process state?** The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states: New, running, waiting, ready, Terminated.
2	**Define IPC?** It provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.Example:

	Chat program
3	**What are the two possibilities exist in terms of execution while creating a process?** The parent continues to execute concurrently with its execution, The parent waits until some or all of its children have terminated.
4	**Define thread pool** A thread pool is a group of pre-instantiated, idle threads which stand ready to be given work. These are preferred over instantiating new threads for each task when there is a large number of short tasks to be done rather than a small number of long ones.
5	**What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?** The two models of interprocess communication are message passing model and the shared-memory model. Message passing is useful for exchanging smaller amounts of data and also easier to implement. Shared memory allows maximum speed and convenience of communication, while message passing requires the more time-consuming task of kernel intervention. Shared memory also has problems of protection and synchronization.
6	**Define Threads (May 2013)** It is also called as lightweight process (LWP), is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
7	**Give two different scenarios of cancellation?** A thread that is to be cancelled is often referred to as the target thread. Asynchronous cancellation: One thread immediately terminates the target thread. Deferred cancellation: The target thread can periodically check if it should terminate, allowing the target thread an opportunity to terminate itself in an orderly fashion.
8	**Define program and process.** A Program is a passive entity, such as the contents of a file stored on disk,

	whereas a process is an active entity, with a program counter specifying the next instruction to execute.
9	**Define dispatcher.** The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves Switching context, Switching to user mode, Jumping to the proper location in the user program to restart that program.
10	**Define Dispatch latency.** The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running.
11	**What are the Scheduling Criteria?** CPU utilization, Throughput, Turnaround time, Waiting time, Response time
12	**How will you implement FCFS policy?** It is managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.
13	**Give the two components of dispatch latency in conflict phase?** Preemption of any process running in the kernel. Release by low-priority processes resources needed by the high-priority process.
14	**Define multilevel queue- scheduling algorithm.** A multilevel queue- scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.
15	**What are the 3 requirements to solve critical-section problem?** Mutual exclusion, Progress, Bounded waiting

16	**What is the concept behind semaphore and spinlock? (Nov 2015)**
	Semaphore is a process synchronization tool used to control access of a common resource in a multiprogramming environment. Semaphore S is an integer variable. Apart from initialization accessed by 2 standard atomic operations wait() & signal().At a time only one process can modify semaphore
	Definition of wait() is as follows:
	wait(S)
	{
	while (S<=0)
	; // busy wait
	S--;
	}
	The definition of signal() is as follows:
	signal(S)
	{
	S++;
	}

17	**What are binary semaphores?** A binary semaphore is a semaphore with an integer value that can range only between 0 and 1. A binary semaphore can be simpler to implement than a counting semaphore, depending on the underlying hardware architecture.
18	**Define spin lock with its advantage? (May 2013) (Nov 2015) (April 2016)** Spin lock or Busy waiting wastes CPU cycles that some other process might be able to use productively. Advantage: Is that no context switch is required when a process must wait on a lock, and a context switch may take

	considerable time. Thus, when locks are expected to be held for short times, spin locks are useful. Not preferred in uniprocessor. Preferred in multiprocessor (as lock held for short time)
19	**Give the structure of semaphore while implementing mutual-exclusion?** do { remainder section } while (1);
20	**Define monitors? How will you represent the monitors?** A monitor is characterized by a set of programmer-defined operators. It consists of declaration of variables whose values define the state of an instance of the type, as well as the bodies of procedures or functions that implements operations on the type.
21	**List the advantages of thread?** Thread minimizes context switching time. Use of threads provides concurrency within a process, Efficient communication, Utilization of multiprocessor architecture.
22	**Give various Scheduling Algorithms?** CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU scheduling algorithms namely First Come First Serve, Shortest Job First, Priority, Round Robin, Multi level Queue, Multi level Feedback Queue
23	**Differences between a Process and Thread**

Process	Thread
Process is a heavy weight process	Thread is a lightweight process
Process switching needs interface with operating System.	Thread switching does not need to call an Operating system and cause an interrupt to the kernel

	In multiple processes, implementation of each process executes the same code but has its own memory and file resources	All thread can share same set of open files, child process
	If one server process is blocked, no other server process can execute until the first process is unblocked	When one server thread is blocked, second thread in the same task could run.
	In multiple process each process operates independently of the others	One thread can read write or completely wipe out another thread stack.
24	**Define deadlock** A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.	
25	**What is the sequence in which resources may be utilized?** Under normal mode of operation, a process may utilize a resource in the following sequence: Request: If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource. Use: The process can operate on the resource. Release: The process releases the resource.	
26	**What are conditions under which a deadlock situation may arise?** A deadlock situation can arise if the following four conditions hold simultaneously in a system: Mutual exclusion, Hold and wait, No pre-emption, Circular wait	
27	**What is a resource-allocation graph?** Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of	

	vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes; P the set consisting of all active processes in the system and R the set consisting of all resource types in the system.
28	**Define request edge and assignment edge.** A directed edge from process Pi to resource type Rj is denoted by Pi→Rj; it signifies that process Pi requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from resource type Rj to process Pi is denoted by Rj→Pi, it signifies that an instance of resource type has been allocated to a process Pi. A directed edge Pi→Rj is called a request edge. A directed edge Rj→Pi is called an assignment edge.
29	**What are the methods for handling deadlocks?** The deadlock problem can be dealt with in one of the three ways: Use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state. Allow the system to enter the deadlock state, detect it and then recover. Ignore the problem all together, and pretend that deadlocks never occur in the system.
30	**Define deadlock prevention.** Deadlock prevention is a set of methods for ensuring that at least one of the four necessary conditions like mutual exclusion, hold and wait, no pre-emption and circular wait cannot hold. By ensuring that that at least one of these conditions cannot hold, the occurrence of a deadlock can be prevented.
31	**Define deadlock avoidance.** An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested. Each request requires the system consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process, to decide whether the could be satisfied or must wait to avoid a possible future deadlock.
32	**What are a safe state and an unsafe state?**

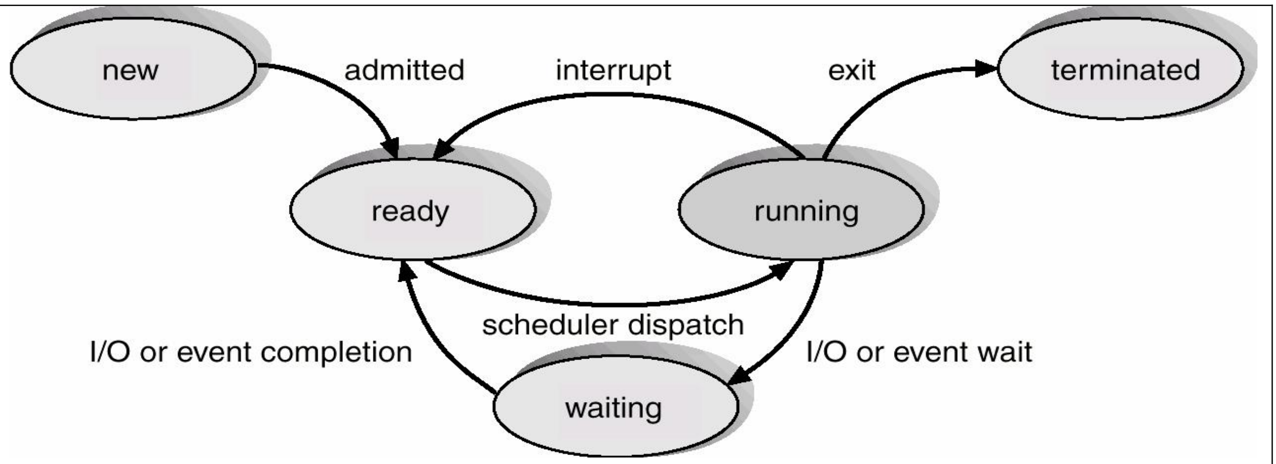
	A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. A system is in safe state only if there exists a safe sequence. A sequence of processes <P1,P2,....Pn> is a safe sequence for the current allocation state if, for each Pi, the resource that Pi can still request can be satisfied by the current available resource plus the resource held by all the Pj, with j<i. if no such sequence exists, then the system state is said to be unsafe.
33	**What can the operating system do to recover from deadlock? (Nov 2014)** Kill all deadlocked processes , Kill one deadlocked process at a time and release its resources , Steal one resource at a time, Roll back all or one of the processes to a checkpoint that occurred before they requested any resources, then continue.
34	**What are the resources required to create a thread ? (Nov 2014) (Nov 2016)** When a thread is created the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space
35	**Difference between pre-emptive and non pre-emptive scheduling? (Nov 2014)** Non-preemptive is designed so that once a process enters the running state (is allowed a process), it is not removed from the processor until it has completed its service time (or it explicitly yields the processor). Preemptive scheduling is driven by the notion of prioritized computation. The process with the highest priority should always be the one currently using the processor. If a process is currently using the processor and a new process with a higher priority enters, the ready list, the process on the processor should be removed and returned to the ready list until it is once again the highest-priority process in the system.
36	**What are the differences between user level threads and kernel level threads? Under what circumstances is one type better that the other?(Nov 2015)(April 2016)**

User Level Threads	Kernel Level Threads

<table>
<tr><td></td><td>Threads that run user programs and applications are user level threads</td><td>Threads that runs the kernel operations and kernel programs are called as kernel level threads</td></tr>
<tr><td></td><td>User threads are supported above the kernel and are managed without kernel support</td><td>kernel threads are supported and managed directly by the operating system.</td></tr>
<tr><td></td><td colspan="2">In one-one thread model, user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application
In many-one thread model developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.</td></tr>
<tr><td>37</td><td colspan="2">Distinguish between CPU bounded and I/O bounded process (Nov 2016)
The CPU-bound process will get the CPU and hold it. During this time, all the other processes will finish their I/O and move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, the I/O devices are idle. Eventually, the CPU-bound process finishes its CPU burst and moves to an I/O device. All the I/O-bound processes, which have very short CPU bursts, execute quickly and moves back to the I/O queues. At this point, the CPU sits idle. The CPU-bound process will then move back to the ready queue and be allocated the CPU.</td></tr>
<tr><td>38</td><td colspan="2">“Priority inversion is a condition that occurs in real time systems where a low priority is starved because higher priority processes have gained hold of the CPU” – Comment on this statement. (April 2017)
Priority inversion can occur without causing immediate harm—the delayed execution of the high priority task goes unnoticed, and eventually the low priority task releases the shared resource. However, there are also many situations in which priority inversion can cause serious problems. If the high priority task is left starved of the resources, it might lead to a system malfunction.
Priority inversion can also reduce the perceived performance of the system. Low</td></tr>
</table>

	priority tasks usually have a low priority because it is not important for them to finish promptly. Similarly, a high priority task has a high priority because it is more likely to be subject to strict time constraints—it may be providing data to an interactive user, or acting subject to real time response guarantees.	
39	**Differentiate single threaded and multi- threaded processes (April 2017)**	
	Multithreaded Programming	**Single Threaded Programming**
	In this type of programming multiple threads run at the same time	In this type of programming a single thread runs at a time.
	Multithreaded model doesn't use event loop with polling	Single threaded model uses a process event loop with polling
	CPU time is never wasted.	CPU time is wasted.
	Idle time is minimum.	Idle time is more.
	It results in more efficient programs.	It results in less efficient programs.
	When one thread is paused due to some reason, other threads run as normal.	When one thread is paused, the system waits until this thread is resumed.
40	**What are the benefits of synchronous and asynchronous communication?(Apr/May 2018)** Interfaces for the O.S that enable I/O devices to be treated in a standard, uniform way. I/O system calls encapsulate device behaviors in generic classes Device-driver layer hides differences among I/O controllers from I/O subsystem of the kernel Devices vary in many dimensions Character-stream or block Sequential or random-access Synchronous or asynchronous Sharable or dedicated Speed of operation read-write, read only, or write only	

41	***Give an programming example in which multithreading does not provide better performance than a single threaded solutions.(Apr/May 2018)*** Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.
42	***What is the meaning of term busy waiting.(Nov/Dec'18)*** Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor.
43	***Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system?(Nov/Dec'18)*** A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.
	Part B
1	**Draw the state diagram of a process from its creation to termination, including ALL transitions, and briefly elaborate every state and every transitions (Nov 2014)** ***Process State*** As a process executes, it changes state **new**: The process is being created. **running**: Instructions are being executed. **waiting**: The process is waiting for some event to occur. **ready**: The process is waiting to be assigned to a processor ***terminated***: The process has finished execution.



What are threads? Why are they required? Discuss and differentiate between kernel level and user level thread (Nov 2014)

A thread, sometimes called a lightweight process(LWP), is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set and a stack. A thread shares with other threads belonging to the same process its code section, data section and other operating-system resources, such as open files and signals. If the process has multiple threads of control, it

can do more than one task at a time.

Why Threads?

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

A process with multiple threads make a great server for example printer server.

Because threads can share common data, they do not need to use interprocess communication.

Because of the very nature, threads can take advantage of multiprocessors.

Threads are cheap in the sense that

They only need a stack and storage for registers therefore, threads are cheap to create.

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.

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. **User Level Threads Vs Kernel Supported Threads** User threads are supported above the kernel and are implemented by a thread library at the user level. Whereas, kernel threads are supported directly by the operating system. For user threads, the thread library provides support for thread creation, scheduling and management in user space with no support from the kernel as the kernel is unaware of user-level threads. In case of kernel threads, the kernel performs thread creation, scheduling and management in kernel space. As there is no need of kernel intervention, user-level threads are generally fast to create and manage. As thread management is done by the operating system, kernel threads are generally slower to create and manage that are user threads. If the kernel is single-threaded, then any user-level thread performing blocking system call, will cause the entire process to block, even if other threads are available to run within the application However, since the kernel is managing the kernel threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. User-thread libraries include POSIX P threads, Mach C-threads and Solaris 2 UI-threads.Some of the cotemporary operating systems that support kernel threads are Windows NT, Windows 2000, Solaris 2, BeOS and Tru64 UNIX(formerly Digital UNIX).
2	**What are interacting processes? Explain any two methods of implementing interacting processes. (Nov 2014)** **Interacting processes:** The concurrent processes executing in the operating system are interacting or cooperating processes if they can be affected by each other. Any process that shares data with other processes is an interacting process. Two methods of implementing interacting process are as follows: Shared memory solution: This scheme requires that these processes share a common buffer pool and

the code for implementing the buffer be written by the application programmer. For example, a shared-memory solution can be provided to the bounded-buffer problem. The producer and consumer processes share the following variables:

```
#define BUFFER_SIZE 10
Typedef struct{
-----}item;
Item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

The shared buffer is implemented as a circular array with two logical pointers: in and out. The variable in points to the next free position in the buffer; out points to the first full position in the buffer. The buffer is empty when in==out; the buffer is full when ((in + 1)%BUFFER_SIZE)==out. The producer process has a local variable nextProduced in which the new item to be produced is stored:

```
while(1){
/* produce and item in nextProduced */
While(((in + 1)%BUFFER_SIZE)==out)
; // do nothing
Buffer[in]=nextProduced;
in =(in+1)% BUFFER_SIZE;}
```

The consumer process has a local variable nextConsumed in which the item to be consumed is stored:

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

Inter process Communication:

The OS provides the means for cooperating processes to communicate with each other via an interprocess communication (IPC) facility. IPC provides a mechanism to allow processes

	to communicate and to synchronize their actions without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on differentcomputers connected with a network. IPC is best implemented by message passing system where communication among the user processes is accomplished through the passing of messages. An IPC facility provides at least the two operations: send(message) and receive(message). Some types of message passing system are as follows: Direct or Indirect Communication: With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send and receive primitives are defined as: •send(P, message)- Send a message to process P. •receive(Q, message)- Receive a message from process Q. A communication link in this scheme has the following properties: A link is established automatically between every p air of processes that want to communicate. The processes need to know onl y each other's identity to communicate. •A link is associated with exactly two processes. •Exactly one link exists between each pair of processes. With indirect communication, the messages are sent to and received from mailboxes, or ports. Each mailbox has a unique identification. In this scheme, a process can communicate with some other process via a number of different DC14 System Software and Operating System45 mailboxes. Two processes can communicate only if they share a mailbox. The send and receive primitives are defined as follows: •send (A, message)- Send a message to mailbox A •receive (A, message)- Receive a message from mailbox A. In this scheme, a communication link has the following properties: •A link is established between a pair of processes only if both members of the pair have a shared mailbox. •A link may be associated with more than two processes.

	•A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox
3	**Explain in detail about the different multi threading models and SMP Management with neat diagram. (Nov 2014) (Nov 2015)**

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

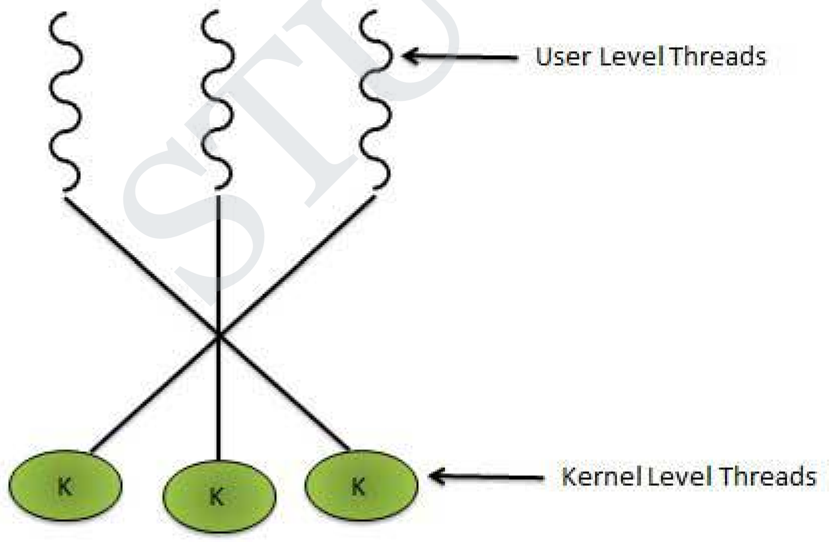
Many to many relationship.

Many to one relationship.

One to one relationship.

Many to Many Model

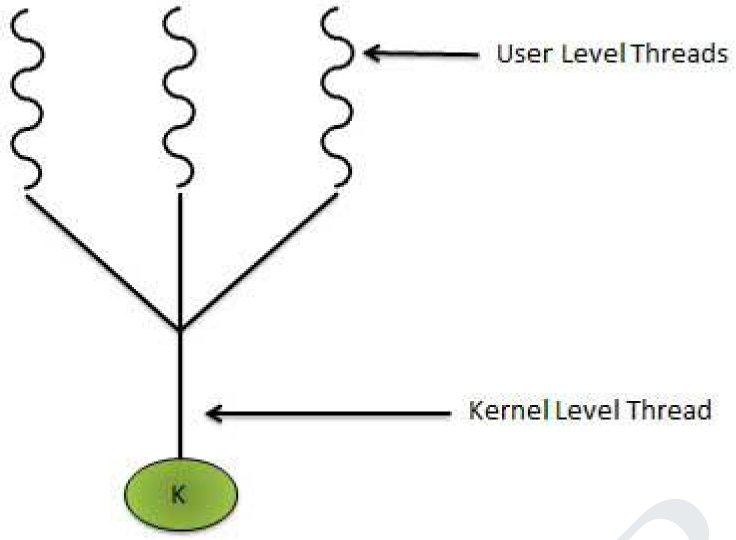
In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine. Following diagram shows the many to many model. In this model, developers can create as many user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



Many to One Model

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

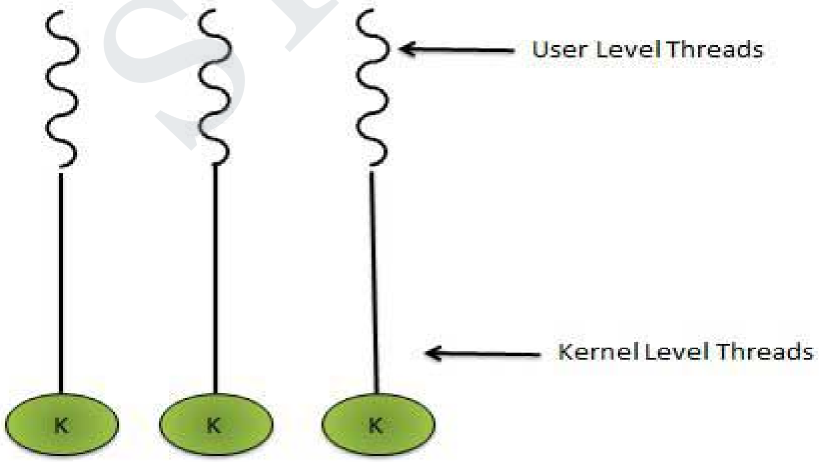
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model

There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



SMP MANAGEMENT

<u>Symmetric Multi Processors(SMP)</u>

SMP can be defined as Stand-alone s/m with following characteristics:

2 or more processors with **comparable compatibility**

Processors share **same main memory & I/O facilities** which are interconnected by a bus.

Memory Access Time is approximately same for each processor

All processors **share access to I/O devices** through same or different channels

All processor can perform **same functions**

Integrated OS provides interaction b/w processors, schedule jobs, manage task, file and data elements

SMP-Organization

Each processor has its own control unit, ALU, Registers and caches

Each has access to shared memory & I/O via shared bus

Processor communicate via

Messages & status info in Shared address spaces in main memory

Also exchange signals among them directly

Simultaneous access to separate blocks of memory

Cache coherence be maintained in SMP: An **update of a block in cache** should be **reflected** in all copies of the block in **other memories and caches**

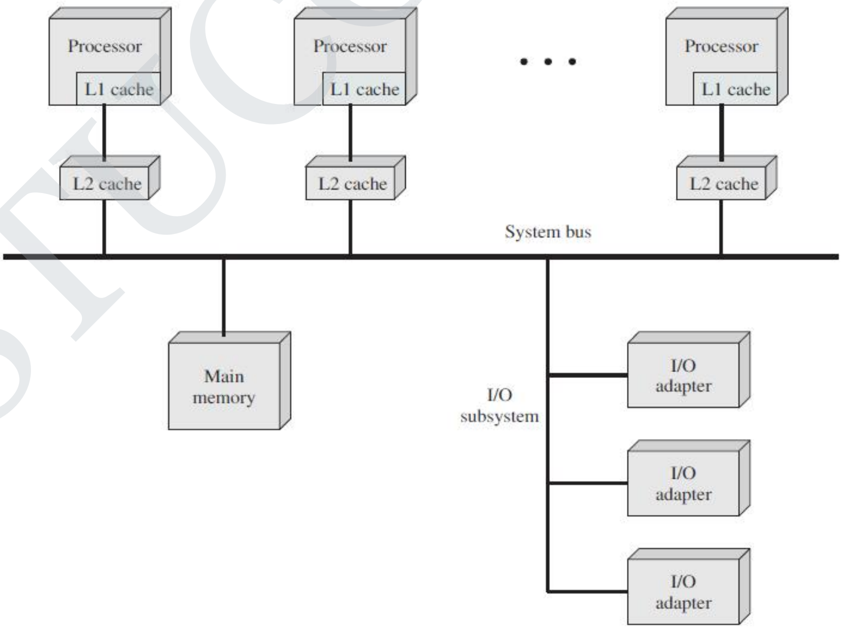


Figure 1.19 Symmetric Multiprocessor Organization

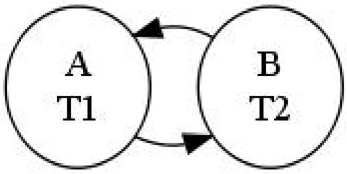
Advantages of SMP over Uniprocessor

Performance-tasks shared and executed in parallel by many processors yield better performance than tasks done by a single processor

Availability-failure of a single processor will not halt the machine

	Incremental growth-user can add additional processor and enhance performance **Scaling**-based on processors vendors can offer range of products with different price and characteristics.
4	**Explain in detail about any three policies for process scheduling that uses resource consumption information. What is response ratio with suitable examples. (Nov 2014)** Three policies for process scheduling are described below in brief: First-come First-served (FCFS) (FIFO) – Jobs are scheduled in order of arrival – Non-preemptive • Problem: – Average waiting time can be large if small jobs wait behind long ones – May lead to poor overlap of I/O and CPU and convoy effects 2 . Shortest Job First (SJF) – Choose the job with the shortest next CPU burst – Provably optimal for minimizing average waiting time • Problem: – Impossible to know the length of the next CPU burst 3. Round Robin(RR) – Often used for timesharing – Ready queue is treated as a circular queue (FIFO) – Each process is given a time slice called a quantum – It is run for the quantum or until it blocks – RR allocates the CPU uniformly (fairly) across all participants. If average queue length is n, each participant gets 1/n – As the time quantum grows, RR becomes FCFS – Smaller quanta are generally desirable, because they improve response time • Problem: – Context switch overhead of frequent context switch
5	**What is a Deadlock? How does a deadlock occur? How can a system recover from deadlock?**

A set of two or more processes are deadlocked if they are blocked (i.e., in the waiting state) each holding a resource and waiting to acquire a resource held by another process in the set.(or)
A process is deadlocked if it is waiting for an event which is never going to happen.



Example:
a system has two tape drives
two processes are deadlocked if each holds one tape drive and has requested the other
Example: semaphores A and B, each initialized to 1:

```
    P_0          P_1
     ---          ---
  A.wait();     B.wait();
  B.wait();     A.wait();
  A.signal();   B.signal();
  B.signal();   A.signal();
```

Deadlock depends on the dynamics of the execution. Illustrates that it is difficult to identify and test for deadlocks which may occur only under certain circumstances.
System model:
resource types: R_1, R_2, ..., R_n
each resource R has W_i instances
each process utilizes a resource as follows:
// request (e.g., open() system call)
// use
// release (e.g., close() system call)
Any instance of a resource type can be used to satisfy a request of that resource.

Conditions Necessary for Deadlock

All of the following four necessary conditions must hold simultaneously for

	deadlock to occur: **mutual exclusion:** only one process can use a resource at a time. **hold and wait:** a process holding at least one resource is waiting to acquire additional resources which are currently held by other processes. **no preemption:** a resource can only be released voluntarily by the process holding it. **circular wait:** a cycle of process requests exists (i.e., P_0 is waiting for a resource hold by P_1 who is waiting for a resource held by P_j ... who is waiting for a resource held by $P_{(n-1)}$ which is waiting for a resource held by P_n which is waiting for a resource held by P_0). Circular wait implies the hold and wait condition. Therefore, these conditions are not completely independent. **Deadlock Recovery** How to deal with deadlock: inform operator and let them decide how to deal with it manually let the system *recover* from the deadlock automatically: abort or more of the deadlocked processes to break the circular wait preempt some resources from one or more of the processes to break the circular wait **Process termination** Aborting a process is not easy; involves clean-up (e.g., file, printer). abort all deadlocked processes (disadvantage: wasteful) abort one process at a time until the circular wait is eliminated disadvantage: lot of overhead; must re-run algorithm after each kill how to determine which process to terminate? minimize cost priority of the process how long has it executed? how much more time does it need? how many and what type of resources has the process used? how many more resources will the process need to complete? how many processes will need to be terminated? is the process interactive or batch? **Resource Preemption** Incrementally preempt and re-allocate resources until the circular wait is broken.

	selecting a victim rollback: what should be done with process which lost the resource? clearly it cannot continue; must rollback to a safe state (???) => total rollback starvation: pick victim only small (finite) number of times; use number of rollbacks in decision

6	**What is the need for process synchronization? Explain critical section problem with a two process.**
	Process synchronization is to ensure data consistency for concurrent access to shared data.
	Critical Section Problem:
	Three Requirements
	1.Mutual Exclusion
	a.Only one process can be in its critical section.
	2.Progress
	a.Only processes not in their remainder section can decide which will enter its critical section.
	b.The selection cannot be postponed indefinitely.
	3.Bounded Waiting
	a.A waiting process only waits for a bounded number of processes to enter their critical sections.
	Notation
	Processes Pi and Pj, where j=1-i;
	Assumption
	Every basic machine-language instruction is atomic.
	Algorithm 1
	Idea: Remember which process is allowed to enter its critical section, That is, process i can enter its critical section if turn = i.
	do {
	while (turn != i) ;
	critical sectionturn=j;
	remainder section
	} while (1);
	Algorithm 1 fails the progress requirement

	Algorithm 2
	Idea: Remember the state of each process.
	flag[i]==true
	Pi is ready to enter its critical section.
	Algorithm 2 fails the progress requirement when flag[0]==flag[1]==true;
	Initially, flag[0]=flag[1]=false
	do {
	flag[i]=true;
	while (flag[j]) ;
	critical sectionflag[i]=false;
	remainder section
	} while (1);
	Algorithm 3
	Idea: Combine the ideas of Algorithms 1 and 2
	When (flag[i] && turn=i), Pjmust wait.
	Initially, flag[0]=flag[1]=false, and turn = 0 or 1
	do{
	flag[i]=true;
	turn=j;
	while (flag[j] && turn==j) ;
	critical sectionflag[i]=false;
	remainder section
	} while (1);
	Properties of Algorithm 3
	Mutual Exclusion
	The eventual value of turndetermines which process enters the critical section.
	Progress
	A process can only be stuck in the while loop, and the process which can keep it waiting must be in its critical sections.
	Bounded Waiting
	Each process wait at most one entry by the other process.

7	**Consider the following set of processes, with the length of the CPU-burst**

time given in milliseconds: (Nov 2015,Apr 2018)

Process Burst Time Priority

P1 10 3

P2 1 1

P3 2 3

P4 1 4

P5 5 2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, 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)?

1. FCFS:

```
-----------------------------------------------------------------
|        P1        | P2 |    P3    | P4 |        P5        |
-----------------------------------------------------------------
0                 10   11         13   14                 19
```

Performance statistics:

Process	Arrival Time	Burst Time	Priority	Finish Time	Turnaround Time	Waiting Time
P_1	0	10	3	10	10	0
P_2	0	1	1	11	11	10
P_3	0	2	3	13	13	11
P_4	0	1	4	14	14	13
P_5	0	5	2	19	19	14
Average					13.4	9.6

2. SJF (nonpreemptive):

```
------------------------------------------------------------
| P2 | P4 |   P3   |     P5     |           P1            |
------------------------------------------------------------
0    1    2        4            9                         19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	19	19	9
P_2	0	1	1	1	1	0
P_3	0	2	3	4	4	2
P_4	0	1	4	2	2	1
P_5	0	5	2	9	9	4
Average					7	3.2

4. Nonpreemptive, Priority:

```
------------------------------------------------------------
| P2 |     P5     |          P1           |   P3   | P4 |
------------------------------------------------------------
0    1            6                       16       18   19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	16	16	6
P_2	0	1	1	1	1	0
P_3	0	2	3	18	18	16
P_4	0	1	4	19	19	18
P_5	0	5	2	6	6	1
Average					13.4	8.2

5. Round Robin:

```
-------------------------------------------------------------------------------------
| P1 | P2 | P3 | P4 | P5 | P1 | P3 | P5 | P1 | P5 | P1 | P5 | P1 | P5 | P1 | P1 | P1 | P1 | P1 |
-------------------------------------------------------------------------------------
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19
```

Performance statistics:

Process	*Arrival Time*	*Burst Time*	*Priority*	*Finish Time*	*Turnaround Time*	*Waiting Time*
P_1	0	10	3	19	19	9
P_2	0	1	1	2	2	1
P_3	0	2	3	7	7	5
P_4	0	1	4	4	4	3
P_5	0	5	2	14	14	9
Average					9.2	5.4

8 **xplain Banker's Algorithm in detail.**

Banker's algorithm – a request for R is made. Let n be the number of

	processes in the system, and m be the number of resource types.
	Define:
	available[m]: the number of units of R currently unallocated (e.g., available[3] = 2)
	max[n][m]: describes the maximum demands of each process (e.g., max[3][1] = 2)
	allocation[n][m]: describes the current allocation status (e.g., allocation[5][1] = 3)
	need[n][m]: describes the *remaining* possible need (i.e., need[i][j] = max[i][j] – allocation[i][j])
	Resource-request algorithm:
	Define:
	request[n][m]: describes the current outstanding requests of all processes (e.g., request[2][1] = 3)
	If request[i][j] <= need[i][j], to to step 2; otherwise, raise an error condition.
	If request[i][j] > available[j], then the process must wait.
	Otherwise, *pretend* to allocate the requested resources to P_i :
	available[j] = available[j] – request[i][j]
	allocation[i][j] = allocation[i][j] + request[i][j]
	need[i][j] = need[i][j] – request[i][j]
	Once the resources are *allocated*, check to see if the system state is safe. If unsafe, the process must wait and the old resource-allocated state is restored.
	Safety algorithm (to check for a safe state):
	Let work be an integer array of length m, initialized to available.
	Let finish be a boolean array of length n, initialized to false.
	Find an i such that both:
	finish[i] == false
	need[i] <= work
	If no such i exists, go to step 4
	work = work + allocation[i];
	finish[i] = true;
	Go to step 2
	If finish[i] == true for all i, then the system is in a safe state, otherwise

	unsafe. Run-time complexity: $O(m \times n^2)$. ii) The Operating System contains 3 resources, the number of instances of each resource type are 7, 7, 10 .The current resource allocation state is shown below **(Nov 2015)** Is the current allocation in a safe state?

	Current Allocation			Maximum Need		
	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

Process	Current Allocation			Maximum Need			Need=Max-Allocation			Available=Total Instances-Total Resource Allocation (7 7 10 - 5 4 10)		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	3	3	6	8	1	4	5	2	3	0
P2	2	0	3	4	3	3	2	3	0			
P3	1	2	4	3	4	4	2	2	0			

9 **Explain in detail the classical critical section problems**

Producer Consumer problem

Concurrent access to shared data may result in data inconsistency

Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer

producer.

```
do {
produce an item in nextp;
.......
wait(empty); /* control buffer availability */
wait(mutex); /* mutual exclusion */
......
add nextpto buffer;
signal(mutex);
signal(full); /* increase item counts */
} while (1);
```

Consumer:

```
do {wait(full); /* control buffer availability */
wait(mutex); /* mutual exclusion */
.......
remove an item from buffer to nextp;
......
signal(mutex);signal(empty); /* increase item counts */
consume nextp;
} while (1);
```

Readers and Writers

The ***first* readers-writers problem,** requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object.

The ***second* readers-writers** problem requires that, once a writer is ready, that writer performs its write as soon as possible.

The reader processes hare the following data structures:

semaphoremutex, wrt;

int readcount;

The semaphores mutex and wrt are initialized to 1; readcount is initialized to 0

Writer:

```
wait(wrt);
......
writing is performed
```

```
......
signal(wrt)
Reader:
wait(mutex);
readcount++;
if(readcount==1)
wait(wrt);
signal(mutex);
......reading......
wait(mutex);
readcount--;
 if (readcount== 0)
signal(wrt);
signal(mutex);
```

Classical Synchronization Problems –Dining-Philosophers

A philosopher tries to grab a chopstick by executing a wait () operation on that semaphore; she releases her chopsticks by executing the signal() operation on the appropriate semaphores. Thus, the shared data are semaphore **chopstick[5]**; where all the elements of chopstick are initialized to **1**.

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

Solutions to Deadlocks:

- At most four philosophers appear.
- Pick up two chopsticks "simultaneously".
- Order their behaviors, e.g., odds pick up their right one first, and evens pick up their left one first.

	Solutions to Starvation: No philosopher will starve to death

10	**Demonstrate that monitors and semaphores are equivalent in so far as they can be used to implement the same types of synchronization problems.**
	A semaphore can be implemented using the following monitor code:
	monitor semaphore {
	int value = 0;
	condition c;
	semaphore increment() {
	value++;
	c.signal();
	}
	semaphore decrement() {
	while (value == 0)
	c.wait();
	value--;
	}
	}
	
	A monitor could be implemented using a semaphore in the following manner. Each condition variable is represented by a queue of threads waiting for the condition. Each thread has a semaphore associated with its queue entry. When a thread performs a wait operation, it creates a new semaphore (initialized to zero), appends the semaphore to the queue associated with the condition variable, and performs a blocking semaphore decrement operation on the newly created semaphore. When a thread performs a signal on a condition variable, the first process in the queue is awakened by performing an increment on the corresponding semaphore.
	Describe how you could obtain a statistical profile of the amount of time spent by a program executing different parts of its code. Discuss the importance of obtaining this information.
	One could issue periodic timer interrupts and monitor what instructions or

	what sections of code are currently executing when the interrupts are delivered. A statistical profile of which pieces of code were active should be consistent with the time spent by the program in different sections of its code. Once such a statistical profile has been obtained, the programmer could optimize those sections of code that are consuming more of the CPU resources.
11	**Explain operation on processes in detail** **Process Creation** A process may create several new processes, via a create-process system call, during execution. •Parent process creates children processes, which, in turn create other processes, forming a tree of processes. •Resource sharing, such as CPU time, memory, files, I/O devices ...– Parent and children share all resources. – Children share subset of parent's resources. – Parent and child share no resources When a process creates a new process, two possibilities exist in terms of execution: – Parent and children execute concurrently. – Parent waits until children terminate. •There are also two possibilities in terms of the address space of the new process: – Child duplicate of parent. – Child has a program loaded into it. •UNIX examples Fork system call creates new process execve system call used after a forkto replace the process' memory space with a new program. Process Termination Process executes last statement and asks the operating system to delete it by using the exit system call. – Output data from child to parent via wait system call. – Process' resources are deallocated by operating system. •Parent may terminate execution of children processes via abort system call for a variety of reasons, such as:

	– Child has exceeded allocated resources.
	– Task assigned to child is no longer required.
	– Parent is exiting, and the operating system does not allow a child to continue if its parent terminates
	12. i) What are semaphores? How do they implement mutual exclusion? (Nov 2014, Nov 2015)
	Synchronization tool that does not require busy waiting
	Semaphore S
	integer variable
	Two standard operations modify S: wait() and signal()
	Originally called P() and V()
	Less complicated
	Can only be accessed via two indivisible (atomic) operations
	
	wait (S) {
	while S <= 0
	; // no-op
	S--;
	}
	
	signal (S) {
	S+=;
	}

	12. ii) Give a solution for readers writers problem using conditional critical region (Nov 2014) A semaphore is a shared integer variable with non negative values which can only be subjected to the following operations Initialization Indivisible operations P and V Indivisibility of P and V operations implies that these operations cannot be executed concurrently. This avoids race conditions on the semaphore. Semantics of P and V operations are as follows: P(S): If S > 0

	then S := S−1 else block the process executing the P operation; V(S): if there exist process(es) blocked on S then wake one blocked process; else S := S + !; Readers - writers problem : Let a data object (such as a file or record) is to be shared among several concurrent processes. Readers are the processes that are interested in only reading the content of shared data object. Writers are the processes that may want to update (that is, to read and write) the shared data object. If two readers access the shared data object simultaneously, no adverse effects will result. However if a writer and some other process (either a reader or writer) access the shared object simultaneously, anomaly may arise. To ensure that these difficulties do not arise, writers are required to have exclusive access to the shared object. This synchronization problem is referred to as the readers - writers problem. Solution for readers - writers problem using conditional critical regions Conditional critical region is a high-level synchronization construct. We assume that a process consists of some local data, and a sequential program that can operate on the data. The local data can be accessed by only the sequential program that is encapsulated within same process. One process cannot directly access the local data of another process. Processes can, however, share global data. Conditional critical region synchronization construct requires that a variable v of type T, which is to be shared among many processes, be declared as v: shared T; The variable v can be accessed only inside a region statement of the following form: region v when B do S; This construct means that, while statement S is being executed, no other process can access the variable v. When a process tries to enter the critical - section region, the Boolean expression B is evaluated. If the expression is

	true, statement S is executed. If it is false, the process releases the mutual exclusion and is delayed until B becomes true and no other process is in the region associated with v.
	Now, let A is the shared data object. Let readcount is the variable that keeps track of how many processes are currently reading the object A. Let writecount is the v ariable that keeps track of how many processes are currently writing the object A. Only one writer can update object A, at a given time. Variables readcount and writecount are initialized to 0. A writer can update the shared object A when no reader is reading the object A.
	region A when(readcount = = 0 AND writecount = = 0){
	……
	writing is performed
	……
	}
	A reader can read the shared object A unless a writer has obtained permission to update the object A.
	region A when(readcount >=0 AND writecount = = 0){
	……
	reading is performed ……
	}

13	**Show how wait() and signal() semaphore operations could be implemented in multiprocessor environments, using the Test and Set() instruction. The solution should exhibit minimal busy waiting. Develop Pseudo code for implementing the operations. (April 2015)** To overcome the need for busy waiting the wait and signal semaphore operations are used. When a process executes the wait operation and finds that the semaphore value is not positive, it must wait. However, rather than busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then, control is transferred to the CPU scheduler, which selects another process to execute. A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal operation. The process

is restarted by a wakeup operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.(The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU-scheduling algorithm.)

Each semaphore has an integer value and a list of processes. When a process must wait on a semaphore, it is added to the list of processes. A signal operation removes one process from the list of waiting processes and awakens that process.

The wait semaphore operation can be defined as

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

The signal semaphore operation can now be defined as

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

The block of operation suspends the process that invokes it. The wakeup (P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls. Although under the classical definition of semaphores with busy waiting the semaphore value is never negative, this implementation may have negative semaphore values. If the semaphore value is negative, its magnitude is the number of process waiting on that semaphore.

ss about the issues to be considered with multithreaded programs (April 2015)

System Calls – One of the issues to keep in mind is how a system call deals with threads contained in a process that is getting duplicated. Do the threads also get duplicated or does the duplicated process only posses a single thread? Some Unix systems provide the means of both methods of duplication.

Cancellations – There are times when we may want to terminate a thread before it completes its purpose. This type of termination is referred to as thread cancellation. Imagine a chess game, where multiple moves are evaluated via different threads in order to find the shortest path of victory based on possible future moves. In such scenario, since all threads are running concurrently, as soon as one thread has found a path of victory, the rest of the threads can be cancelled since the found path would be the shortest path to a checkmate. When cancelling a "target" thread, we can take on of two approaches. One is asynchronous cancellation, where one thread terminates another that could lead to orphan resources since the target thread did not have a chance to free them. And the other, deferred cancellation, where each thread keeps checking if it should terminate and if so, do so in an orderly fashion freeing system resources used by the terminating thread.

Signal Handling – Unix systems use signals to keep track of events which must follow the same path of execution as follows

Create • An event occures. • Signal is created.

Deliver •Deliver new signal to a Process.

Perform • Signal recieved. •Handle Signal.

An "illegal memory access" or "devide by 0" actions produce synchronous signals sent to the causing operation's process. Asynchronous signals are those recieved as the result of an external event such as a keyboard commands like terminating a process, which are typically sent to another process.

Thread Pools – Eventhough creation of threads is more conservative than creating processes, unlimited threads can use up all the resources of a system. One way to keep this problem in check is the user of thread pools. The idea is to have a bunch of threads made upon the start of a process and hold them in a "pool", where they await task assignment. Once a request is recieved, it is passed on to an available thread in the pool. Upon

	completion of the task, the thread then returns to the pool awaiting its next task. If the pool is empty, the system holds the requests until an available thread returned to the pool. This method limits the number of threads in a system to a managable size, most beneficial when the system does not posses enough resources to handle a high number of threads. In return, the performance of the system increases as thread creation is ofthen slower than reuse of an existing one. Thread-Specific Data – The sharing of resources of the parent process does benefit multithreading programs, but in cases where a thread may need to hold it's on copy of some data, called thread-specified data, it could be a downfall as well. The 3 main thread libraries discussed in this do provide support for such thread-specific handling which are often used as unique identifiers in transaction processing systems.
14	**possible to have concurrency but not parallelism? Explain. (April 2016)** Yes concurrency is possible but not parallelism, parallelism simply means doing many tasks simultaneously, on the other hand concurrency is the ability of the kernel to perform many tasks by constantly switching among many processes. In order to achieve parallelism it is important that system should have many cores only then parallelism can be achieved efficiently, and there is lot of hit on performance and lot of overhead is incurred if parallelism is tried on a single core machine. For example, earlier system had only one core and CPU schedulers would give an illusion of parallelism by constantly switching between processes allowing each process to make progress. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). That is, parallelism always implies concurrency. Also, are multi-threaded programs running on multi-cores CPU but where the different threads are doing totally different computation considered to

	be using "parallelism"? No,The essence of parallelism is that a large problem is divided into smaller ones so that the smaller pieces can be solved concurrently. The pieces are mutually independent (to some degree at least), but they're still part of the larger problem, which is now being solved in parallel. The essence of concurrency is that a number of threads (or processes, or computers) are doing something simultaneously, possibly (but not necessarily) interacting in some ways. Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. **Consider a system consisting of four resources of same type that are shared by 3 processes, each of which needs atmost two resources. Show that the system is deadlock free. (April 2016)** Yes, this system is deadlock-free. Proof by contradiction. Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.
15	**Describe the actions taken by a kernel to context switch between processes. (April 2016)** In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

In the diagram below we have Process P0 executing initially. On an interrupt /system call the control is transferred to OS after the save state (current state of P0) is saved in PCB0. Now P0 is made idle. The OS now loads the state of process P1 from PCB1, thus P1 starts executing. Again on an interrupt, save state of P1 saved in PCB1 and reload state of P0 from PCB0 is loaded to resume the execution of P0

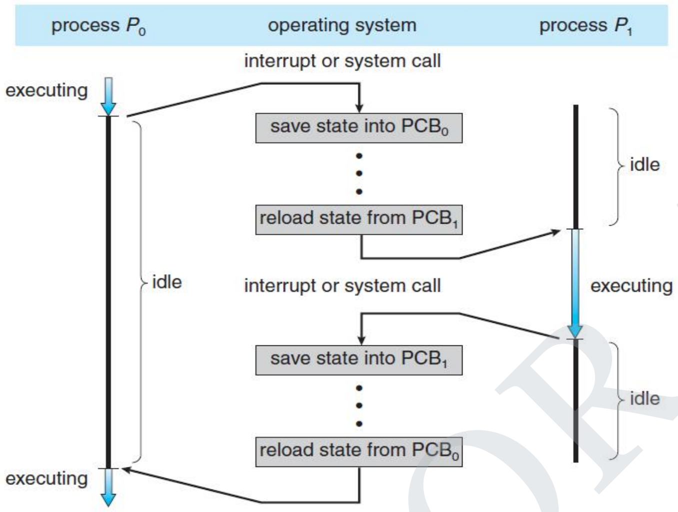


Figure 3.4 Diagram showing CPU switch from process to process.

Provide two programming examples in which multithreading does not provide better performance than a single threaded solution. (April 2016)

Programs with heavy I/O, hardly any CPU usage. Bottleneck will always be at I/O, threading will only add more overhead. – Shell programs. Have to track and keep update current state. Synchronization overhead. – Mainly sequential programs that modify small chunks of data. Synchronization overhead will outweigh threading benefits.

Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working

	directory.
16	**Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes (Nov 2016)** **Ordinary pipes:** Ordinary pipes require parent-child relationship between communicating processes. Ordinary pipes are unidirectional (producer-consumer style). **Named pipes:** No parent-child relationship is necessary between the communicating processes. Named pipes is bidirectional. An example of a situation in which ordinary pipes are more suitable than named pipes is when communication is only required between two processes or communication is only required in one direction. And the pipe is no longer needed after this communication is completed. An example of a situation in which named pipes are more suitable is when multiple processes need access to the pipes and bidirectional communication. They continue to exist after communicating processes have finished Simple communication works well with ordinary pipes. For example, assume we have a process that counts characters in a file. An ordinary pipe can be used where the producer writes the file to the pipe and the consumer reads the files and counts the number of characters in the file. Next, for an example where named pipes are more suitable, consider the situation where several processes may write messages to a log. When processes wish to write a message to the log, they write it to the named pipe. A server reads the messages from the named pipe and writes them to the log file.
17	**Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems. (Nov 2016)(NOV/DEC 2018)** Depends on how interrupts are implemented, but regardless of how, it is a poor choice of techniques. Case 1 -- interrupts are disabled for ONE processor only -- result is that

threads running on other processors could ignore the synchronization primitive and access the shared data

Case 2 -- interrupts are disabled for ALL processors -- this means task dispatching, handling I/O completion, etc. is also disabled on ALL processors, so threads running on all CPUs can grind to a halt

Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

What are the different thread libraries used? Explain any one with example (Nov 2016)

Threads libraries

The interface to multithreading support is through a subroutine library, libpthread for POSIX threads, and libthread for Solaris threads. They both contain code for:

creating and destroying threads

passing messages and data between threads

scheduling thread execution

saving and restoring thread contexts

The POSIX Threads Library:libpthread, <pthread.h>

Creating a (Default) Thread

Use the function pthread_create() to add a new thread of control to the current process.

Wait for Thread Termination

Use the pthread_join function to wait for a thread to terminate

Detaching a Thread

The function pthread_detach() is an alternative to pthread_join() to reclaim storage for a thread that is created with a detachstate attribute set to PTHREAD_CREATE_JOINABLE.

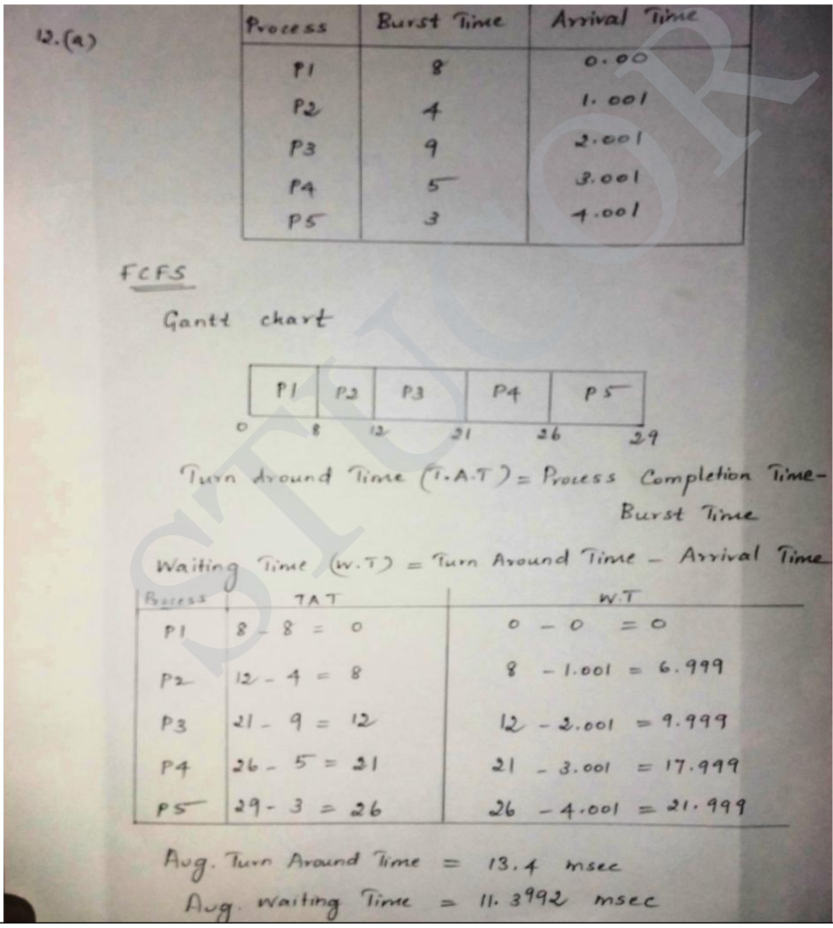
Create a Key for Thread-Specific Data

int pthread_key_create(pthread_key_t *key, void (*destructor) (void *));

	Delete the Thread-Specific Data Key The function pthread_keydelete() is used to destroy an existing thread-specific data key.
18	**Consider the following set of processes with the length of the CPU burst time in given ms:**

Process	Burst Time	Arrival Time
P1	8	0.00
P2	4	1.001
P3	9	2.001
P4	5	3.001
P5	3	4.001

Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF,Priority and RR (quantum=2) scheduling. Also calculate waiting time and turnaround time for each scheduling algorithms. (April 2017)



12.(a)

Process	Burst Time	Arrival Time
P1	8	0.00
P2	4	1.001
P3	9	2.001
P4	5	3.001
P5	3	4.001

FCFS

Gantt chart

P1	P2	P3	P4	P5

0 8 12 21 26 29

Turn Around Time (T.A.T) = Process Completion Time − Burst Time

Waiting Time (W.T) = Turn Around Time − Arrival Time

Process	TAT	W.T
P1	8 − 8 = 0	0 − 0 = 0
P2	12 − 4 = 8	8 − 1.001 = 6.999
P3	21 − 9 = 12	12 − 2.001 = 9.999
P4	26 − 5 = 21	21 − 3.001 = 17.999
P5	29 − 3 = 26	26 − 4.001 = 21.999

Avg. Turn Around Time = 13.4 msec

Avg. Waiting Time = 11.3992 msec

SJF

Gantt Chart

(6.999)	✓			✓	
P1	P2	P5	P4	P1	P3
0.00 – 1.001	1.001 – 5.001	5.001 – 8.001	8.001 – 13.001	13.001 – 20	20 – 29

Process	T.A.T	W.T
P1	20 – 8 = 12	12 – 0.00 = 12
P2	5.001 – 4 = 1.001	1.001 – 1.001 = 0
P3	29 – 9 = 20	20 – 2.001 = 17.999
P4	13.001 – 5 = 8.001	8.001 – 3.001 = 5
P5	8.001 – 3 = 5.001	5.001 – 4.001 = 1

Avg. Turn Around Time = 9.2006 msec

Avg. Waiting Time = 7.1998 msec

Round Robin

quantum = 2

Gantt Chart

(6)	(2)	(7)	(3)	(1)	(4)	✓	(5)	(1)	✓	(2)	(3)	✓	✓	(1)	✓
P1	P2	P3	P4	P5	P1	P2	P3	P4	P5	P1	P3	P4	P1	P2	P3
0.00–2	2–4	4–6	6–8	8–10	10–12	12–14	14–16	16–18	18–19	19–21	21–23	23–24	24–26	26–28	28–29

Process	Turn Around Time	Waiting Time
P1	26 – 8 = 18	18 – 0 = 18
P2	14 – 4 = 10	10 – 1.001 = 8.999
P3	29 – 9 = 20	20 – 2.001 = 17.999
P4	24 – 5 = 19	19 – 3.001 = 15.999
P5	19 – 3 = 16	16 – 4.001 = 11.999

Avg. Turn Around Time = 16.6 msec

Avg. waiting Time = 14.5992 msec

19 **What is a race condition? Explain how a critical section avoids this condition. What are the properties which a data item should possess to implement a critical section? Describe a solution to the Dining philosopher problem so that no races arise. (April 2017)**

A race condition is a special condition that may occur inside a critical section. A critical section is a section of code that is executed by multiple threads and where the sequence of execution for the threads makes a difference in the result of the concurrent execution of the critical section.

Critical Sections

Running more than one thread inside the same application does not by itself cause problems. The problems arise when multiple threads access the same resources. For instance the same memory (variables, arrays, or objects), systems (databases, web services etc.) or files.

In fact, problems only arise if one or more of the threads write to these resources. It is safe to let multiple threads read the same resources, as long as the resources do not change.

Race Conditions in Critical Sections

When multiple threads execute this critical section, race conditions occur.

More formally, the situation where two threads compete for the same resource, where the sequence in which the resource is accessed is significant, is called race conditions. A code section that leads to race conditions is called a critical section.

Preventing Race Conditions

To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction. That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.

Race conditions can be avoided by proper thread synchronization in critical sections. Thread synchronization can be achieved using a synchronized block of Java code. Thread synchronization can also be achieved using other synchronization constructs like locks or atomic variables.

Three properties to implement critical section .

1.Mutual Exclusion

a.Only one process can be in its critical section.

2.Progress

a.Only processes not in their remainder section can decide which will enter its critical section.

b.The selection cannot be postponed indefinitely.

3.Bounded Waiting

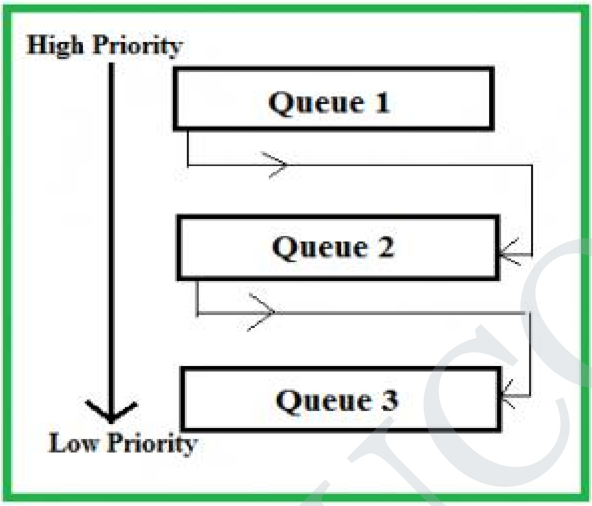
	a.A waiting process only waits for a bounded number of processes to enter their critical sections.
	Notation
	Processes Pi and Pj, where j=1-i;
	Assumption
	Every basic machine-language instruction is atomic.
	Solution to Dining Philosophers Problem
	monitor DiningPhilosophers
	{
	enum { THINKING; HUNGRY, 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:
	DiningPhilosophers.pickup(i);
	EAT
	DiningPhilosophers.putdown(i);

20	**Describe the difference among short-term, medium-term and long-term scheduling with suitable example.(April 2018)** **Long – Term** A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming.It is also called a **job scheduler** **Medium –Term** Medium-term planning applies more permanent solutions to short-term problems. If training courses for employees solved problems in the short term, companies schedule training programs for the medium term. If there are quality issues, the medium-term response is to revise and strengthen the company's quality control program. Where a short-term response to equipment failure is to repair the machine, a medium-term solution is to arrange for a service contract. **Short –Term** Short-term (CPU scheduler): selects a process from those that are in memory and ready to execute, and allocates the CPU to it. Medium-term (memory manager): selects processes from the ready or blocked queue and removes them from memory, then reinstates them later to continue running. Long-term (job scheduler): determines which jobs are brought into the system for processing
21	**Explain the difference in the degree to which the following scheduling algorithms discriminate in favour of short processes(April 2018)** RR Multilevel feedback queues. RR Often used for timesharing

– Ready queue is treated as a circular queue (FIFO)
– Each process is given a time slice called a quantum
– It is run for the quantum or until it blocks
– RR allocates the CPU uniformly (fairly) across all participants. If average queue length is n, each participant gets 1/n
– As the time quantum grows, RR becomes FCFS
– Smaller quanta are generally desirable, because they improve response time
• Problem:
– Context switch overhead of frequent context switch
Multilevel feedback queues.

Multilevel Feedback Queue Scheduling (MLFQ) keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.Now, look at the diagram and explanation below to understand it properly.



Now let us suppose that queue 1 and 2 follow round robin with time quantum 4 and 8 respectively and queue 3 follow FCFS.One implementation of MFQS is given below –

When a process starts executing then it first enters queue 1.

In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.

If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2.

Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit.In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.

In the last queue, processes are scheduled in FCFS manner.

	A process in lower priority queue can only execute only when higher priority queues are empty. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.
22	***Consider a system consisting of 'm' resources of the same type being shared by 'n' processes. Resource can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:*** i) ***The maximum need of each process is between 1 and m resources.*** ii) The sum of all maximum needs is less than m+n.(**April 2018**) Suppose N = Sum of all Needi, A = Sum of all Allocationi, M = Sum of all Maxi. Use contradiction to prove. Assume this system is not deadlock free. If there exists a deadlock state, then A = m because there's only one kind of resource and resources can be requested and released only one at a time. From condition b, N + A = M < m + n. So we get N + m < m + n. So we get N < n. It shows that at least one process i that Needi = 0. From condition a, Pi can release at least 1 resource. So there are n-1 processes sharing m resources now, condition a and b still hold. Go on the argument, no process will wait permenently, so there's no deadlock.

UNIT III STORAGE MANAGEMENT

Main Memory-Contiguous Memory Allocation, Segmentation, Paging, 32 and 64 bit architecture Examples; Virtual Memory- Demand Paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples.

PART-A

1	***Define logical address and physical address. (April 2016)*** An address generated by the CPU is referred as logical address. An address seen by the memory unit that is the one loaded into the memory address register of the memory is commonly referred to as physical address.
2	***What is logical address space and physical address space?*** The set of all logical addresses generated by a program is called a logical address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.
3	***What is the main function of the memory-management unit?***

	The runtime mapping from virtual to physical addresses is done by a hardware device called a memory management unit (MMU).
4	**What are overlays?** To enable a process to be larger than the amount of memory allocated to it, overlays are used. The idea of overlays is to keep in memory only those instructions and data that are needed at a given time. When other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed.
5	**Define swapping.** A process needs to be in memory to be executed. However a process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. This process is called swapping.
6	**What are the common strategies to select a free hole from a set of available holes?** The most common strategies are First fit, Best fit, Worst fit
7	**What do you mean by best fit?** Best fit allocates the smallest hole that is big enough. The entire list has to be searched, unless it is sorted by size. This strategy produces the smallest leftover hole.
8	**What do you mean by first fit?** First fit allocates the first hole that is big enough. Searching can either start at the beginning of the set of holes or where the previous first-fit search ended. Searching can be stopped as soon as a free hole that is big enough is found.
9	**Define Paging** Paging is a scheme that allows the logical address space of a process to be non-contiguous. Paging avoids the considerable problem of fitting the varying sized memory chunks onto the backing store.
10	**What do you mean by frames and pages?** Physical memory is broken into fixed size blocks called frames. Logical address is also broken into blocks of same size called pages.

11	**Define Page table** The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The page number is used as an index into the page table.
12	**Define Inverted Page table?** An inverted page table has one entry for each real page of memory. Each entry consists of the virtual address of the page stored in that real memory.
13	**Define Segmentation?** Segmentation is a memory management scheme that supports the user view of memory. A logical address space is a collection of segments. Each segment has a name and length.
14	**Define External Fragmentation?(Apr/May 2018)** External fragmentation is occurs when all blocks of free memory are too small to accommodate a segment. Segmentation may cause external fragmentation
15	**What is virtual memory?** Virtual memory is a technique that allows the execution of processes that may not be completely in memory. It is the separation of user logical memory from physical memory. This separation provides an extremely large virtual memory, when only a smaller physical memory is available.
16	**What is Demand paging? (Nov 2015)** Virtual memory is commonly implemented by demand paging. In demand paging, the pager brings only those necessary pages into memory instead of swapping in a whole process. Thus it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.
17	**Define lazy swapper.** Rather than swapping the entire process into main memory, a lazy swapper is used. A lazy swapper never swaps a page into memory unless that page

	will be needed.
18	**What is a pure demand paging?** When starting execution of a process with no pages in memory, the operating system sets the instruction pointer to the first instruction of the process, which is on a non-memory resident page, the process immediately faults for the page. After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This schema is pure demand paging.
19	**Define effective access time.** Let p be the probability of a page fault ($0 \le p \le 1$). The value of p is expected to be close to 0; that is, there will be only a few page faults. Effective access time = (1-p) * ma + p * page fault time where ma : memory-access time
20	**Define secondary memory** This memory holds those pages that are not present in main memory. The secondary memory is usually a high speed disk. It is known as the swap device, and the section of the disk used for this purpose is known as swap space.
21	**What is the basic approach of page replacement?** If no frame is free is available, find one that is not currently being used and free it. A frame can be freed by writing its contents to swap space, and changing the page table to indicate that the page is no longer in memory. Now the freed frame can be used to hold the page for which the process faulted.
22	**What is the various page replacement algorithms used for page replacement?** FIFO page replacement, Optimal page replacement, LRU page replacement, LRU approximation page replacement, Counting based page replacement, Page buffering algorithm.
23	**What are the major problems to implement demand paging?** The two major problems to implement demand paging is developing –

	Frame allocation algorithm, Page replacement algorithm
24	**What is the resident set and working set of the process? (Nov 2014)** Resident set is that portion of the process image that is actually in real-memory at a particular instant. Working set is that subset of resident set that is actually needed for execution.
25	**List the steps needed to perform page replacement? (Nov 2014) (Nov 2015)** The steps required to perform page replacement are: Find out which page is to be removed from the memory Perform an operation of page-out. Perform an operation page-in.
26	**What do you mean by "Thrashing"? State the side effects of thrashing in an operating system.(April 2015) (April 2016)(Nov/dec '18)** Thrashing – A high paging activity. Consider a process not having enough frames. This raises to page fault and an active page is selected as victim. This again produces quick Page Fault again and again. It can be avoided using working set strategy . This frequently leads to high, runaway CPU utilization that can grind the system to a halt.
27	**Mention the significance of LDT and GDT in segmentation. (April 2015)(Nov/Dec '18)** A table stores the information about all such segments and is called Global Descriptor Table (GDT). A GDT entry is called Global Descriptor. Segment register contains a selector that selects a descriptor from the descriptor table. The descriptor contains information about the segment, e.g., it's base address, length and access rights. There is also an LDT or Local Descriptor Table. The LDT is supposed to contain memory segments which are private to a specific program, while the GDT is supposed to contain global segments.
28	**What is the purpose of paging the page table? (Nov 2016)** The page number is used as an index into a page table. The page table

	contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
29	**Why are the page sizes always power of 2? (Nov 2016)** The paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.
30	**What is the difference between a user-level instruction and a privileged instruction? Which of the following instructions should be privileged and only allowed to execute in kernel mode? (April 2017)** **oad a value from a memory address to a general-purpose register-Privileged** **et a new value in the program counter (PC) register- Privileged** **urn off interrupts- Privileged** Machines have two kinds of instructions 1. "normal" instructions, e.g., add, sub, etc. 2. "privileged" instructions, e.g., initiate I/O switch state vectors or contexts load/save from protected memory
31	**Will optimal page replacement algorithm suffer from Belady's anomaly? Justify your answer. (April 2017)** Optimal page replacement algorithm does not suffer from Belady's anomaly.
32	**What are the counting based page replacement algorithm?(Apr/May 2018)** Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers; that is, it is an integer sorting algorithm. It operates by counting the number of objects that have each distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence.
	PART-B
1	**Explain about Memory management techniques and contiguous memory allocation. (Nov 2015)**

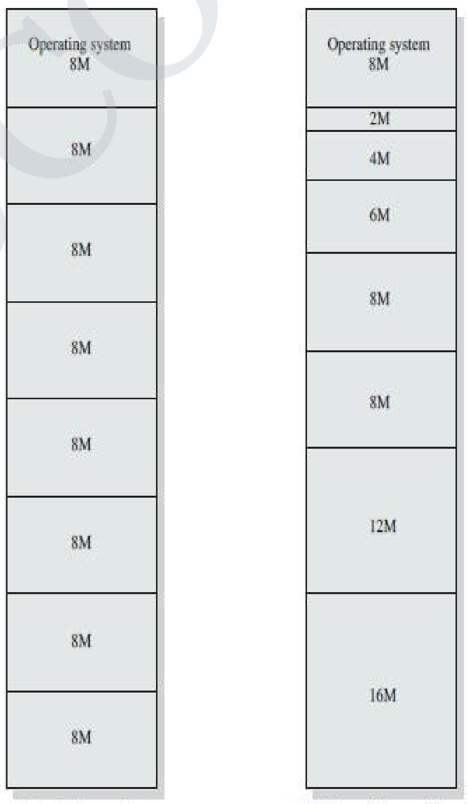
Main memory usually into two partitions:

Resident operating system, usually held in low memory with interrupt vector

User processes then held in high memory

One of the simplest methods for allocating memory is to divide memory into several fixed-sized **partitions**. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this **multiple partition method**, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process

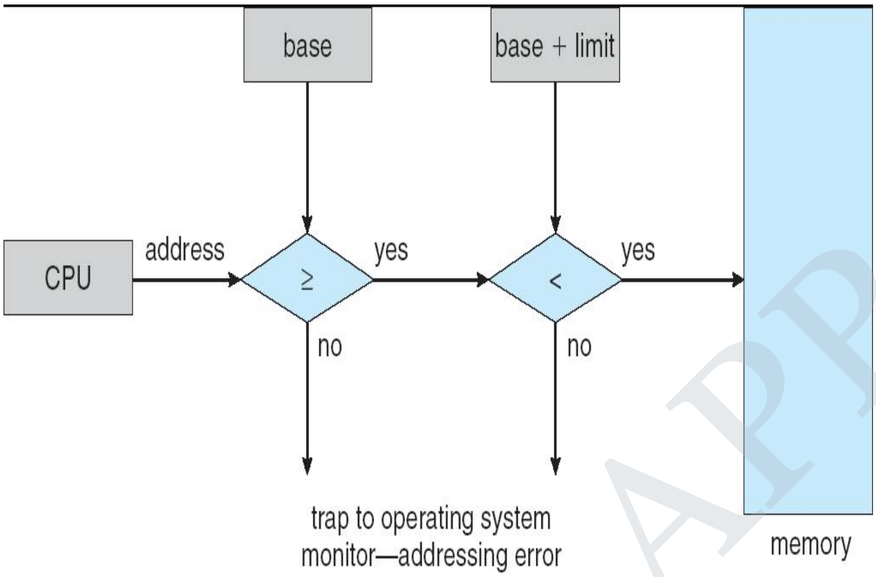
In the **variable-partition** scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a **hole**. Eventually, as you will see, memory contains a set of holes of various sizes



(a) Equal-size partitions (b) Unequal-size partitions

Memory protection

Relocation registers used to protect user processes from each other, and from changing operating-system code and data. Base register contains value of smallest physical address. Limit register contains range of logical addresses – each logical address must be less than the limit register. MMU maps logical address dynamically



CPU scheduler - selects a process for execution

Dispatcher - loads relocation & limit registers with correct values as part of context switch

Relocation scheme – efficient way to allow OS size to change dynamically – transient OS code eg. Device driver or other OS services – not commonly used, do not want to keep the code & data in memory

Initially, all memory is available for user processes and is considered one large block of available memory, a **hole.**

When a process arrives, it is allocated memory from a hole large enough to accommodate it

Operating system maintains information about:

a) allocated partitions b) free partitions (hole)

When Process terminates, it releases its block of memory – placed back in the set of holes

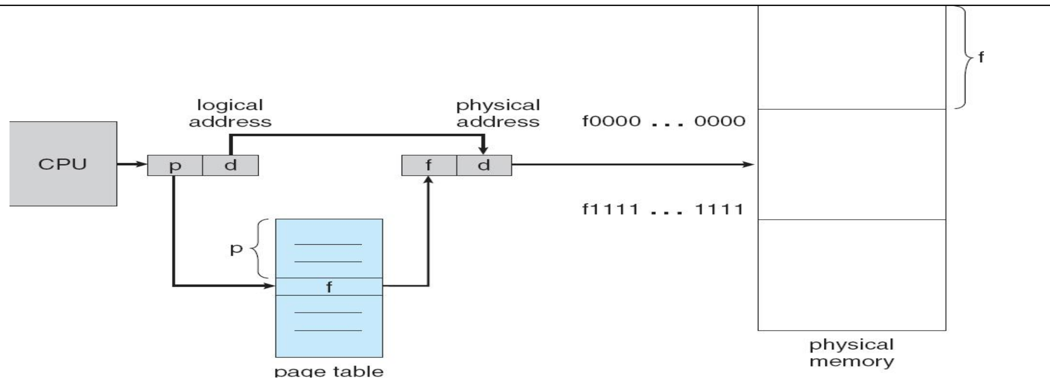
If new hole is adjacent to other holes, adjacent holes are merged to form one larger hole

Dynamic storage allocation problem

The **first-fit, best-fit,**and**worst-fit** strategies are the ones most commonly used to select a free holefrom the set of available holes.

First fit. Allocate the *first* hole that is big enough. Searching can start

	eitherat the beginning of the set of holes or where the previous first-fit searchended. We can stop searching as soon as we find a free hole that is largeenough. Best fit. Allocate the *smallest* hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole. Worst fit. Allocate the *largest* hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach **Fragmentation** **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used Reduce external fragmentation by compaction Shuffle memory contents to place all free memory together in one large block Compaction is possible *only* if relocation is dynamic, and is done at execution time Simple compaction – move all processes toward one end of memory; all holes move in other direction → expensive
2	**Give the basic concepts about paging (April 2015) (Nov 2015)** Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8,192 bytes) Divide logical memory into blocks of same size called pages Page size (like frame size) is defined by the hardware Size of the page in powers of 2 makes the translation of logical address into page number and page offset particularly easy . The hardware support for paging is illustrated in Figure 8.7.



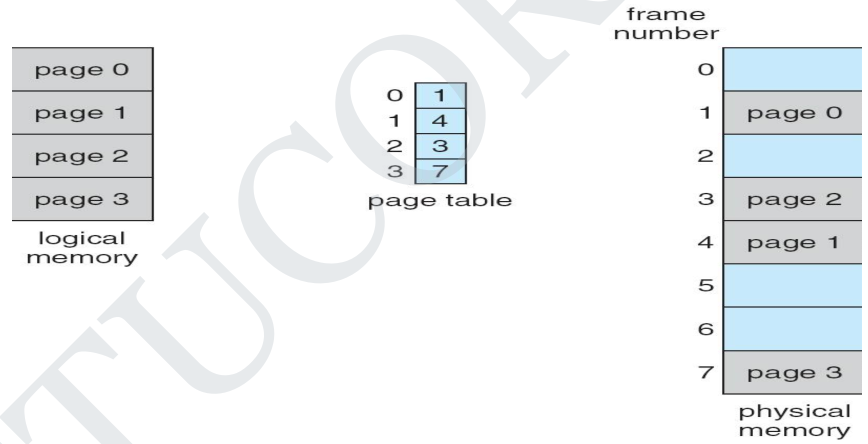
Every address generated by CPU is divided into:

Page number (*p*) – used as an index into a *pagetable* which contains base address of each page in physical memory

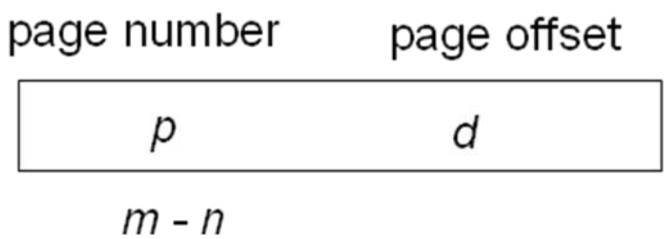
Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

For given logical address space 2^m and page size2^n.

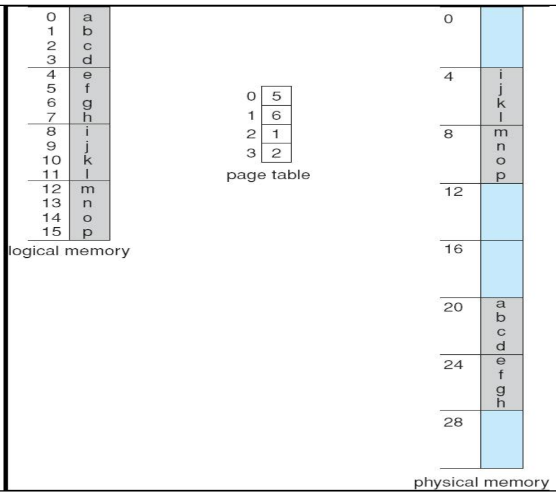
This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The paging model of memory is shown in Figure



If the size of logical address space is 2'''* and a page size is 2'' addressing units (bytes or words), then the high-order *m* – *n* bits of a logical address designate the page number, and the *n* low-order bits designate the page offset. Thus, the logical address is as follows:



Paging example:



3 **Explain in details about paging with TLB. (Nov 2015) (April 2017)**

Hardware Implementation of Page Table

Page table is implemented as a set of dedicated registers

dedicated registers are built with very high-speed logic to make the paging address translation efficient

CPU dispatcher reloads these registers as it reloads the other registers

Instructions to load or modify page table registers are privileged – only OS can change the memory map

Use of registers for page table – satisfactory if the page table is reasonably small (eg., 256 entries)

Page table is kept in main memory

Suitable if the size of the page table is large

Page-table base register (PTBR) points to the page table

Changing page table requires changing only this register – reducing context switch time

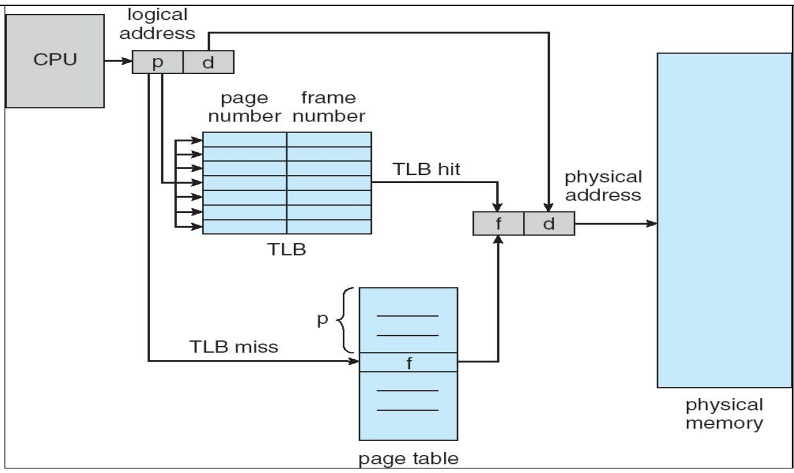
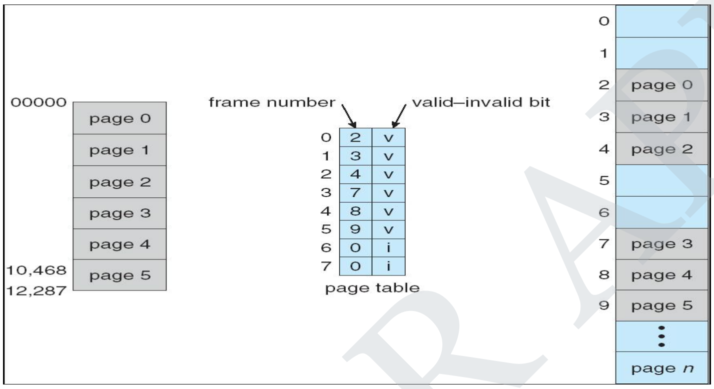
Page-table length register (PRLR) indicates size of the page table

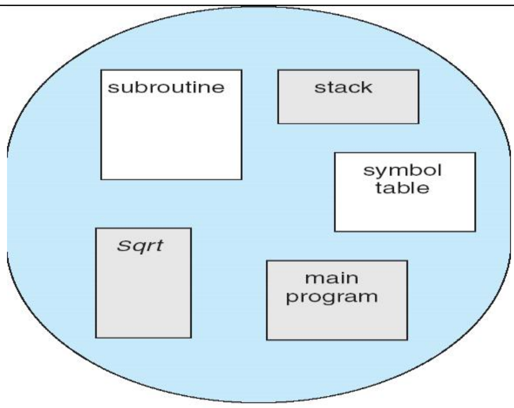
Problem:

In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The two memory access problem can be solved by the use of a **special fast-lookup hardware cache called associative memory or Translation Look-aside Buffers (TLBs)**

Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

	 **Protection**  Memory protection implemented by associating protection bit with each frame Valid-invalid bit attached to each entry in the page table: "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page "invalid" indicates that the page is not in the process' logical address space
4	**Explain the basic concepts of segmentation. (April 2016)** Let's first assume no paging in the system User generates logical addresses These addresses consist of a segment number and an offset into the segment Use segment number to index into a table Table contains the physical address of the start of the segment often called the base address Add the offset to the base and generate the physical address before doing this, check the offset against a limit the limit is the size of the segment **User's View of a Program**



logical address

Segmentation

Segmentation is a technique for breaking memory up into logical pieces

Each "piece" is a grouping of related information

data segments for each process

code segments for each process

data segments for the OS

etc.

Like paging, use virtual addresses and use disk to make memory look bigger than it really is

Segmentation can be implemented with or without paging

Segmentation is a memory-management scheme that supports this user view of memory.

A logical address space is a collection of segments. Each segment has a name and a length.

The addresses specify both the segment name and the offset within the segment.

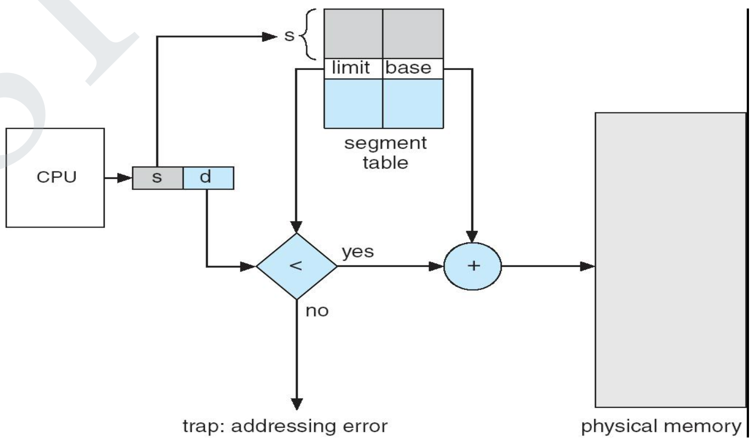
The user therefore specifies each address by two quantities: a segment name and an offset.

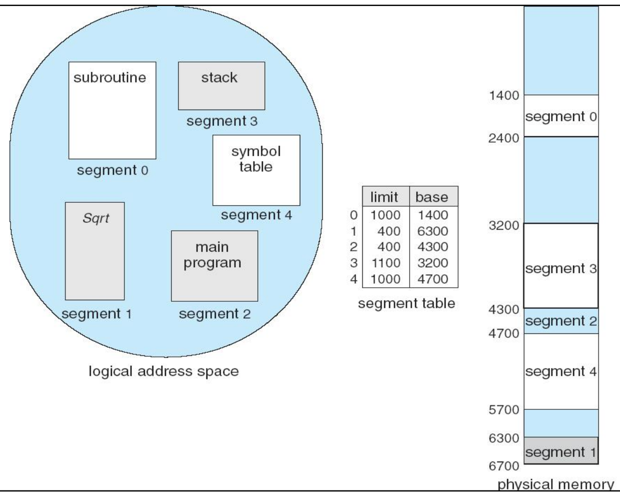
For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Tluis, a logical address consists of a *two tuple:*

<segment-number, offset >.

Hardware

Each entry in the segment table has a *segment base* and a *segment limit.* The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length

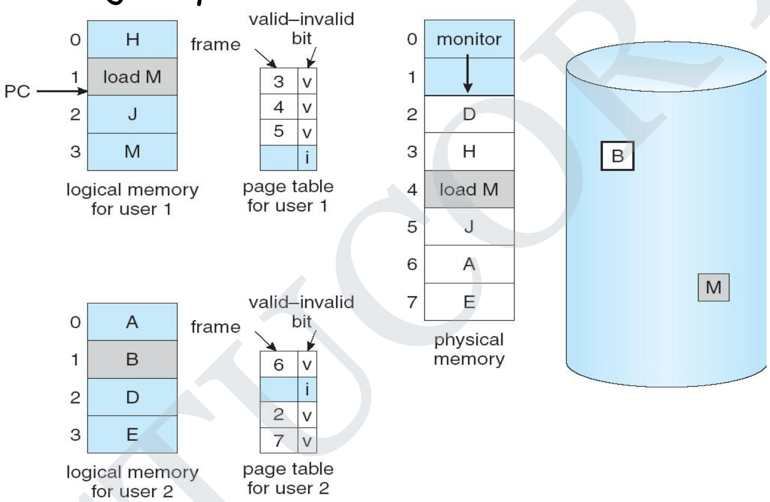
	of the segment. A logical address consists of two parts: a segment number, *s*, and an offset into that segment, *d*. The segment number is used as an index to the segment table. The offset *d* ofthe logical address must be between 0 and the segment limit. If it is not, we trapto the operating system (logical addressing attempt beyond, end of segment). When an offset is legal, it is added to the segment base to produce the addressin physical memory of the desired byte. **Segment-table base register (STBR)** points to the segment table's location in memory **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*<STLR** **Protection** With each entry in segment table associate: validation bit = 0 ⇒ illegal segment read/write/execute privileges Protection bits associated with segments; code sharing occurs at segment level Since segments vary in length, memory allocation is a dynamic storage-allocation problem **Segmentation Hardware**  **Example of Segmentation**



5 **What is the need for page Replacement? Write down the steps for page replacement.**

Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Need For Page Replacement



Basic Page Replacement

Routine to include page replacement:

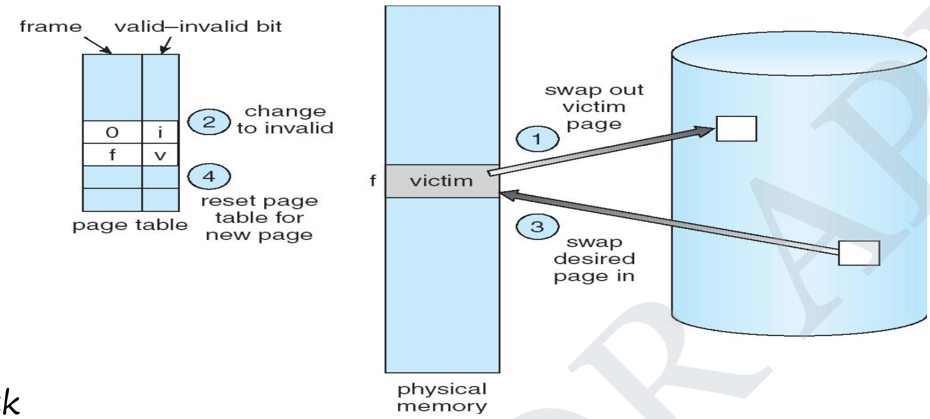
1. Find the location of the desired page on the disk.
2. Find a free frame:
    a. If there is a free frame, use it.
    b. If there is no free frame, use a page-replacement algorithm toselect a victim frame.
    c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and

frame tables.

4. Restart the user process.

If no frames are free, *two* page transfers (one out and one in) are required. We can reduce this overhead by using a **modify bit** (or **dirty bit**).

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. When we select a page for replacement, we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the



disk

Page Replacement Policies

Goal: want to minimize number of (major) page faults (situations where a page must be brought in from disk.)

Also: want to reduce their cost (ideally, evict those pages from their frames that are already on disk –save writeback time)

Possible scopes in which replacement is done:

Global replacement policies

Treat frames used by all processes equally

Local replacement polices

Pool frames according to user or process when considering replacement

Hybrids

Algorithms can be applied in these scopes

6 **Explain the various page replacement strategies. (Nov 2014)**

Replacement Algorithms

Optimal:

"know the future"

Obviously impractical, just a benchmark for comparison/analysis

FIFO –evict oldest page

LRU –evict least recently used page

Clock algorithm ("NRU")

In all our examples, the reference string is

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

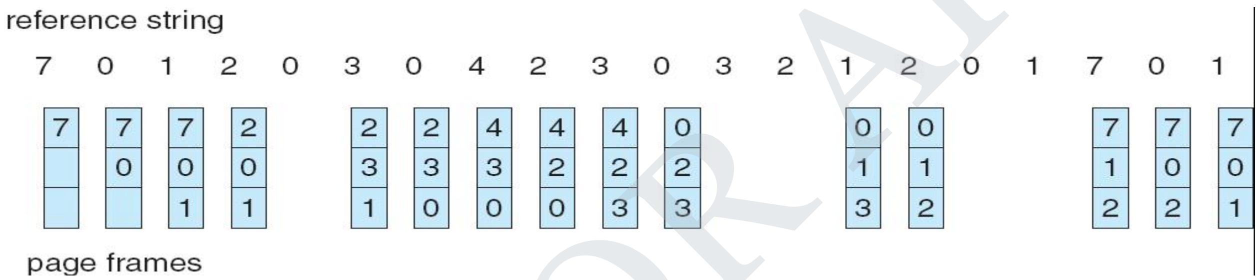
First-In-First-Out (FIFO) Algorithm

Oldest page is chosen for replacement

Associate time when the page was brought in to memory with each page

Not strictly necessary to record the time when a page is brought in

Create a FIFO queue to hold all pages in memory



Easy to understand and implement

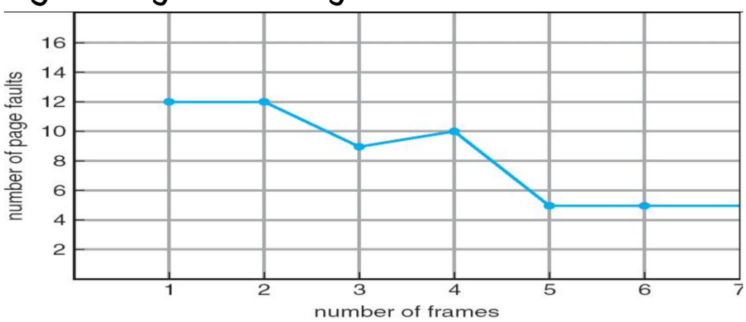
Performance is not always good

If we page out an active page to bring in a new one, a fault occurs almost immediately to retrieve the active page.

This bad replacement choice increases the page-fault rate and slows process execution

Belady's Anomaly: more frames Þ more page fault

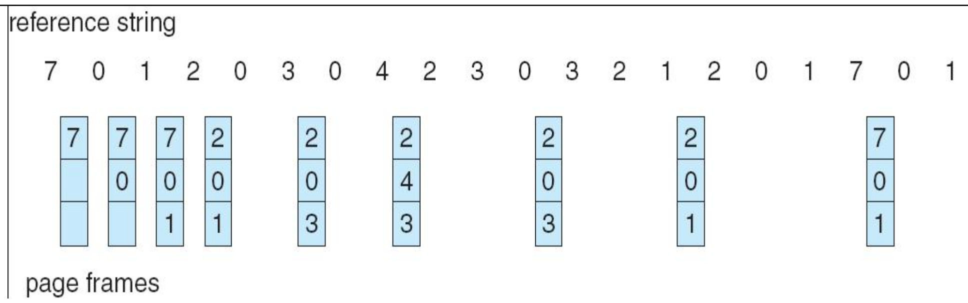
FIFO Illustrating Belady's Anomaly



Optimal Algorithm

Replace the page that will not be used for longest period of time

Has lowest page-fault rate of all algorithms



Guarantees the lowest possible page fault rate for fixed no. of frames

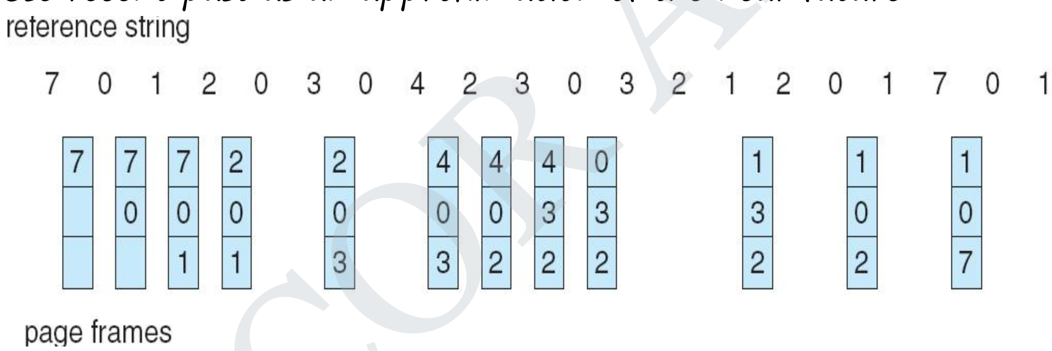
But, difficult to implement, because it requires future knowledge of the reference string

Used mainly for comparison studies

Least Recently Used (LRU) Algorithm

Replace the page that has not been used for the longest period of time

Use recent past as an approximation of the near future



Counter implementation

Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter

When a page needs to be changed, look at the counters to determine which are to change

Stack implementation – keep a stack of page numbers in a double link form:

Page referenced:

move it to the top

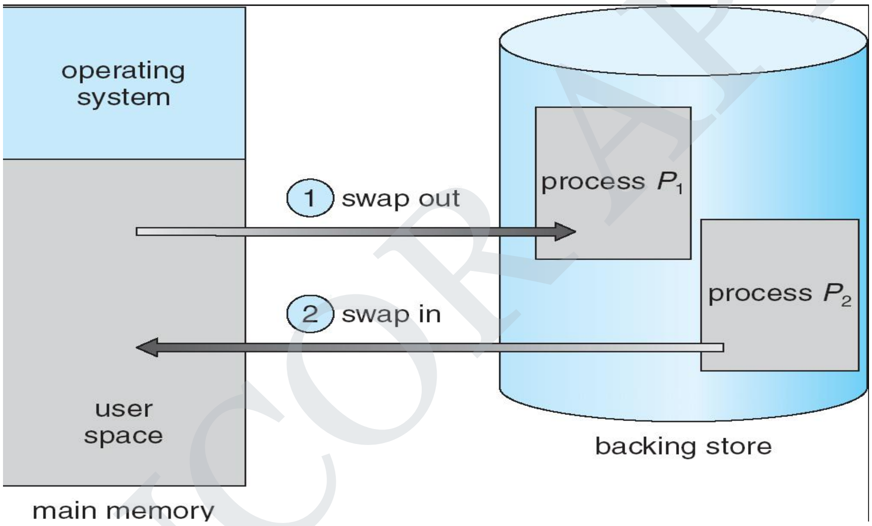
requires 6 pointers to be changed

No search for replacement

7 **Explain in detail about swapping and thrashing. (Nov 2015)**

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

Backing store – fast disk large enough to accommodate copies of all

	memory images for all users; must provide direct access to these memory images Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows) System maintains a ready queue of ready-to-run processes which have memory images on disk.  Context switching time in such swapping system is fairly high Size of user process = 1 MB Backing store – standard hard disk with transfer rate of 5 MB/sec Actual transfer of the 1 MB process to or from memory takes 1000 KB / 5000KB/sec = 1/5 sec or 200ms Assume no head seek, avg latency = 8ms Swap time = 208ms Total swap time (both swap-in and swap-out) = 416ms Swapping is constrained by other factors To swap a process, process must be completely idle, not waiting for I/O solution to the problem ➔ never swap a process with pending I/O Modification of swapping ➔ used in UNIX swapping is normally disabled but would start if many processes were

running & were using threshold amount of memory

swapping again be halted if the load on system were reduced

Thrashing(Nov 2015)

•Process(es) "frequently"reference page not in memory

–Spend more time waiting for I/O then getting work done

•Three different reasons

–process doesn't reuse memory, so caching doesn't work (past != future)

–process does reuse memory, but it does not "fit"

–individually, all processes fit and reuse memory, but too many for system.

access pattern

Cause of Thrashing

The operating system monitors CPU utilization. If CPU utilization is too low,we increase the degree of multiprogramming by introducing a new process to the system.

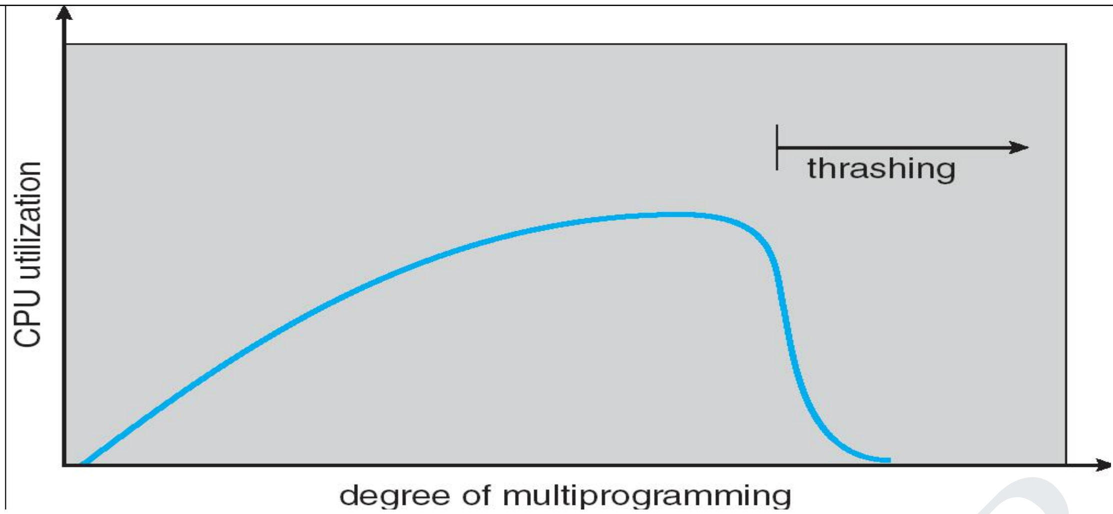
The CPU scheduler sees the decreasing CPU utilization and *increases* thedegree of multiprogramming as a result. The new process tries to get startedby taking frames from running processes, causing more page faults and a longerqueue for the paging device.

As a result, CPU utilization drops even further,and the CPU scheduler tries to increase the degree of multiprogramming evenmore.

Thrashing has occurred, and system throughput plunges. The pagefaultrate increases tremendously As a result, the effective memory-accesstime increases.

We can limit the effects of thrashing by using a **local replacement algorithm** (or **priority replacement algorithm).** With local replacement, if one processstarts thrashing, it cannot steal frames from another process and cause the latterto thrash as well.

To prevent thrashing, we must provide a process with as many frames as it needs. But how do we know how many frames it "needs'? There are several techniques. The working-set strategy starts by looking at how many frames a process is actually using. This approach defines the locality model of process execution.The locality model states that, as a process executes, it moves from locality to locality. A locality is a set of pages that are actively used together



If a process does not have "enough" pages, the page-fault rate is very high. This leads to:

–low CPU utilization

–operating system spends most of its time swapping to disk

•Thrashing is a process is busy swapping pages in and out

8 **Write in detail about Virtual memory.**

Virtual memory – separation of user logical memory from physical memory.

Only part of the program needs to be in memory for execution

Logical address space can therefore be much larger than physical address space

Allows address spaces to be shared by several processes

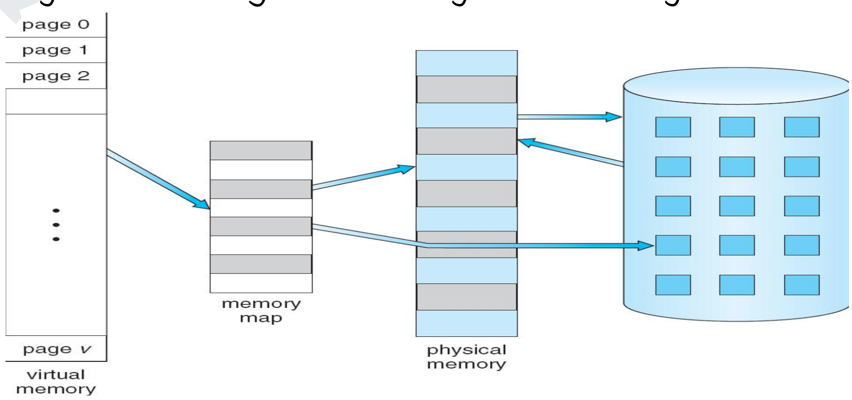
Allows for more efficient process creation

Virtual memory can be implemented via:

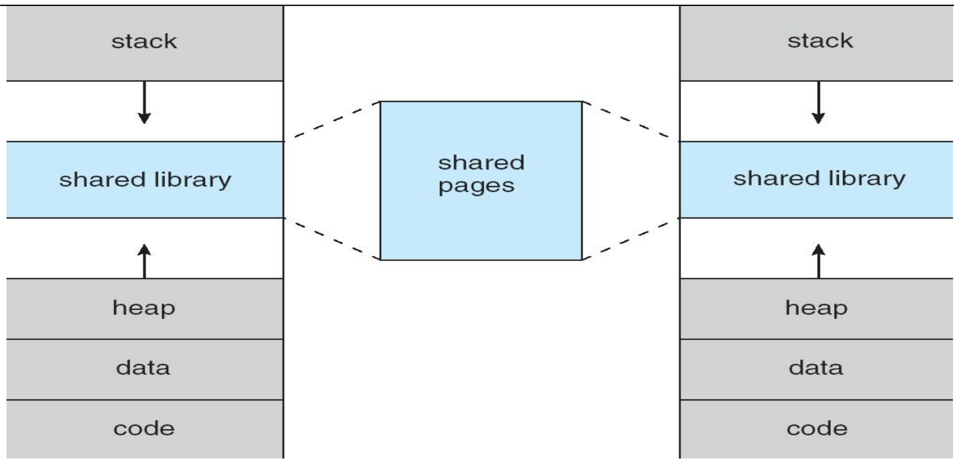
Demand paging

Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Shared Library Using Virtual Memory



Bring a page into memory only when it is needed

Less I/O needed

Less memory needed

Faster response

More users

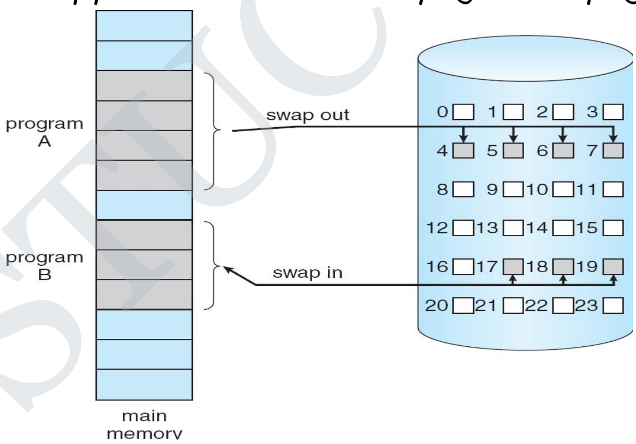
Page is needed ⇒ reference to it

invalid reference ⇒ abort

not-in-memory ⇒ bring to memory

Lazy swapper – never swaps a page into memory unless page will be needed

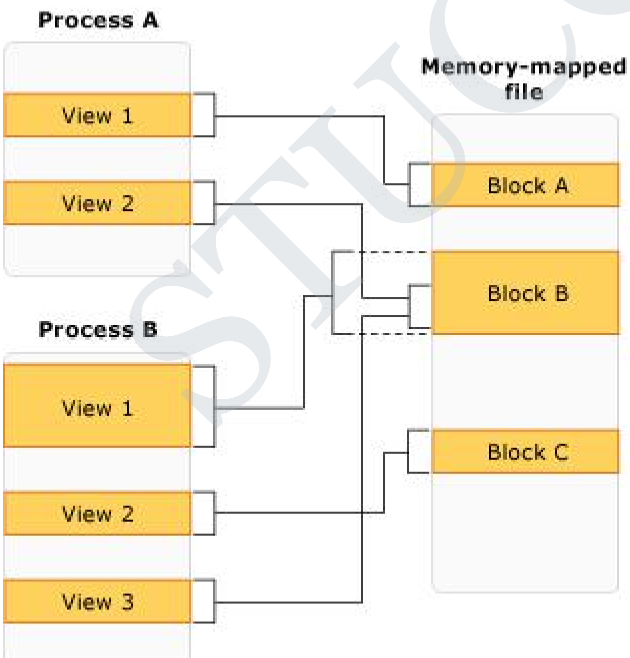
Swapper that deals with pages is a pager



With each page table entry a valid–invalid bit is associated (v⇒ in-memory, i⇒ not-in-memory).Initially valid–invalid bit is set to i on all entries.

During address translation, if valid–invalid bit in page table entry is I⇒ page fault

Write short notes on Memory mapped files. **(April 2015)**

	A **memory-mapped file** is a segment of virtual memory which has been assigned a direct byte-for-byte correlation with some portion of a file or file-like resource. This resource is typically a file that is physically present on-disk, but can also be a device, shared memory object, or other resource that the operating system can reference through a file descriptor. Once present, this correlation between the file and the memory space permits applications to treat the mapped portion as if it were primary memory. There are two types of memory-mapped files: **Persisted memory-mapped files** Persisted files are associated with a source file on a disk. The data is saved to the source file on the disk once the last process is finished. These memory-mapped files are suitable for working with extremely large source files. **Non-persisted memory-mapped files** Non-persisted files are not associated with a file on a disk. When the last process has finished working with the file, the data is lost. These files are suitable for creating shared memory for inter-process communications (IPC). 
9	**A system provides support for user-level and kernel-level threads. The mapping in this system is one to one (there is a corresponding kernel thread for each user thread). Does a multithreaded process consist of (a)**

a working set for the entire process or (b) a working set for each thread? Explain.

In multithread process, process control consists of a working set of for each thread. because, each user-level thread maps to a separate kernel thread. Therefore, creating a user thread implies creating a kernel thread (more overhead than a new user-level thread in the same user process). below figure shows the one to one mapping of Kerner supported user threads.

The slab – allocation algorithm uses a separate cache for each different object type. Assuming there is one cache per object type, explain why this scheme doesn't scale well with multiple CPUs. What could be done to address this scalability issue?

This had long been a problem with the slab allocator – poor scalability with multiple CPUs. The issue comes from having to lock the global cache when it is being accesses. This has the effect of serializing cache accesses on multiprocessor systems. Solaris has addressed this by introducing a per-CPU cache, rather than a single global cache.

Segment table		Page Tables						
		0	1	2	3	4		
		0	0x73	0x25	0x85	0x0f		0x17
0	0x3	1	0x2c	0x2d	0x31	0x3d		0x00
1	0x1	2	0x05	0x1e	0x01	0x5d		0x0d
2	0x0	3	0x17	0x5a	0x1f	0x1e		0x66
3	0x4	4	0x57	0x0f	0x09	0x6e		0x62
		5	0x1	0x7a	0x0a	0x2f		0x50

10 **Consider the following segmented paging memory system. There are 4**

	segments for the given process and a total of 5 page tables in the entire system. Each page table has a total of 8 entries. The physical memory requires 12 bits to address it; there are a total of 128 frames. (i) How many bytes are contained within the physical memory? (ii) How large is the virtual address? (iii) What is the physical address that correspond to virtual address 0x312? (iv) What is the physical address that correspond to virtual address 0x1E9? **(Nov 2014)** How many bytes are contained within the physical memory? 2^{12}= 4096 bytes How large is the virtual address? 2 + 3 + 5 = 10 bits What is the physical address that corresponds to virtual address 0x312? 0x2F2 What is the physical address that corresponds to virtual address 0x1E9? 0xD89
11	**Consider the following page reference string: 1 2 3 2 5 6 3 4 6 3 7 3 1 5 3 6 3 4 2 4 3 4 5 1 Indicate page faults and calculate total number of page faults and successful ratio for FIFO,optimal and LRU Algorithms. Assume there are four frames and initially all frames are empty. (Nov 2015)**

ANSWER:

No of frames=4

Page fault= When referred page is not in frame , page fault occurs

FIFO

First In frame is replaced first on event of a page fault

1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1		1	6		6			6	6	1	1		1		1	2		2		2	2
	2	2		2	2		4			4	4	4	5		5		5	5		3		3	3
		3		3	3		3			7	7	7	7		6		6	6		6		5	5
				5	5		5			5	3	3	3		3		4	4		4		4	1

Number of page faults=16

Optimal

In Optimal the pages that will be least used in future are replaced on event of a page fault

1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1		1	1		1			1			1				1	2					1
	2	2		2	6		6			6			6				4	4					4

		3		3	3		3			3			3				3	3					3
				5	5		4			7			5				5	5					5

Number of page faults= 11

<u>LRU</u>

In LRU the pages that was least used in past are replaced on event of a page fault

1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1		1	6		6			6		6	5		5		5	2				2	1
	2	2		2	2		4			4		1	1		1		4	4				4	1
		3		3	3		3			3		3	3		3		3	3				3	3
				5	5		5			7		7	7		6		6	6				5	5

Number of page faults= 14

Why are segmentation and paging sometimes combined into one scheme? Explain them in detail with example. (April 2016)

12 **Discuss situations in which the least frequently used (LFU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also discuss under what circumstances the opposite holds good. (April 2016)**

Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.

Under what circumstances do page fault occur? Describe the actions taken by the operating system, when a page fault occurs. (April 2016)

Page fault occurs, when a process tries to use a page that was not brought in its memory. Access to a page marked invalid becomes a page fault trap.

Procedure for handling page fault

Check an internal table to determine the reference is valid/invalid

If the reference is illegal, terminate the process

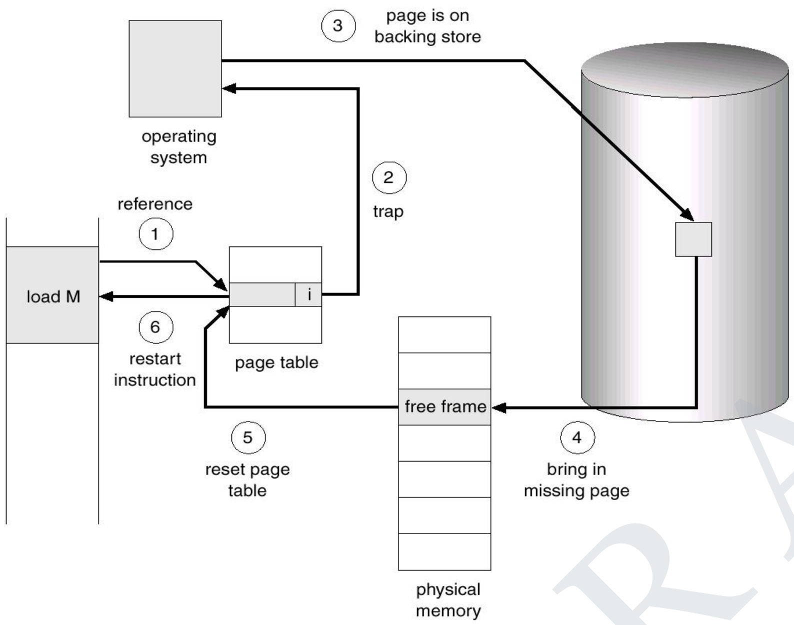
If it was legal, bring the page to memory

For that find a free frame

Schedule a disk op to read & copy the desired page into newly allocated frame

Modify internal table and update page table

Restart the instruction that was interrupted



.

A Page fault sequences

A page fault causes the following sequences to occur:

Trap to the operating system

Save the user register and process state

Determine that the interrupt was a page fault

Check that the page reference was legal and determine the location of the page on the disk

Issue a read from the disk to a free frame:

Wait in a queue for the this device until the read request is serviced

Wait for the device seek and or latency time

Begin the transfer of the page to a free frame

While waiting allocate the CPU to some other user (CPU scheduling; optional)

Interrupt from the disk (I/O completed)

	Save the registers and process state for the other user (if step 6 is executed) Determine that the interrupt was from the disk Correct the page table and other tables to show that the desired page is now in memory Wait for the CPU to be allocated to this process again Restore the user registers, process state and new page table , then resume the interrupted instruction
13	**What is the copy-on-write feature, and under what circumstances is its use beneficial? What hardware support is required to implement this feature? (Nov 2016)** Sometimes referred to as implicit sharing or shadowing, is a resource-management technique used in computer programming to efficiently implement a "duplicate" or "copy" operation on modifiable resources. If a resource is duplicated but not modified, it is not necessary to create a new resource; the resource can be shared between the copy and the original. Modifications must still create a copy, hence the technique: the copy operation is deferred to the first write. By sharing resources in this way, it is possible to significantly reduce the resource consumption of unmodified copies, while adding a small overhead to resource-modifying operations. When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

	Copy-on-write is a kernel feature that takes advantage of a system's paging hardware to avoid copying the contents of a frame when a page needs to be duplicated. When a page copy is requested, the OS manipulates the copying process's page table so that the entry for the copied page points to the page to be copied. The process then accesses the original page by generating references to the copied page. Only when one of the processes attempts to modify the page are the contents of the page actually copied. Once the copy takes place, the OS updates the page table of the process that wants to modify the page. Since COW is implemented using demand paging, paging hardware is required. In addition, the system must be able to intercept writes to a COW page (usually through a trap) before the write hits memory. Thus, we need some way of indicating whether a page is a COW page. This can be implemented using an additional bit in the page table entry, or by marking the page as unwritable if the page table supports a 'W' bit. **Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme? What modifications to the virtual memory system provide this functionality? (Nov 2016)** The program could have a large code segment or use large sized arrays as data. These portions of the program could be allocated to larger pages, thereby decreasing the memory overheads associated with a page table. The virtual memory system would then have to maintain multiple freelists of pages for the different sizes and also needs to have more complex code for address translation to take into account different page sizes.
14	**Explain the difference between internal and external fragmentation (Nov 2016)** Both internal fragmentation and external fragmentation are phenomena where memory is wasted. Internal fragmentation occurs in fixed size memory allocation while external fragmentation occurs in dynamic memory allocation. When an allocated partition is occupied by a program that is lesser than the partition, remaining space goes wasted causing internal fragmentation. When enough adjacent space cannot be found after loading and unloading of programs, due to the fact that free space is distributed

here and there, this causes external fragmentation. Fragmentation can occur in any memory device such as RAM, Hard disk and Flash drives.

COMPARISON	INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
Basic	It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
Occurrence	When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
Solution	The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging

Discuss the situation in which the most frequently used page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also discuss under what circumstances the opposite holds (Nov 2016)

	Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.
15	**Discuss the given Memory Management techniques with diagrams – Partition Allocation Methods (April 2017)**

Partition Allocation Methods (7)

Main memory usually has two partitions –

Low Memory – Operating system resides in this memory.

High Memory – User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Fragmentation

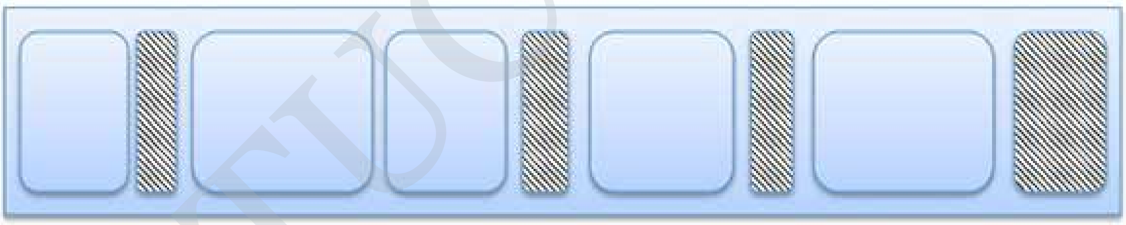
As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

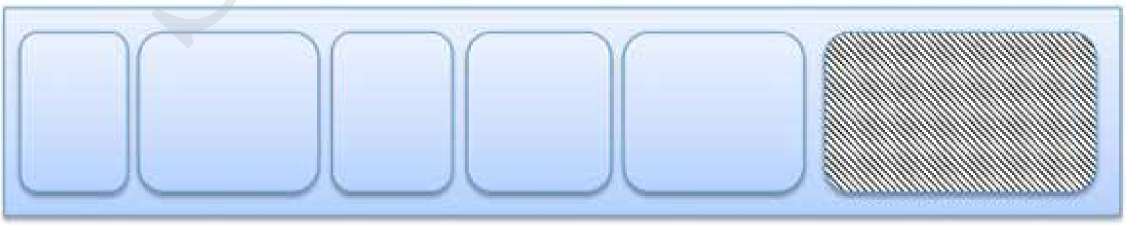
S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory blo k assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –

Fragmented memory before compaction



Memory after compaction



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the

	smallest partition but large enough for the process.
16	**about free space management on I/O buffering and blocking (April2017)** To keep track of free space, system maintains a free-space list **Bit vector bit map** (n blocks) Eg. Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free, and the rest of the blocks are allocated. Free-space bit map would be: 001111001111110001100000011100000 Main advantage:simplicity and efficiency in finding the first free block or n consecutive free blocks on the disk Eg. Intel family starting with 80386, Motorola family starting with 68020 have instructions that return the offset in a word of the first bit with the value 1. Apple Macintosh OS – uses the bit-vector method to allocate disk space To find first free block, Macintosh OS, checks sequentially each word in the bit map to see whether that value is not 0, since a 0-valued word has all 0-bits and represents a set of allocated blocks. The first non-0 word is scanned for the first 1-bit, which is the location of the first free-block Calculation of block number is: Block number = (number of bits per word) * (number of 0-value words) + offset of first 1 bit Bit vectors are inefficient unless the entire vector is kept in main memory. (written to disk occasionally for recovery needs) Example: block size = 212 bytes disk size = 230 bytes (1 gigabyte) n = 230/212 = 218 bits (or 32K bytes) Easy to get contiguous files

Linked list

Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory

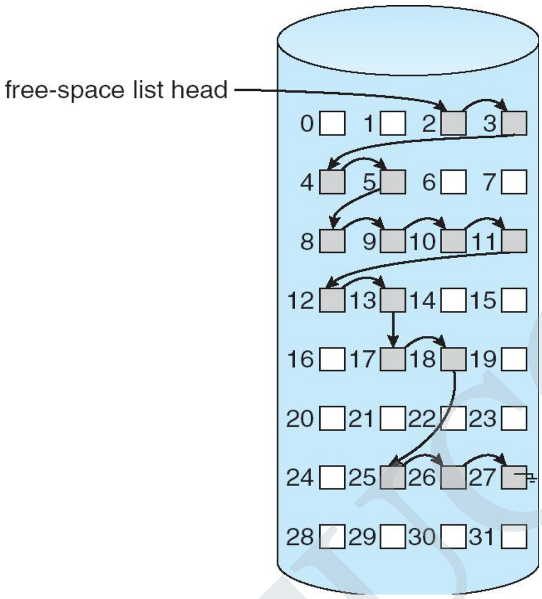
First block contains a pointer to the next free disk block and so on.

Not efficient to traverse the list, must read each block, which requires substantial I/O time

Cannot get contiguous space easily

No waste of space

Linked Free Space List on Disk



Grouping – store the addresses of n free blocks in the first free block, last block contains the addresses of another n free blocks

– addresses of a large number of free blocks can be found quickly

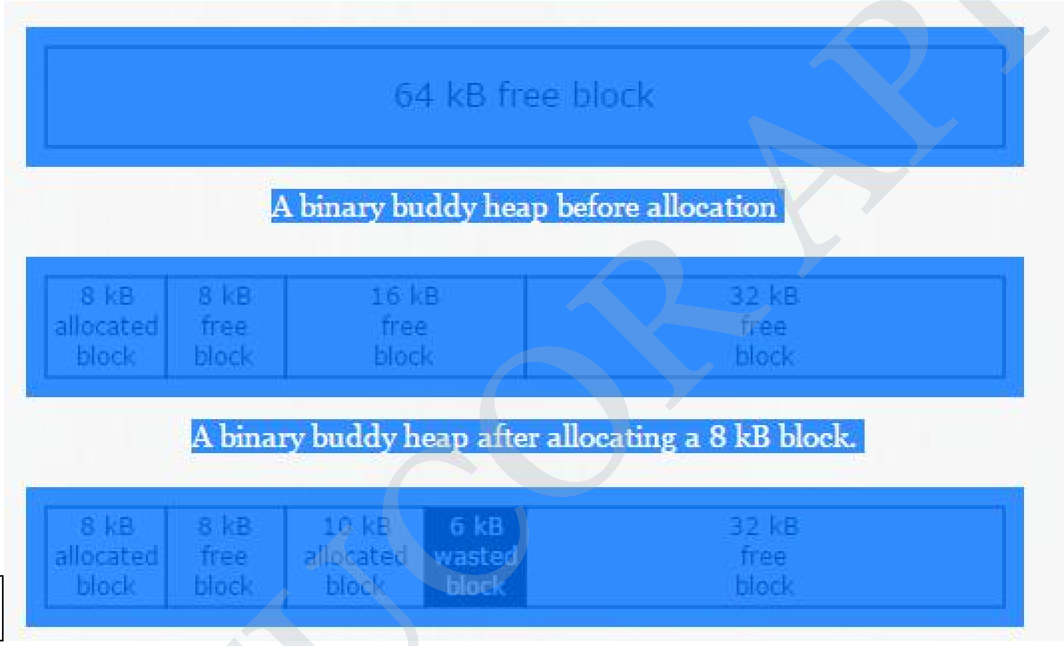
Counting– keep the address of the first free block and the no. of free contiguous blocks that follow the first block

Discuss the concept of buddy system allocation with neat sketch (April 2017)

In a buddy system, the allocator will only allocate blocks of certain sizes, and has many free lists, one for each permitted size. The permitted sizes are usually either powers of two, or form a Fibonacci sequence (see below for example), such that any block except the smallest can be divided into two smaller blocks of permitted sizes.

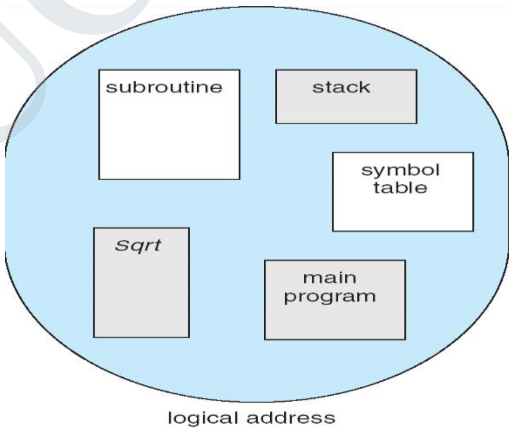
When the allocator receives a request for memory, it rounds the requested size up to a permitted size, and returns the first block from that size's free list. If the free list for that size is empty, the allocator splits a block from a larger size and returns one of the pieces, adding the other to the appropriate free list.

When blocks are recycled, there may be some attempt to merge adjacent blocks into ones of a larger permitted size (coalescence). To make this easier, the free lists may be stored in order of address. The main advantage of the buddy system is that coalescence is cheap because the "buddy" of any free block can be calculated from its address.



A binary buddy heap after allocating a 10 kB block; Note the 6 kB wasted because of rounding up.

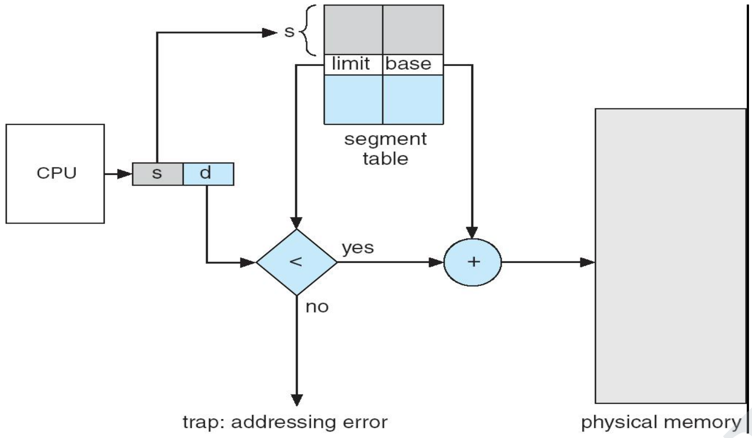
For example, an allocator in a binary buddy system might have sizes of 16, 32, 64, …, 64 kB. It might start off with a single block of 64 kB. If the application requests a block of 8 kB, the allocator would check its 8 kB free list and find no free blocks of that size. It would then split the 64 kB block into two block of 32 kB, split one of them into two blocks of 16 kB, and split one of them into two blocks of 8 kB. The allocator would then return one of the 8 kB blocks to the application and keep the remaining three blocks of 8 kB, 16 kB, and 32 kB on the appropriate free lists. If the application then requested a block of 10 kB, the allocator would round this request up to 16 kB, and return the 16 kB block from its free list, wasting 6 kB in the process.

	A Fibonacci buddy system might use block sizes 16, 32, 48, 80, 128, 208, … bytes, such that each size is the sum of the two preceding sizes. When splitting a block from one free list, the two parts get added to the two preceding free lists. A buddy system can work very well or very badly, depending on how the chosen sizes interact with typical requests for memory and what the pattern of returned blocks is. The rounding typically leads to a significant amount of wasted memory, which is called internal fragmentation. This can be reduced by making the permitted block sizes closer together.
17	**Explain why sharing a re-entrant module is easier when segmentation is used than when pure paging is used with example.(April 2018)** Let's first assume no paging in the system User generates logical addresses These addresses consist of a segment number and an offset into the segment Use segment number to index into a table Table contains the physical address of the start of the segment often called the base address Add the offset to the base and generate the physical address before doing this, check the offset against a limit the limit is the size of the segment **User's View of a Program**  **Segmentation** Segmentation is a technique for breaking memory up into logical pieces Each "piece" is a grouping of related information data segments for each process code segments for each process data segments for the OS etc.

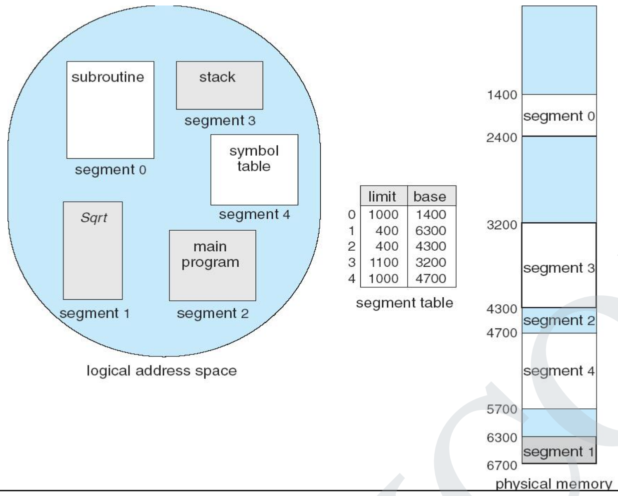
	Like paging, use virtual addresses and use disk to make memory look bigger than it really is Segmentation can be implemented with or without paging Segmentation is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset. For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Tluis, a logical address consists of a *two tuple:* ***<segment-number, offset >.*** **Hardware** Each entry in the segment table has a *segment base* and a *segment limit.* The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment. A logical address consists of two parts: a segment number, *s,* and an offset into that segment, *d.* The segment number is used as an index to the segment table. The offset *d* ofthe logical address must be between 0 and the segment limit. If it is not, we trapto the operating system (logical addressing attempt beyond, end of segment). When an offset is legal, it is added to the segment base to produce the addressin physical memory of the desired byte. **Segment-table base register (STBR)** points to the segment table's location in memory **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*****<STLR** ***Protection*** With each entry in segment table associate: validation bit = 0 ⇒ illegal segment read/write/execute privileges Protection bits associated with segments; code sharing occurs at segment level Since segments vary in length, memory allocation is a dynamic storage-allocation

problem

Segmentation Hardware



Example of Segmentation



18 **Discuss situation under which the most frequently used page replacement algorithm generates fewer page faults than the least recently used page replacement algorithm. Also discuss under which circumstance the opposite holds.(April 2018)**

Consider the sequence in a system that holds four pages in memory: 1 23 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence "1 2 3 4 4 4 51," the LRU algorithm makes the right decision.

19 **Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.**

Answer: Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

20	Most systems allow programs to allocate more memory to its address space during execution.Data allocated in the heap segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes: a. contiguous-memory allocation b. pure segmentation c. pure paging contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space. • pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space. • pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space.

UNIT IV I/O SYSTEMS

Mass Storage Structure- Overview, Disk Scheduling and Management; File System Storage-File Concepts, Directory and Disk Structure, Sharing and Protection; File System Implementation- File System Structure, Directory Structure, Allocation Methods, Free Space Management; I/O Systems.

PART-A

1	**Define rotational latency (May 2013) and seek time (May 2012)** Rotational latency - additional time waiting for the disk to rotate the desired sector to the disk head. Seek time - time for the disk arm to move the heads to the cylinder containing the desired sector.
2	**Define disk bandwidth** The disk bandwidth - total number of bytes transferred, divided by the time between the first request for service and the completion of the last transfer.
3	**What is the need for disk scheduling algorithm? (Nov 2012)**

	The disk-scheduling algorithms focus primarily on minimizing the amount of disk head movement in magnetic disk drives.
4	**What is low-level formatting or physical formatting?** Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting or physical formatting. Low-level formatting fills the disk with a special data structure for each sector. Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC).
5	**What is logical formatting?** *Logical formatting* or "making a file system" – OS stores the initial file-system data structures onto the disk. These DS may include maps of free and allocated space (FAT or inodes) and an initial empty directory
6	**Define boot partition and Master Boot Record (MBR).** Boot partition contains the operating system and device drivers. The Windows system places its boot code in the first sector on the hard disk, which it terms the Master Boot Record ,or MBR. Booting begins by running code that is resident in the system's ROM memory. This code directs the system to read the boot code from the MBR. In addition to containing boot code, the MBR contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.
7	**What is the use of boot block?** For a computer to start running when powered up or rebooted it needs to have an initial program to run. This bootstrap program tends to be simple. It finds the operating system on the disk loads that kernel into memory and jumps to an initial address to begin the operating system execution. The full bootstrap program is stored in a partition called the boot blocks, at fixed location on the disk. A disk that has boot partition is called boot disk or system disk.

8	**What is sector sparing?** Low-level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.
9	**Give the importance of swap-space management. (Nov 2012)** Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system.
10	**What is a file? List the various file attributes and operations. (Nov 2012)(Nov 2018)** A file is a named collection of related information that is recorded on secondary storage. A file contains either programs or data. A file has certain "structure" based on its type. File attributes: Name, identifier, type, size, location, protection, time, date File operations: creation, reading, writing, repositioning, deleting, truncating, appending, renaming File types: executable, object, library, source code etc.
11	**What is the information associated with an open file?** Several pieces of information are associated with an open file which may be: File pointer - pointer to last read/write location, per process that has the file open File-open count - counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it Disk location of the file - cache of data access information Access rights - per-process access mode information
12	**What are the different accessing methods of a file?** Sequential access: Information in the file is accessed sequentially Direct access: Information in the file can be accessed without any particular

	order. Other access methods: Creating index for the file, indexed sequential access method (ISAM) etc.
13	**What is Directory? What are the operations that can be performed on a directory?** The device directory or simply known as directory records information-such as name, location, size, and type for all files on that particular partition. The directory can be viewed as a symbol table that translates file names into their directory entries. The operations that can be performed on a directory are Search for a file, Create a file, Delete a file, Rename a file, List directory, Traverse the file system
14	**Define UFD and MFD.** In the two-level directory structure, each user has own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. When a job starts the system's 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.
15	**What is a path name? What are the types of path names?** A pathname is the path from the root through all subdirectories to a specified file. In a two-level directory structure a user name and a file name define a path name. Two types of path names: absolute and relative. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory.
16	**Define File system mounting** A file system must be mounted before it can be accessed or it can be available to processes on the system. OS is given the name of the device, and the location within the file structure at which to attach the file system is called mount point.
17	**What is consistency semantics in file sharing?**

	Consistency semantics specify how multiple users are to access a shared file simultaneously and should specify when modifications of data by one user are observable by other users.
18	**Define Network File System** On distributed systems, files may be shared across a network. Network File System (NFS) is a common distributed file-sharing method.
19	**What is Distributed Information Systems?** Distributed information systems known as distributed naming services, provide unified access to the information needed for remote computing to make client – server systems easier to manage.
20	**What is access control list?** Access control list is used for specifying the user name and type of access allowed to access the each user.
21	**Define FCB.** File Control Block (FCB) – storage structure consisting of information about a file file permissions file dates (create, access, write) file owner, group, ACL file size file data blocks or pointers to file data blocks
22	**What are the various layers of a file system?** The file system is composed of many different levels. Each level in the design uses the feature of the lower levels to create new features for use by higher levels.Application programs, Logical file system, File-organization module, Basic file system, I/O control, Devices
23	**What are the functions of virtual file system (VFS)?** It has two functions It separates file-system-generic operations from their implementation defining a clean VFS interface. It allows transparent access to different

	types of file systems mounted locally. VFS is based on a file representation structure, called a v node. It contains a numerical value for a network-wide unique file .The kernel maintains one v node structure for each active file or directory.
24	**What is FAT?** File-allocation table (FAT) is an important variation of linked allocation. Disk-space allocation used by MS-DOS and OS/2. It has one entry for each disk block, & is indexed by block number
25	**What are port, bus and controller?** Port – connection point through which the device communicates with machine Bus – set of wires – widely used (daisy chain or shared direct access) Controller (host adapter) – collection of electronics that can operate a port, a bus or a device
26	**How to maintain free space list?** There are two methods for maintaining free space list. Linked list: Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. First block contains a pointer to the next free disk block and so on. Bit vector Grouping Counting
27	**What are the responsibilities of file manager? (May 2013)** The file manager responsible for the maintenance of secondary storage. implements the abstraction and provides directories for organizing files. also provides a spectrum of commands to read and write the contents of a file, to set the file read/write position, to set and use the protection mechanism, to change the ownership, to list files in a directory, and to remove a file. provides a protection mechanism to allow machine users to administer how

	processes executing on behalf of different users can access the information in files.
28	**What is meant by free space management? (Nov 2012)** Free-Space Management Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. When a file is deleted, its disk space is added to the free-space list.
29	**Compare bitmap-based allocation of blocks on disk with a free block list. (Nov 2014)** **Bitmap-based allocation** - each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0. The **main advantage of this approach** is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk. Indeed, many computers supply bit-manipulation instructions that can be used effectively for that purpose. **Free block list** - link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on. **This scheme is not efficient**; to traverse the list, we must read each block, which requires substantial I/O time.
30	**What are the different I/O registers?** Status register – contain bits that can be read by host, there has been a device error Control register – can be written by the host to start a command or to change the mode of a device Data-in register – is read by the host to get input

	Data-out register – is written by the host to send
31	**Define polling.** Host signals its wishes via the command-ready bit in the command register. Host repeatedly reads the busy bit until that bit becomes clear.
32	**What is the need for DMA?** DMA - Used to avoid programmed I/O for large data movement - Requires DMA controller - Bypasses CPU to transfer data directly between I/O device and memory. DMA controller seizes the memory bus, the CPU is momentarily prevented from accessing main memory (it can still access data items in its primary and secondary cache) ➔ cycle stealing
33	**Compare Character device and block device.** **Block devices include disk drives** - Understands commands include read(), write(), seek() - Raw I/O or file-system access (simple linear array of blocks) - Memory-mapped file access possible - layered on top of block-device derivers - same mechanism as demand-paged virtual-memory access **Character devices include keyboards, mice, serial ports** - Commands include get(), put() - Libraries layered on top allow line editing
34	**What is blocking and non-blocking I/O?** **Blocking I/O-** process suspended until I/O completed - Easy to use and understand - Insufficient for some needs **Non-blocking I/O** - I/O call returns as much as available - User interface, data copy (buffered I/O) - Eg. User interface receives keyboard and mouse i/p while processing and displaying data on the screen - Returns quickly with count of bytes read or written
35	**What is an I/O buffer? (Nov 2014) / Define buffering.** A buffer is a memory area that stores data while they are transferred

	between two devices or between a device and an application. Buffering is done for three reasons To cope with a speed mismatch between the producer and consumer of a data stream To adapt between devices that have different data-transfer sizes To support copy semantics for application I/O
36	**Define caching.** A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes.
37	**Define spooling.** A spool is a buffer that holds output for a device, such as printer, that cannot accept interleaved data streams. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer. The spooling system copies the queued spool files to the printer one at a time.
38	**Define streams.** **STREAM** – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond. A STREAM consists of STREAM head interfaces with the user process driver end interfaces with the drivers Zero or more STREAM modules between them.
39	**Define SAN.** SAN – Storage Area Network is a private network among the servers and storage units. It is common in large storage environments (and becoming more common). Here multiple hosts are attached to multiple storage arrays .
40	**What is HSM? Where it is used? (April 2015)** Hierarchical storage management (HSM) is a data storage technique, which

	automatically moves data between high-cost and low-cost storage media. HSM systems exist because high-speed storage devices, such as hard disk drive arrays, are more expensive (per byte stored) than slower devices, such as optical discs and magnetic tape drives. While it would be ideal to have all data available on high-speed devices all the time, this is prohibitively expensive for many organizations.
41	**A disk has 2310 cylinders, 16 tracks and 63 sectors. The disk spins at 7200 rpm. Seek time between adjacent tracks is 1ms. How long does it take to read the entire disk? (Nov 2015)** Bytes per cylinder ,b=512bytes*63sectors*16tracks=516096 bytes Rotation Speed=7200 rotations per minute (1minute /7200 rotations)*(60 seconds/1minute)=60 seconds/7200 rotations=8.33ms Seek time=1ms Disk has 63 sectors per track ,performs rotation in 8.33ms. Thus transfer time, x =(16/63)*8.33ms=2.11ms t_{read}, Total time to read 16 tracks in the disk=8.33+1 ms=9.33ms
42	**Identify the 2 important functions of Virtual file System layer in the concept of system implementation.(Nov 2015)** The VFS layer serves two important functions: It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally. It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a **vnode**, that contains a numerical designator for a network-wide unique file.
43	**Why is rotational latency usually not considered in disk scheduling?(April**

	2016) Most disks do not export their rotational position information to the host. Even if they did, the time for this information to reach the scheduler would be subject to imprecision and the time consumed by the scheduler is variable, so the rotational position information would become incorrect. Further, the disk requests are usually given in terms of logical block numbers, and the mapping between logical blocks and physical locations is very complex.
44	**How does DMA increase system concurrency (April 2016)** DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.
45	**Define c-scan scheduling (Nov 2016)** Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
46	**Why is it important to scale up system bus and device speed as CPU speed increases? (Nov 2016)** Consider a system which performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to remove the current system bottleneck, and to increase overall system performance, rather than

	blindly increasing the performance of individual system components.
47	**Suppose that the disk rotates at 7200 rpm. What is the average rotational latency of the disk drive? (April 2017)** The disk rotates at 7200 rpm; i.e., it makes one rotation in 8.33 milliseconds. The average rotational latency is the time to rotate the disk half way around, or 4.17 milliseconds. MAXIMUM Rotational Latency: 7200RPM = 60 / 7,200 seconds per rotation = 8.3 milliseconds (meaning how long does one point on a disk need to achieve an entire rotation throughout its path) AVERAGE Rotational Latency: 7200RPM = 0.5 * 60 / 7,200 seconds per rotation = 4.2 milliseconds (= 4.17 milliseconds) = MAXIMUM Rotational Latency / 2
48	**Differentiate between file and directory (April 2017)** Directory is a collection of files stored as a group , say it is a group of files with single name. A directory can be classified into two types **Root Directory** : root is the parent of total directories , say main drive or main filesystem(/) is the root directory . **Sub directory** : these are the directories that comes under the root directory in hierarchy. In general a directory within a director can be called as sub directory. Files are collection of data items stored in a disk or it is a device which can store the information like data, music (mp3,ogg), photographs, movie, sound, book etc. It is like everything you store in a system should be a file. Files are always associated with devices like hard disk ,floppy disk etc. File is the last object in your file system tree.
49	**State the typical bad sector transactions.(April 2018)** A bad sector is an unusable part or subdivision within a track on a magnetic or optical disc located on a computer's hard disk or flash drive. It typically forms as a result of physical damage or, rarely, the operating system's (OS) inability to access the information.

50	**What are the advantages of bit vector approach in free space management? (April 2018)** To keep track of free space, system maintains a free-space list **Bit vector bit map** (n blocks) Eg. Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free, and the rest of the blocks are allocated. Free-space bit map would be: 001111001111110001100000011100000 Main Advantage: simplicity and efficiency in finding the first free block or n consecutive free blocks on the disk.
51	**Do FAT file system advantageous? justify your answer.(Nov 2018)** The main advantage of FAT is its efficient use of disk space. FAT can place the parts of the file wherever they fit. File names can be up to 255 characters and file extensions longer than 3 characters. Easy to recover file names that have been deleted.
	PART-B
1	**Explain the various disk scheduling algorithms with examples. (April 2011,2015,2016,2018)** The operating system is responsible for using hardware efficiently — for the disk drives, this means having a **fast access time and disk bandwidth.** **Access time has two major components** ***Seek time*** is the time for the disk are to move the heads to the cylinder containing the desired sector. Minimize seek time Seek time ≈ seek distance ***Rotational latency*** is the additional time waiting for the disk to rotate the desired sector to the disk head. **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. Improve seek time and disk bandwidth by scheduling the servicing of disk I/O requests in a good order Illustration with a request queue (0-199). 98, 183, 37, 122, 14, 124, 65, 67 Head pointer at 53

Type of Disk Scheduling

1. FCFS –The I/O requests are serviced strictly in the same order as they are received.

SSTF Selects the request with the minimum seek time from the current head position.

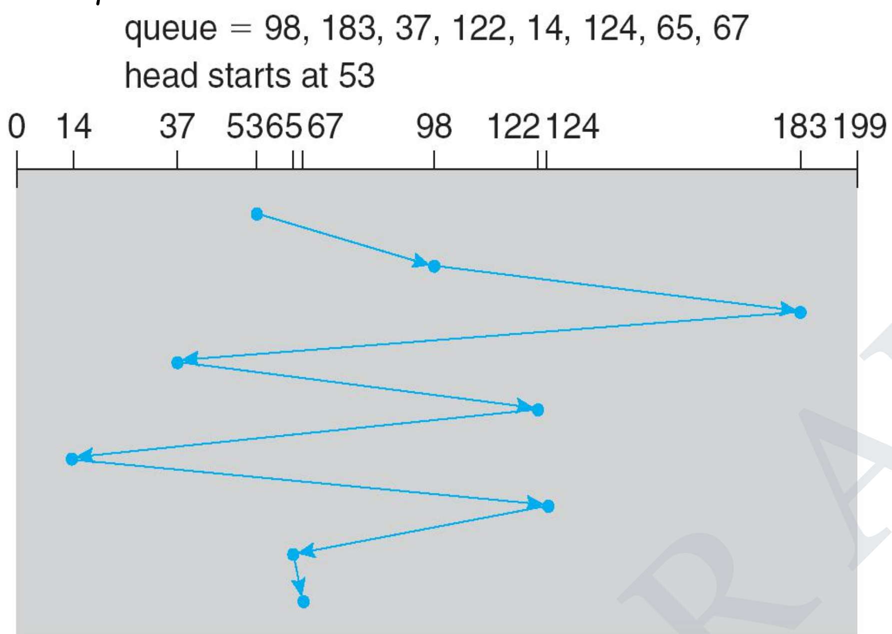


Illustration shows total head movement of 640 cylinders.

2. SSTF **-Shortest Seek Time First**

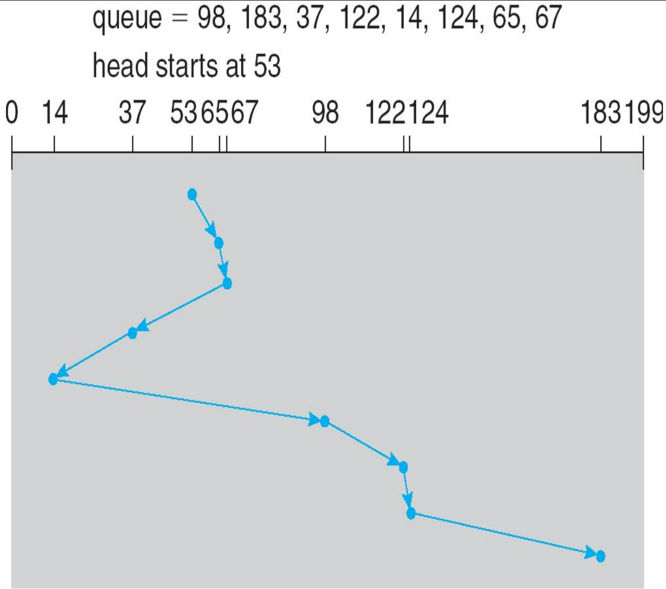
Selects the request with the minimum seek time from the current head position.

Seek time increases with no. of cylinders traversed by the head

SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Not optimal

Illustration shows total head movement of 236 cylinders.



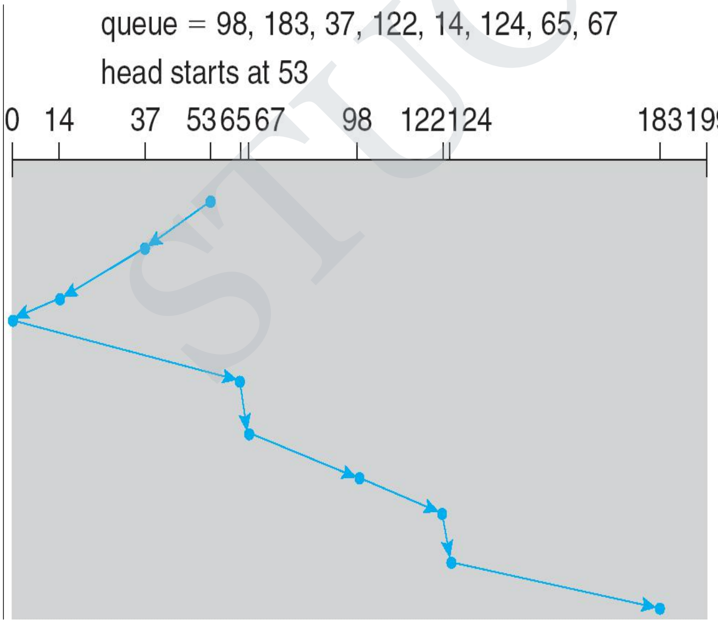
3. SCAN –The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

If request arrives in the queue just in front of the head, it will be serviced almost immediately

Request arrives just behind the head will have to wait until the arm moves to the end of disk, reverses direction & comes back

Sometimes called the *elevator algorithm*.

Illustration shows total head movement of 208 cylinders.



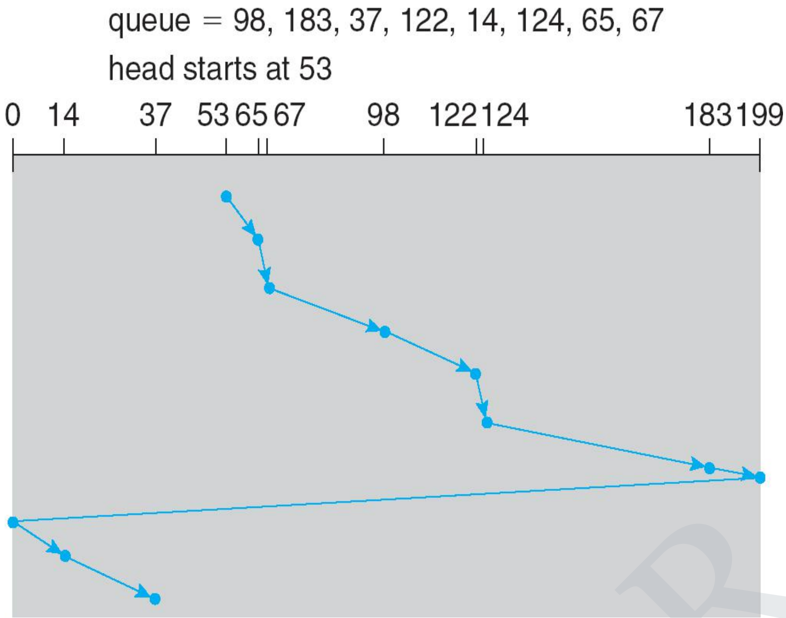
4. C-SCAN

Circular SCAN

Provides a more uniform wait time than SCAN.

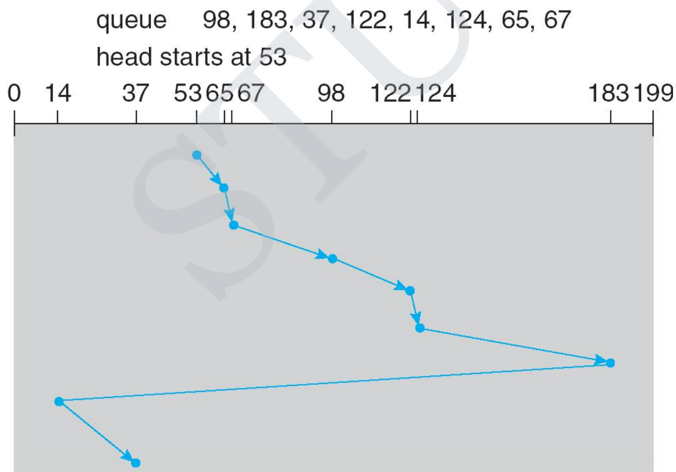
The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



5. Look-The disk arm moves once from cylinder 0 side to cylinder n side and then back to cylinder 0 side and keeps servicing requests on the way.

6. C-Look-Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



2 **i) What is disk management? Explain in detail how to improve the disk performance. (Nov 2014)**

OS is responsible for several other aspects of disk management

Disk initialization

	Booting from disk and Bad – block recovery ***1. DISK formatting*** *New magnetic disk – blank slate* *Low-level formatting, or physical formatting*— Dividing a disk into sectors that the disk controller can read and write. Fills the disk with a special data structure for each sector – consists of a header, a data area (usually 512 bytes in size) and a trailer Header and trailer – contain information used by the disk controller such as sector no. and error-correcting code (ECC) When the controller writes a sector of data, the ECC is updated with a value computed from all the bytes in the data area When sector is read, the ECC is recalculated & is compared with the stored value; mismatch indicates that the data area of sector has become corrupted and that disk sector may be bad. Low-level formatted at the factory as a part of manufacturing process To use a disk to hold files, the operating system still needs to record its own data structures on the disk – in 2 steps *Partition* the disk into one or more groups of cylinders. *Logical formatting* or "making a file system" – OS stores the initial file-system data structures onto the disk. These DS may include maps of free and allocated space (FAT or inodes) and an initial empty directory **2. Boot block** – initializes system. **Initial bootstrap pgm**tends to be simple – initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory and then starts the OS **The bootstrap is stored in ROM** – convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset – read only hence it can not be

infected by virus

Full bootstrap pgmis stored in a partition called the boot blocks, at a fixed location on the disk

A disk that has a boot partition is called a **boot disk or system disk**

Bootstrap loader program - Code in the boot ROM instructs the disk controller to read the boot blocks into memory and then executing that code

Full bootstrap pgm is more sophisticated than the Bootstrap loader in the boot ROM – able to load the entire os from a non-fixed location on disk and to start the os running

3. Bad blocks

- one or more sectors become defective

Methods to handle bad blocks:

Sector Sparing or forwarding – controller can be told to replace each bad sector logically with one of the spare sectors

- low level formatting sets aside spare sectors not visible to os

Controllers are instructed to replace bad block by **Sector slipping**

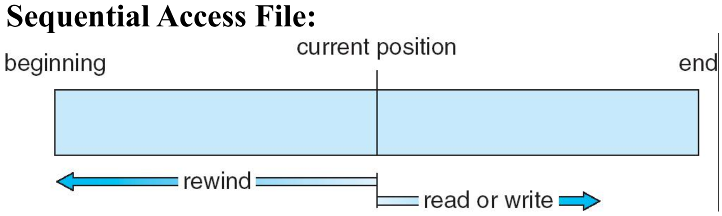
eg. Logical block 17 becomes defective

first available spare follows sector 202 , then sector slipping remaps all the sectors from 17 to 202, moving them all down one spot sector 202 would be copied into the spare, then 201 into 202, 200 into 201 …… 18 into 19, so sector 17 into 18

Describe three circumstances under which blocking I/O should be used. Describe three under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready? **(Nov 2014)**

Blocking I/O is appropriate when the process will only be waiting for one specific event.

Examples include a disk, tape, or keyboard read by an application program.

	Non-blocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using non-blocking I/O to keep both devices fully occupied. Non-blocking I/O is more complicated for programmers, because of the asynchonous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O so the overall system performance would decrease.
3	**i)What are files and explain the access methods for files? (April 2011) (April 2017)** **File** – named collection of related information that is recorded on secondary storage **File Access Methods** **1. Sequential Access –** information in the file is processed in order, one after the other - read operations read the next portion of the file & automatically advances a file pointer read next write next reset no read after last write (rewrite) **Sequential Access File:**  **2. Direct Access or relative access – access any file block, file is viewed as numbered**

sequence of blocks or records

readn

writen

position to n

read next

write next

rewrite*n*

n = relative block number

Simulation of Sequential Access on a Direct-access File

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Other access methods

- built on top of a direct access method
- involves the construction of an index for the file

1) **Index** – contains pointers to the various blocks

To find a record in the file,

first search the index and then

use the pointer to access the file directly and

to find the desired record

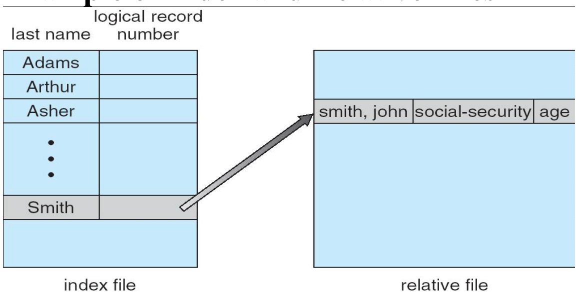
Problem: With large files, the index file itself may become too large to be kept in memory

Solution: create index for the index file – primary index file would contain pointers to secondary index files, which would point to the actual data items

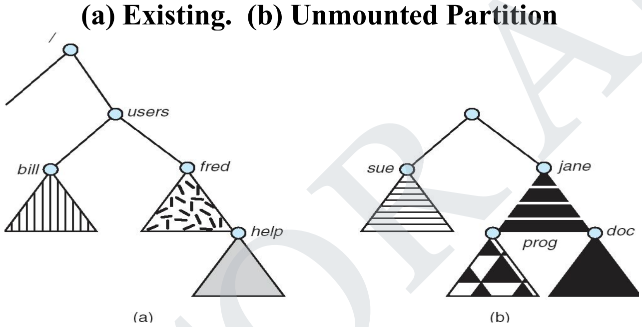
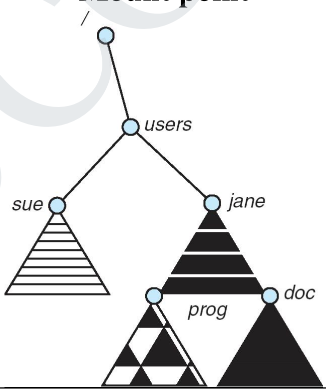
IBM's Indexed Sequential-access Method (ISAM)

- uses small master index that points to disk blocks of a secondary index, secondary blocks point to the actual file blocks

Example of Index and Relative Files



Explain about file system mounting with example.

	A file system must be mounted before it can be accessed or it can be available to processes on the system OS is given the name of the device, and the location with in the file structure at which to attach the file system – mount point Mount point - empty dir at which the mounted file system will be attached On UNIX, a file system containing user's home directories might be mounted as /home, to access the dir structure with in the file system, precede the directory names with /home A unmounted file system (i.e. Fig.) is mounted at a mount point MS-Windows family of OS – maintains an extended 2-level dir structure, with devices and partitions assigned a drive letter **(a) Existing. (b) Unmounted Partition**  **Mount point** 
4	Suppose that a disk drive has 5,000 cylinders, numbered from 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms? The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081. The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750,

	1774. The total seek distance is 1745. The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 130, 86. The total seek distance is 9769. The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek distance is 3319. The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 0, 86, 130. The total seek distance is 9985. The C-LOO K schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130. The total seek distance is 3363
5	**Explain about file protection and sharing. (April 2011)** **File Sharing:** Sharing of files on multi-user systems is desirable Sharing may be done through a protection scheme On distributed systems, files may be shared across a network Network File System (NFS) is a common distributed file-sharing method **File Sharing – Multiple Users** User IDs identify users, allowing permissions and protections to be per-user Group IDs allow users to be in groups, permitting group access rights **File Sharing – Remote File Systems** Uses networking to allow file system access between systems Manually via programs like FTP Automatically, seamlessly using distributed file systems Semi automatically via the world wide web Client-server model allows clients to mount remote file systems from servers Server can serve multiple clients Client and user-on-client identification is insecure or complicated

	NFS is standard UNIX client-server file sharing protocol CIFS is standard Windows protocol Standard operating system file calls are translated into remote calls Distributed Information Systems (distributed naming services) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing **File Sharing – Failure Modes** Remote file systems add new failure modes, due to network failure, server failure Recovery from failure can involve state information about status of each remote request Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security RAID – prevent loss of data **Consistency Semantics** **Consistency semantics** specify how multiple users are to access a shared file simultaneously should specify when modifications of data by one user are observable by other users Similar to Ch 7 process synchronization algorithms Tend to be less complex due to disk I/O and network latency (for remote file systems Andrew File System (AFS) implemented complex remote file sharing semantics Unix file system (UFS) implements: Writes to an open file visible immediately to other users of the same open file Sharing file pointer to allow multiple users to read and write concurrently AFS has session semantics Writes only visible to sessions starting after the file is closed **File Protection** File owner/creator should be able to control: what can be done

by whom

Types of access

Read

Write

Execute

Append

Delete

List

Access Lists and Groups

Mode of access: read, write, execute

Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1RWX
b) **group access**	6	⇒	1 1 0RWX
c) **public access**	1	⇒	0 0 1

Ask manager to create a group (unique name), say G, and add some users to the group.

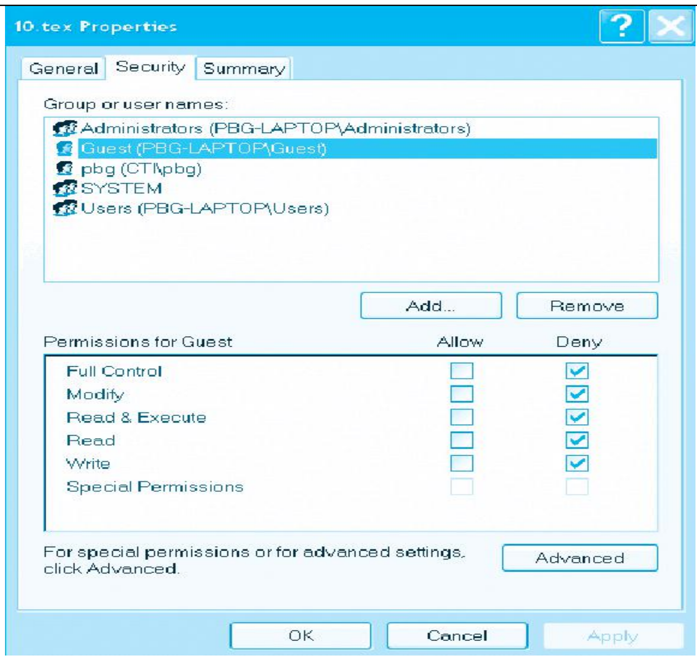
For a particular file (say *game*) or subdirectory, define an appropriate access.

Owner Group Public

chmod 761 game

Attach a group to a file: chgrp G game

Windows-XP Access-control List Management



A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Explain about directory implementation.

Directory Implementation

1) **Linear list of file names with pointer to the data blocks.**

simple to program

time-consuming to execute

To create a new file, a) search the dir to be sure that no existing file has the same name b) add new entry at the end of the dir

To delete a file, a) search the dir for the named file b) release the space allocated to it

Disadv: linear search to find a file

Remedy: - many OS implement a s/w cache to store most recently used dir information

sorted list allows a binary search and decreases the avg search time

2). Hash Table – linear list with hash data structure.

takes a value computed from the file name & returns a pointer to the filename in the linear list decreases directory search time

insertion and deletion are also fairly straightforward

Problem : a) collisions – situations where two file names hash to the same location

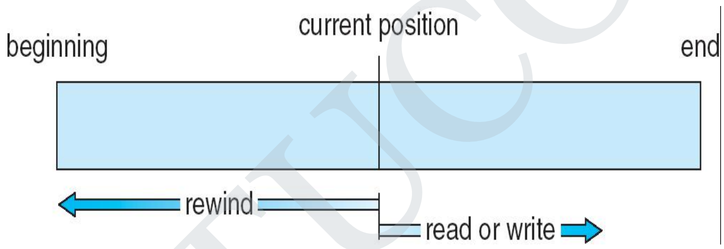
b) hash table is fixed size and dependence of the hash function on that size.

6 **Discuss the different techniques with which a file can be shared among different users. (Nov 2014) (April 2017)**

Sequential Access

Information in the file is processed in order, one after the other. Read operations reads the next portion of the file & automatically advances a file pointer

read next
write next
reset
no read after last write
(rewrite)



Direct Access or relative access – access any file block, file is viewed as numbered sequence of blocks or records

readn
writen
position to n
read next
write next
rewriten

n = relative block number

Simulation of Sequential Access on a Direct-access File

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Other access methods

– built on top of a direct access method

– involves the construction of an index for the file

Index – contains pointers to the various blocks

To find a record in the file, first search the index and then use the pointer to access the file directly and to find the desired record.

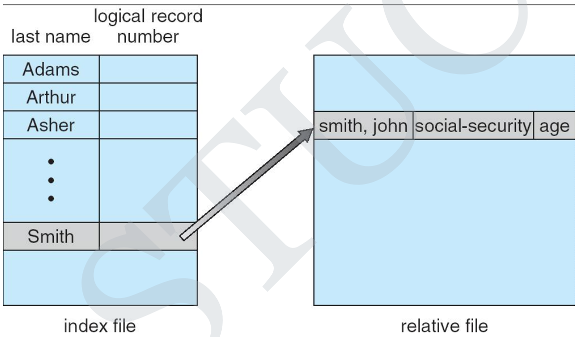
Problem: With large files, the index file itself may become too large to be kept in memory

Solution: Create index for the index file – primary index file would contain pointers to secondary index files, which would point to the actual data items

IBM's Indexed Sequential-access Method (ISAM)

– uses small master index that points to disk blocks of a secondary index, secondary blocks point to the actual file blocks

Example of Index and Relative Files



Refer Previous question for file sharing Q5

What is File protection and security? Compare both and also explain the techniques to protect user files. (Nov 2014)

Refer Previous question (Q,No. 6) for file protection

File Security:

Security, on the other hand, requires not only an adequate protection system but also consideration of the external environment within which the system operates. program threats and other security techniques

7 **Explain the different file allocation methods for disk space. Mention their**

advantages and disadvantages. (Nov 2012)

An allocation method refers to how disk blocks are allocated for files:

1) Contiguous allocation
2) Linked allocation
3) Indexed allocation

Contiguous Allocation

Each file occupies a set of contiguous blocks on the disk

Simple – only starting location (block #) and length (number of blocks) are required

For sequential access, the file system remembers the disk address of the last block reference and then reads the next block.

For direct access to block i of a file that starts at block b immediately access block b+i

Both sequential and direct access can be supported.

Difficulties :

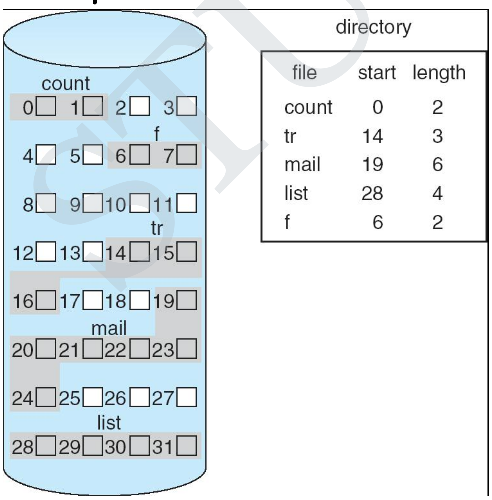
a) finding space for a new file

b) Wasteful of space (dynamic storage-allocation problem)

c) External fragmentation problem (solve by compaction, but more down time)

d) Files cannot grow

Example:

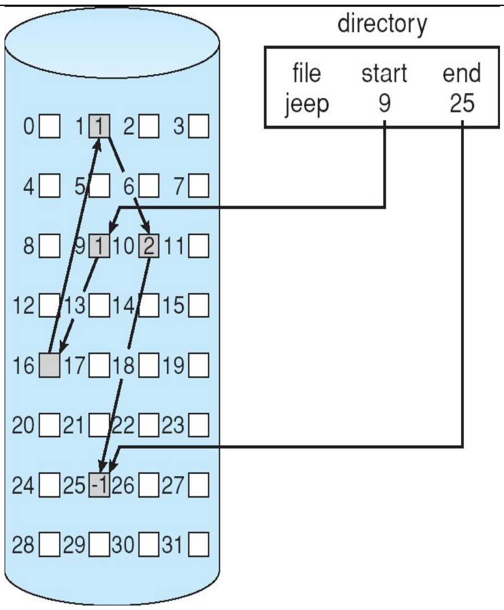


Extent-Based Systems

Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme

Extent-based file systems allocate disk blocks in extents (another

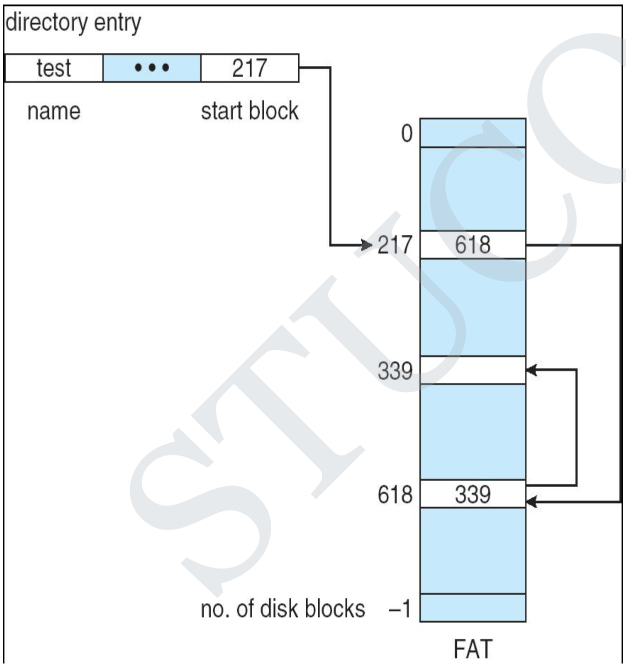
	chunk of contiguous space) An extent is a contiguous block of disks Extents are allocated for file allocation A file consists of one or more extents. The location of file's blocks is then recorded as a location & a block count, plus a link to the first block of the next extent Internal fragmentation problem if the extents are too large External fragmentation problem if extents are of varying sizes are allocated and deallocated **Linked Allocation:** Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk. Directory contains a pointer to the first and last blocks of the file Simple – need only starting address Free-space management system – no waste of space No random access No external fragmentation File can continue to grow as long as free blocks are available Disadvantages: 1) used effectively for sequential access files 2) space required for the pointers (eg: if a pointer requires 4 bytes out of 512-byte block, then 0.78% of the disk is being used for pointers) Solution: collect blocks into multiples called clusters (cluster of 4 blocks), but internal fragmentation problem 3) reliability – what would happen if a pointer were lost or damaged partial solution: use doubly linked lists or to store the file name & relative block number in each block → require more overhead for each file



File-Allocation Table

File-allocation table (FAT) – important variation of linked allocation - - disk-space allocation used by MS-DOS and OS/2.

– has one entry for each disk block, & is indexed by block number



Benefit:

Random access time improved, because the disk head can find the location of any block by reading the information in the FAT

Indexed Allocation

Brings all pointers together into the index block.

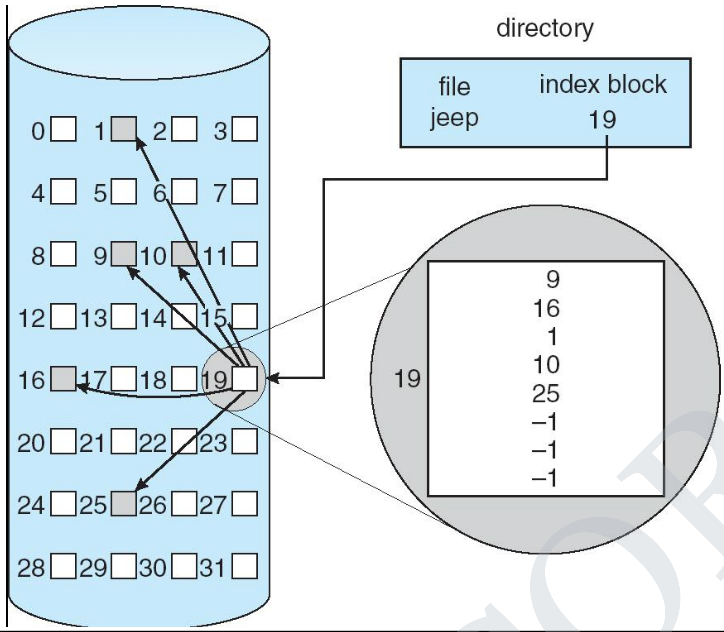
Need index table

Supports Random access

Dynamic access without external fragmentation, but have overhead of index block.

Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

Entire index block must be allocated , even if only one or two pointers



8 **Write in detail about File-System Implementation (Nov 2014) (Nov 2015)**

To implement file system

1) on-disk data structure

2) in-memory data structure

1) On-disk structure includes

a) boot control block

b) partition control block – contains partition details such as

i) no. of blocks in the partition

ii) size of the blocks

iii) free-block count

iv) free-block pointers

v) FCB count and pointers

c) directory structure to organize the files

d) FCB

File Control Block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

2) In-Memory File System Structures include:

In-memory partition table – contains information about each mounted partition

In-memory directory structure – holds the dir information of recently accessed directories

System-wide open-file table – contains a copy of the FCB of each open file, as well as other information

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

To create a new file:

Application pgm calls the logical file system knows the format of dir structure

2) Logical file system allocates a new FCB,

- reads the appropriate dir into memory,
- updates it with the new file name and FCB
- writes it back to the disk

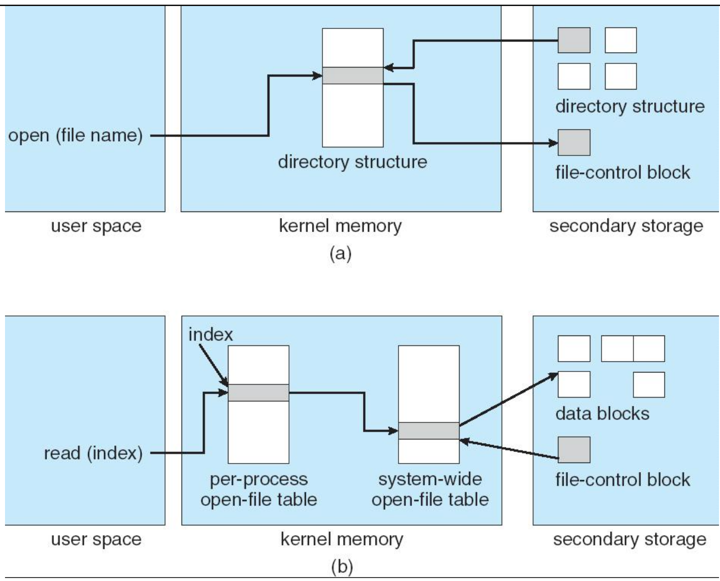
To do I/O:

file must be opened

The following figure illustrates the necessary file system structures provided by the operating systems.

Figure (a) refers to opening a file.

Figure (b) refers to reading a file.



A file system on a disk has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory using contiguous, linked and indexed allocation strategies answer the following questions. (Nov 2014)

Answer:

Let Z be the starting file address (block number).

a. Contiguous allocation. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

i. Add X to Z to obtain the physical block number. Y is the displacement into that block.

ii. 1 block

b. Linked allocation. Divide the logical physical address by 511 with X and Y the resulting quo-tient and remainder respectively.

i. Chase down the linked list (getting X + 1 blocks). Y + 1 is the displacement into the last physical block.

ii. 4 blocks

c.Indexed allocation. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

i. Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.

ii. 2 blocks

9

Explain the issues in designing a file system.

Naming and Name Resolution:

Name refers to an object such as file or a directory. Name Resolution refers to the process of mapping a name to an object that is physical storage. Name space is collection of names. Names can be assigned to files in distributed file system in three ways:

a) Concatenate the host name to the names of files that are stored on that host.

Advantages:

File name is unique

System wide

Name resolution is simple as file can be located easily.

Limitations:

It conflicts with the goal of network transparency.

Moving a file from one host to another requires changes in filename and the application accessing that file that is naming scheme is not location independent.

b) Mount remote directories onto local directories. Mounting a remote directory require that host of directory to be known only once. Once a remote directory is mounted, its files can be referred in location transparent way. This approach resolve file name without consulting any host.

c) Maintain a single global directory where all the files in the system belong to single namespace. The main limitation of this scheme is that it is limited to one computing facility or to a few co-operating computing facilities. This scheme is not used generally.

Name Server: It is responsible for name resolution in distributed system. Generally two approaches are used for maintaining name resolution information.

Way 1: Use a single name server that is all clients send their queries to single server which maps names to objects. Its limitation is: If name server fails, the entire system is affected and Name server become a bottleneck and degrades the performance of the system.

Way 2: Use several name servers(on different hosts) wherein each server is responsible for a mapping objects stored in different domains. This approach is generally used. Whenever a name is to be mapped to an object, the local name server is queried. The local name server may point to remote server for further mapping of the name. Example: "a/b/c"- requires a remote server mapping the /b/c part of the filename. This procedure is repeated until the name is completely resolved.

2. Caches on Disk or Main Memory:

Caching refers to storage of data either into the main memory or onto disk space after its first reference by client machine.

Advantages of having cache in main memory:

Diskless workstations can also take advantage of caching.

Accessing a cache in main memory is much faster than accessing a cache on local disk.

The server cache is in the main memory at the server; a single design for a caching mechanism is used for clients and servers.

Limitations:

Large files cannot be cached completely so caching done block oriented which is more complex.

It competes with virtual memory system for physical memory space, so a scheme to deal with memory contention cache and virtual memory system is necessary. Thus, more complex cache manager and memory management is required.

Advantages of having cache on a local disk:

Large files can be cached without affecting performance.

Virtual memory management is simple.

Portable workstation can be incorporated in distributed system.

3. Writing Policy:

This policy decides when a modified cache block at client should be transferred to the server. Following policies are used:

a) Write Through: All writes required by clients applications are also carried out at servers immediately. The main advantage is reliability that is when client crash, little information is lost. This scheme cannot take advantage of caching.

b) Delayed Writing Policy: It delays the writing at the server that is modifications due to write are reflected at server after some delay. This scheme can take advantage of caching. The main limitation is less reliability that is when client crash, large amount of information can be lost.

c) This scheme delays the updating of files at the server until the file is closed at the client. When average period for which files are open is short then this policy is equivalent to write through and when period is large then this policy is equivalent to delayed writing policy.

4. Cache Consistency:

When multiple clients want to modify or access the same data, then cache consistency problem arises. Two schemes are used to guarantee that data returned to the client is valid.

a) Server initiated approach: Server Inform the cache manager whenever data in client caches becomes valid. Cache manager at clients then retrieve the new data or invalidate the blocks containing old data in cache in cache. Server has maintained reliable records on what data blocks are cached by which cache managers. Co-operation between servers and cache manager is required.

b) Client-initiated approach: It is the responsibility of cache manager at the clients to validate data with server before returning it to the clients. This approach does not take benefit of caching as the cache manager consults the server for validation of cached block each time.

5. Availability:

It is one of the important issue is design of Distributed file system. Server failure or communication network can attract the availability of files.

Replication: The primary mechanism used for enhancing availability of files is replication. In this mechanism, many copies or replicas of files are maintained at different servers.

Limitations:

Extra storage space is required to store replicas.

Extra overhead is required in maintained all replicas up to date.

Explain the various directory structures. (Nov 2012) (April 2017)

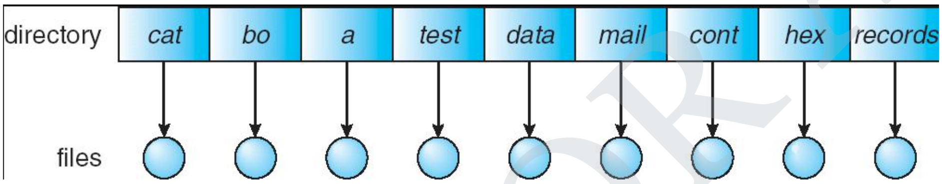
Directory Structure

To organize the files on disk

A collection of nodes containing information about all files

(i) Single-Level Directory

A single directory for all users



Naming problem – if many users call their data file with same name, unique name rule is violated

– difficult to remember the names of all the files, as the no. of files increases

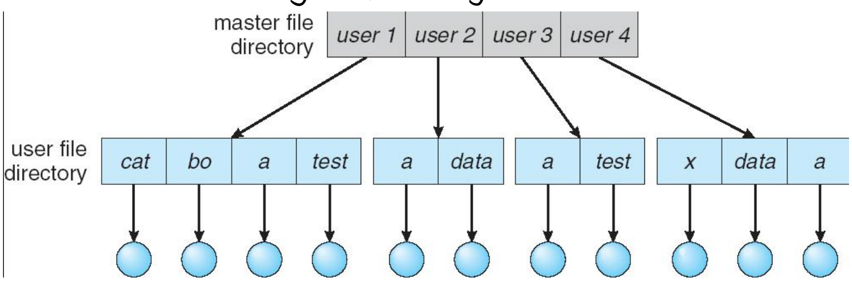
Grouping problem – uncommon for a user to have hundreds of files on one computer system, keeping track of so many files is a difficult task

(ii) Two-Level Directory

Separate directory for each user

Each user has own user file directory (UFD)

When a user logs in, the system's master file directory (MFD) is searched



To create and delete a file for a user, OS confines its search to the local UFD

Advantages:

Solves name-collision problem - Can have the same file name for different user

Efficient searching

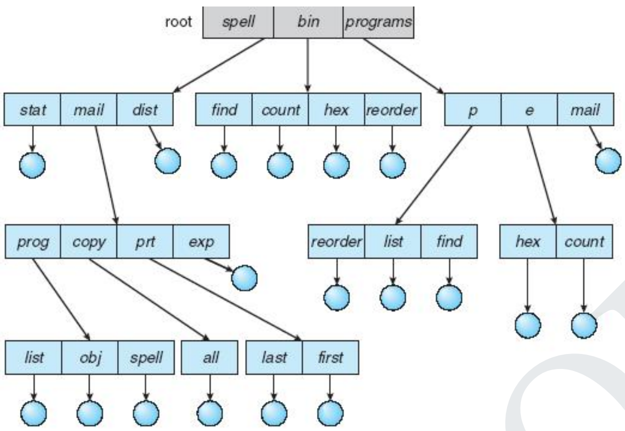
Isolates one user from another

Disadvantages:

users difficult to cooperate with other users (if access is permitted, use path name)

No grouping capability

(iii) Tree-Structured Directories



Efficient searching

Grouping Capability

Current directory (working directory)

cd /spell/mail/prog

type list

Absolute or **relative** path name

Creating a new file is done in current directory

Delete a file

rm<file-name>

Creating a new subdirectory is done in current directory

mkdir<dir-name>

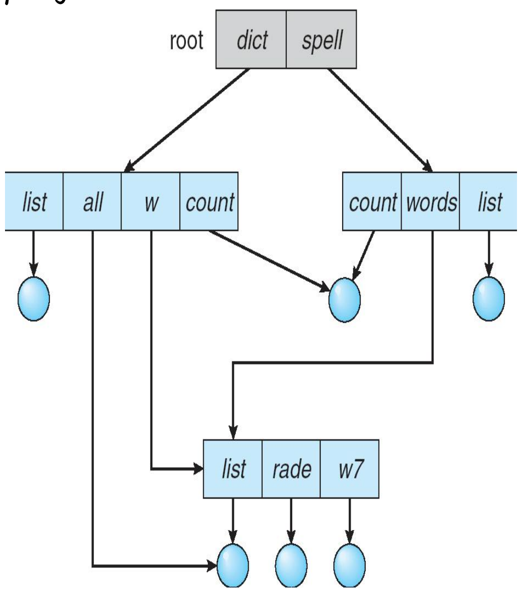
Example: if in current directory /mail

mkdir count

Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

(iv) Acyclic-Graph Directories

Have shared subdirectories and files (two programmers work on a joint project



Two different names (aliasing)

If *dict* deletes *list* ⇒ dangling pointer

Solutions:

Backpointers, so we can delete all pointers

Variable size records a problem

Backpointers using a daisy chain organization

Entry-hold-count solution

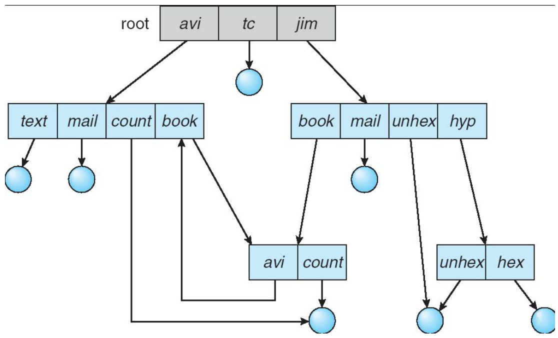
New directory entry type

Link – another name (pointer) to an existing file

Resolve the link – follow pointer to locate the file

Serious problem – ensuring that there in no cycles

(v) General Graph Directory



How do we guarantee no cycles?