Name: Ankush babbar
Enrollno : 03813702011

Operating system – assignment
Question paper- may-june (2009)

Q1.) Discuss the role of operating system.

Answer:

The basic functions of an operating system are:

- Booting the computer
- Performs basic computer tasks eg managing the various peripheral devices eg mouse, keyboard
- Provides a user interface, e.g. command line, graphical user interface (GUI)
- Handles system resources such as computer's memory and sharing of the central processing unit (CPU) time by various applications or peripheral devices
- Provides file management which refers to the way that the operating system manipulates, stores, retrieves and saves data.

1.)Booting the computer

The process of starting or restarting the computer is known as booting. A cold boot is when you turn on a computer that has been turned off completely. A warm boot is the process of using the operating system to restart the computer.

2.)Performs basic computer tasks

The operating system performs basic computer tasks, such as managing the various peripheral devices such as the mouse, keyboard and printers.

3.)Provides a user interface

The operating system is responsible for providing a consistent application program interface (API) which is important as it allows a software developer to write an application on one computer and know that it will run on another computer of the same type even if the amount of memory or amount of storage is different on the two machines.

4.)Handles system resources

The operating system also handles system resources such as the computer's memory and sharing of the central processing unit (CPU) time by various applications or peripheral devices.

5.)Provides file management

The operating system also handles the organisation and tracking of files and directories (folders) saved or retrieved from a computer disk.

(B)Discuss the various scheduling technique of operating system.

Answer :

In many multitasking systems the processor scheduling subsystem operates on three levels, differentiated by the time scale at which they perform their operations:

Long term scheduling: which determines which programs are admitted to the system for execution and when, and which ones should be exited.

Medium term scheduling: which determines when processes are to be suspended and resumed;

Short term scheduling (or dispatching): which determines which of the ready processes can have CPU resources, and for how long.

Short-term scheduler, also known as the process or CPU scheduler, controls the CPU sharing among the

‘‘ready’’ processes. The selection of a process to execute next is done by the short-term scheduler. Usually, a new process is selected under the following circumstances:

- When a process must wait for an event.
- When an event occurs (e.g., I/O completed, quantum expired).
- When a process terminates.

Long term scheduling obviously controls the degree of multiprogramming in multitasking systems, following certain policies to decide whether the system can honour a new job submission or, if more than one job is submitted, which of them should be selected.

Medium term scheduling is essentially concerned with memory management, hence it's very often designed as a part of the memory management subsystem of an OS. Its efficient interaction with the short term scheduler is essential for system performances, especially in virtual memory systems. This is the reason why in paged system the pager process is usually run at a very high (dispatching) priority level.

(C)What is deadlock?

Answer :
A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. In other words, each member of the set of deadlock processes is waiting for a resource that can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened. It is important to note that the number of processes and the number and kind of resources possessed and requested are unimportant.
The resources may be either physical or logical. Examples of physical resources are Printers, Tape Drivers, Memory Space, and CPU Cycles. Examples of logical resources are Files, Semaphores, and Monitors.
Deadlock in an operating system means that two or more operation are requesting time from the CPU for completing their operation. But due to operational constraint the CPU is in a confused state and so it cannot give time to any of the operation. So the time period of the operation goes up. This is due to overloading the system.
In real life situation this can be compared to a traffic jam during peak hour. Every vehicle wants to go on without any delay. But the traffic signal puts some time constraint for each side of the junction. When a vehicle gets stuck in the middle of the road when it should not be there it cause the other vehicles not to be able to move... This cause a deadlock at the junction.

(D)What is disk scheduling?
Answer :
As we know that on a single Computer we can Perform Many Operations at a Time so that Management is also necessary on all the Running Processes those are running on the System at a Time. With the help or Advent of the Multi-programming we can Execute Many Programs at a Time. So fir Controlling and providing the Memory to all the Processes Operating System uses the Concept of Disk Scheduling.
In this the Time of CPU is divided into the various Processes and also determines that all the processes will Work Properly. So that Disk Scheduling Will Specifies that at which time which Process will be executed by the CPU. So that the Scheduling means to Execute all the Processes those are given to a CPU at a Time. The Scheduling is used for Divide the Total Time of the CPU between the Number or Processes So that the Processes can execute Concurrently at a Single Time. For

Sharing the Time or For Dividing the Total Time of the CPU, the CPU uses the following the Scheduling Techniques.
1) **FCFC or First Come First Serve:** In this Jobs or Processes are Executed in the Manner in which they are entered into the Computer. In this Operating System Creates a Queue which contains the Sequence Order in which they are to be Executed and the Sequence in which the CPU will Execute the Process.. In this all the Jobs are performed according to their Sequence Order as they have entered. In this the Job which had Requested first will firstly performed by the CPU. And the Jobs those are entered Later will be Executed in to their Entering Order.

1) **SSTF or Shortest Seek Time First :-** in this Technique The Operating System will Search for the Shortest time means this will search which job will takes a Less Time of CPU for Running. And After Examining all the jobs, all the Jobs are arranged in the Sequence wise or they are Organized into the Priority Order. The Priority of the Process will be the Total Time which a Process will use For Execution. The Shortest Seek Time Will Include all the Time means Time to Enter and Time to Completion of the Process. Means the Total Time which a Process Will Take For Execution.

(E)What is file allocation table?
Answer :
File Allocation Table (FAT) is a file system that was created by Microsoft in 1977. FAT is still in use today as the preferred file system for floppy drive media and portable, high capacity storage devices like flash drives.
FAT was the primary file system used in all of Microsoft's consumer operating systems from MS-DOS through Windows ME. NTFS is the primary file system on Microsoft's newer operating systems but FAT is still a supported option.
The File Allocation Table file system has seen advancements over time primarily due to the need to support larger hard disk drives and larger file sizes.
Below is more information on the versions of the FAT file system:
FAT12 - The initial version of the FAT file system, FAT12 was introduced in 1977, even before MS-DOS, and was the primary file system for Microsoft operating systems up to MS-DOS 4.0. FAT12 supports drive sizes up to 32MB.
FAT16 - The second implementation of FAT was FAT16, introduced in 1988. FAT16 was the primary file system for MS-DOS 4.0 up to Windows 95. FAT16 supports drive sizes up to 2GB.

FAT32 - FAT32 is the latest version of the FAT file system. It was introduced in 1996 for Windows 95 OSR2 users and was the primary file system for consumer Windows versions through Windows ME. FAT32 supports drive sizes up to 8TB.

(F)What is virtual memory?

Answer :

Virtual memory is a feature of an operating system that enables a process to use a memory (RAM) address space that is independent of other processes running in the same system, and use a space that is larger than the actual amount of RAM present, temporarily relegating some contents from RAM to a disk, with little or no overhead.

In a system using virtual memory, the physical memory is divided into equally-sized pages. The memory addressed by a process is also divided into logical pages of the same size. When a process references a memory address, the memory manager fetches from disk the page that includes the referenced address, and places it in a vacant physical page in the RAM. Subsequent references within that logical page are routed to the physical page. When the process references an address from another logical page, it too is fetched into a vacant physical page and becomes the target of subsequent similar references.

(G)What is swapping?

Answer :

When the physical memory in the system runs out and a process needs to bring a page into memory then the operating system must decide what to do. It must fairly share the physical pages in the system between the processes running in the system, therefore it may need to remove one or more pages from the system to make room for the new page to be brought into memory. How virtual pages are selected for removal from physical memory affects the efficiency of the system. Linux uses a page aging technique to fairly choose pages which might be removed from the system. This scheme involves every page in the system having an age which changes as the page is accessed. The more that a page is accessed, the younger it is; the less that it is accessed the older it becomes. Old pages are good candidates for swapping.

Assignment

Q Explain various directory structure used in operating systemfor storing files give its merits and demerits?

Directories are basically symbol tables of files. A single fait directory can contain a list of all files in a system. A directory contains information about the files, including attributes, location and ownership. Operating system is managed this information. The directory is itself a file, owned by the operating system and accessible by various file management routines. Thus user can not directly access the directory, even in read only mode. Fig.l shows the typical file system organiation.

The simplest form of structure for a directory is that of a list of entries, one for each file. Directory entries are added by the creation of files and of aliases to existing files. Entries are removed from the directories when files are deleted. Directories are frequently searched to locate and access files for processing and to add or to remove entries.

To understand the requirements for a file structure, it is well to consider the type of operations that may be performed on the directory.

1. Search 3. Delete file 5. List directory
2. Create a file 4. Rename a file
    1. Search: Directory structure is searched for finding particular file in the directory. Files have symbolic names and similar names may indicate a relationship between files.
    2. Create a file: When a new file is created, an entry must be added to the directory.
    3. Delete a file: When a file is deleted, an entry must be removed from the directory.
    4. Rename a file: Name of the files must be changeable when the content or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
    5. List directory: Al! or portion of the directory may be requested. Request is made by a user and result in a listing of all files owned by that user plus some of the attributes of each file.

Different types of directory structures are given below.

1. Single level directory 4. Acyclic graph directories
2. Two level directory 5. General graph directory
3. Tree structured directory

Single level directory: Single level directory structure is simple directory structure. All files are contained in the same directory. Fig. shows single level directory structure. Easy to implement and maintain.

Disadvantages of single level directory are as follows:

1. Not suitable for a large number of files and more than one user.
2. Because of single directory, files require unique file name.
3. It is difficult to remember the names of all the files as the number of files increases.

MS-DOS operating system allows only 11 character file names whereas Unix allows 255 characters.

Two level directory: In two level directory, each user has his own directory. It is called user file directory (UFD). Each user file directory has a similar structure. Fig.3 shows the two level directory. When a user refers to a particular file, only his own UFD is searched. Different users may have files with the same name, as long as all the file names within each UFD are unique.

To create a file for a user, the operating system searches only that users directory to ascertain whether another file of that name exists. To delete a file, the operating system confines the search to the local UFD.

Operating system can not accidently delete another users file that has the same name.

The structured directories: MS-DOS system is a tree structure directory. It allows users to create their own subdirectory and to organize their files accordingly. A subdirectory contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.

All the directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1). Special system calls are used to create and delete directories.

In normal use, each user has a current directory. Current directory should contain most of the files that are of current interest to the user. When a reference is made to a file, the current directory is searched. Path name is used to search or for any operation on file with another directory. Path names can be of two types:

a) Absolute path name b) Relative path name

a) Absolute path name begins at the root and follows a path down to the specified file, giving the director)' names on the path.

b) Relative path name defines a path from the current directory. MS- DOS will not delete a directory unless it is empty. For deleting a directory, two approaches can be taken.

1. User must delete all the files from the directory. Make it empty directory.
2. In Unix, rm command is used with some option for deleting directory.

When a request is made to delete a directory, all that directory files and subdirectories are also to be deleted. A path to a file in a tree structured directory can be longer than that in a two level directory.

Advantages;

1. It allows users to create their own subdirectory.
2. User can access the files of other users.
3. It allows users to define their own search paths.

Disadvantages:

1. Special system calls are required to cerate and delete directories.
2. It prohibits the sharing of files and directories.
3. Path to the file is longer than the two level directory. Acyclic graph directories

Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure. (Note the ***directed*** arcs from parent to child.)
- UNIX provides two types of ***links*** for implementing the acyclic-graph structure. (See "man ln" for more details.)
    - A ***hard link*** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
    - A ***symbolic link***, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.
- Windows only supports symbolic links, termed ***shortcuts.***
- Hard links require a ***reference count***, or ***link count*** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.
- For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:
    - One option is to find all the symbolic links and adjust them also.
    - Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
    - What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?

Advantages:

1. 1. An acyclic graph allows directories to have shared subdirectories and files.
2. Structure is more flexible than a simple tree structure.

3. Simple for traverse a file.

Disadvantages:

1. Deletion of file is difficult.

2. A major problem with duplicate directory entries is maintaining consistency of file is modified.

General Graph Directory

- If cycles are allowed in the graphs, then several problems can arise:
    - Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.)
    - Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem. (chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.)

Q:6 (a) How Files System are organized with UNIX ? Explain with an example .

Sol:-Files are the heart of the Unix system. The Unix system knows about three kinds of files: ordinary files, directory files and special files. The programs you create, the text you edit, and all the Unix commands are stored in ordinary files (which we will simply call files). A file is a sequence of bytes. No structure is imposed on a file by the Unix system, and no meaning is attached to its contents. The meaning of the bytes depends solely on the programs that interpret the file.

All files in Unix are organised into directories. A directory is a file containing information about other files and directories; we might think of a directory as a thing which `contains' files and other directories.

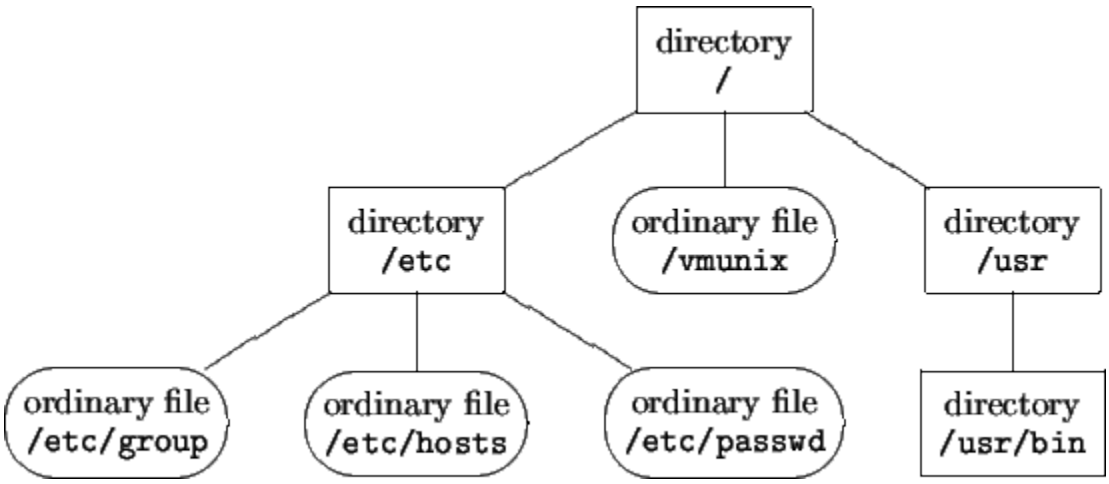


Figure 1: An Hierarchy of Directories and Ordinary Files

When you are given an account on the system, you are assigned a `home directory'. Every time you login you will find yourself positioned in this directory.

Files in the Unix system are grouped into directories and the directories are organised into a hierarchy or tree. The top of the hierarchy is a special directory called the root directory. The root directory contains a variety of system-related files and it usually contains some standard directories such as /bin, /usr, /dev, /etc, and /lib.

Filenames

Every file has a filename. A filename can be composed of any character except `/' and the null character. You will, however, avoid confusion if you choose characters from the following list.

- uppercase letters [A-Z]
- lowercase letters [a-z]
- numbers [0-9]
- underscore [_]

- period [.]
- comma [,]

The only exception is the root directory, which is always named and referred to by `/'. No other file can use this name.

Like children of one parent, no two files in the same directory can have the same name. Files in different directories, like children of different parents, can have the same name.

6(b) what is a Deadlock? How it is detected? What are the necessary conditions for a deadlock to occur?

Sol: It a state where two or more operations are waiting for each other, say a computing action 'A' is waiting for action 'B' to complete, while action 'B' can only execute when 'A' is completed. Such a situation would be called a deadlock. In operating systems, a deadlock situation is arrived when computer resources required for complete of a computing task are held by another task that is waiting to execute. The system thus goes into an indefinite loop resulting into a deadlock.

The deadlock in operating system seems to be a common issue in multiprocessor systems, parallel and distributed computing setups.

Deadlock detection is performed by a lock monitor thread that periodically initiates a search through all of the tasks in an instance of the Database Engine. The following points describe the search process:

- The default interval is 5 seconds.
- If the lock monitor thread finds deadlocks, the deadlock detection interval will drop from 5 seconds to as low as 100 milliseconds depending on the frequency of deadlocks.
- If the lock monitor thread stops finding deadlocks, the Database Engine increases the intervals between searches to 5 seconds.
- If a deadlock has just been detected, it is assumed that the next threads that must wait for a lock are entering the deadlock cycle. The first couple of lock waits after a deadlock has been detected will immediately trigger a deadlock search rather than wait for the next deadlock detection interval. For example, if the current interval is 5 seconds, and a deadlock was just detected, the next lock wait will kick off the deadlock detector immediately. If this lock wait is part of a deadlock, it will be detected right away rather than during next deadlock search.

Deadlock Conditions

1. **Mutual exclusion**
   The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.
2. **Hold and wait or partial allocation**
   The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.
3. **No pre-emption**
   The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.

4. **Resource waiting or circular wait**
   A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

Assignment

Q1) Explain the following:

- **Multitasking:**

  multitasking is a method where multiple tasks, also known as processes, are performed during the same period of time. The tasks share common processing resources, such as a CPU and main memory.

  In the case of a computer with a single CPU, only one task is said to be running at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves the problem by scheduling which task may be the one running at any given time, and when another waiting task gets a turn.

- **Real time operating system:**

  A real-time operating system is an operating system intended to serve real-time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter.

  A key characteristic of an Real Time Operating System is the level of its consistency concerning the amount of time it takes to accept and complete an application's task.

  Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency; a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

- **Distributed system:**

  A distributed system is a collection of physically separate , possibly heterogeneous, computer systems that are networked to provide the users with access to the various resources that the system maintains. Access to a shared resource increases computation speed, functionality, data availability and reliability.

  Distributed systems depend on networking for their functionality.

Q2) A variable portion memory system has at some point in time the following box sizes in the order 20k,15k,40k,60k,10k,25k, a new process is to be loaded which block will be filled using best fit, first fit, worst fit respectively.

Ans:

In case of best fit, process would be accommodated in a block which is sufficient enough to hold the process and left over space is minimum.

In case of first fit whichever block is sufficient enough to hold the process would be accommodated in a block which would be sufficient enough to hold the process.

In case of worst fit the leftover case is maximum.

OPERATING SYSTEM

ASSIGNMENT (YEAR – 2010)

Ques4 (a) Describe the shortest–job-first scheduling algorithm.

(b) What are the mechanisms to evaluate an algorithm related to CPU scheduling? Discuss any one of them.

ANS. (a) **Shortest-job-first scheduling:** This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is free, it is assigned to the process of the ready queue which has smallest next CPU burst. If two processes have the same length, FCFS scheduling is used to break the tie. SJF scheduling algorithm is used frequently in long term scheduling. SJF algorithm may be either pre-emptive or non-pre-emptive.

A pre-emptive SJF algorithm will pre-empt the currently executing process whereas a non-pre-emptive SJF algorithm will allow the currently running process to finish its CPU burst. Let us consider the set of process with burst time in milliseconds.

SJF algorithm cannot be implemented at the level of short term CPU scheduling. There is no way to know the length of the next CPU burst

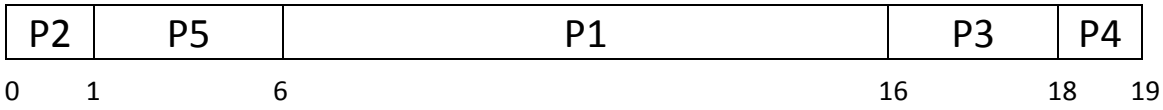
(b) There are several mechanisms to evaluate an algorithm related to CPU scheduling.

1. First-come first-serve scheduling,FCFS
2. Shortest-job-first scheduling, SJF
3. Priority scheduling
4. Round robin scheduling
5. Multilevel queue scheduling
6. Multilevel feedback-queue scheduling

Priority scheduling

- Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first(SJF uses the inverse of the next expected burst time as its priority- the smaller the expected burst, the higher the priority.)
- Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether “high” priorities use large numbers or small numbers.
- For example, the following Gantt chart is based upon these process burst times and priorities, and yields an average waiting time of 8.2ms:

PROCESS	BURST TIME	PRIORITY
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



- Priorities can be assigned either internally or externally. Internally priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.

- Priority scheduling can be either pre-emptive or non-pre-emptive.
- Priority scheduling can suffer from a major problem known as indefinite blocking, or starvation, in which a low priority task can wait forever because there are always some other jobs around that have higher priority.
    - If this problem is allowed to occur, then processs will either run eventually when the system load lightens, or will eventually get lost when the systems is shut down or crashes.
    - Other common solution to this problem is aging, in which priorities of jobs increase the longer they wait. Under this scheme a low priority job will eventually get its priority raised high enough that it gets run.

Ques5. (a) Describe the dining-philosopher's problem and provide its solution

(b) How will you handle synchronization problem using hardware? Discuss.

ANS. (a) There is *N* philosopher's sitting around circular table eating spaghetti and discussing philosophy. The problem is that each philosopher needs 2 forks to eat, and there are only *N* forks, one Between each 2 philosophers. Design an algorithm that the philosophers can follow that insures that none starves as long as each philosopher eventually stops eating, and such that the maximum number of philosophers can eat at once.

Why describe problems this way? Well, the analogous situations in computers are sometimes so technical that they obscure creative thought. Thinking about philosophers makes it easier to think

abstractly. And many of the early students of this field were theoreticians who like abstract problems. There are a bunch of named problems - Dining Philosophers

(b)

OPERATING SYSTEM

ASSINGMENT

Name: Rohit Garg
Enrollment No. : 02413702011

Ques: Discuss the structure of directory and its implementation in detail.

Ans: A directory structure is the operating system's file system and its files are displayed to the user.

File names and extensions
A filename is a string used to uniquely identify a file stored on the file system of a computer. Before the advent of 32-bit operating systems, file names were typically limited to short names (6 to 14 characters in size). Modern operating systems now typically allow much longer filenames.

Windows, DOS
In DOS, Windows the root directory is "*drive*:\", for example, the root directory is usually "C:\". The directory separator is usually a "\", but the operating system also internally recognizes a "/". Physical and virtual drives are named by a drive letter, as opposed to being combined as one.[1] This means that there is no "formal" root directory, but rather that there are independent root directories on each drive. However, it is possible to

combine two drives into one virtual drive letter, by setting a hard drive into a RAID setting of 0.

Common Windows directory structures:-

$Recycle.Bin
Recycle folder (hidden)

Boot
Boot folder (hidden, since Windows Vista)

Documents and Settings
User folders (up to Windows XP, legacy and hidden since Windows Vista)

Inetpub
IIS folder (if installed)

Perflogs
Created by Windows Performance Information and Tools

Program Data
Program data (hidden, since Windows Vista)

Directory Implementation

Linear list of file names with pointer to the data blocks.
1). Simple to program.
2). Time-consuming to execute due to linear search.

Hash Table – linear list with hash data structure.

1). decreases directory search time
2). collisions– situations where two file names hash to the same location
3).Fixed Size.

Ques: What are the various free space management techniques? Discuss.

Ans: Various free space management techniques are:-

1). Bits Vector

This method uses a vector containing one bit for each block on the disk.
Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use.

2). **Chained free portions**

Free portions are linked.

Fragmentation if using variable allocation.

Required read before write to a block.

3). **Free block list**

Single list of a set of free block lists (unallocated blocks)

Manage as LIFO or FIFO on disk with ends in main memory

Background jobs can reorder list for better contiguity

Operating System

Question Paper: 2006

Q 7

A) What are the various functions of KERNEL of UNIX?

In computer science, the **kernel** is the central component of most computer operating system(OS). Its responsibilities include managing the system's resources (the communication between hardware and software components). As a basic component of an operating system, a kernel provides the lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that application software must control to perform its function. It typically makes these facilities available to application programs through interprocess communication mechanisms and system calls.
These tasks are done differently by different kernels, depending on their design and implementation. While monolithic kernels will try to achieve these goals by executing all the code in the same address space to increase the performance of the system, microkernels run most of their services in user space, aiming to improve maintainability and modularity of the codebase. A range of possibilities exists between these two extremes.

B) What is the critical section problem? What are its various solutions?

Critical Sections

- Mutual Exclusion is the way of avoiding the race conditions
- If one process is using a shared variable or file, the other processes will be excluded from doing the same thing
- Sometimes, a process is busy doing internal computations and other things that do not lead to race conditions
- However, sometimes a process may be accessing shared memory or files that lead to races
- The part of the program where the shared memory is accessed is called the critical region or critical section
- If no two processes are in the critical regions at a same time, there will be no race condition
- Although this is enough for avoiding race conditions, this is not sufficient for having parallel processes cooperate correctly and efficiently using shared data

There are four conditions that need to be satisfied to have a good solution
o No two processes may be simultaneously inside their critical regions
o No assumptions may be made about speeds or the number of CPUs
o No process running outside its critical region may block other processes
o No process should have to wait forever to enter its critical region

Operating System Assignment

Submitted By:
Mudit Narang
04313702011
BCA 5th Morning

Q] Explain the services provided by a Kernel I/O sub system.

Kernel provided many services related to I/O. The several services. (1) Scheduling (2) Buffering (3) Caching (4) Spooling

(5) Device reservation and

(6) Error handling – are provided by Lhe Kernel's I/O subsystems.

1) I/O scheduling: To schedule a set of-I/O requests means to determine a good order in which to execute them. The order in which applications issue system calls rarely is the best choice. Scheduling can improve overall system performance, can share device access fairly among processes and can reduce the average waiting time for I/O to complete.

Operating system developers implement scheduling by maintaining a queue of requests for each device. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The I/O sub system improves the efficiency of the computer is by scheduling I/O operations.

2) Buffering: A Buffer is a memory area that stores data while they are transferred between two devices (or) between a device and an applications. Buffering is done fore three reasons.

One reason is to cope with a speed mismatch between the producer and consumer of a data stream. For example that a file is being received via modem for storage on the hard disk. The modem is about a thousand times slower than the hard disk. So a buffer is created in main memory to accumulate the bytes received from the modem. When an entire buffer ot data has arrived, the buffer can be written to disk in a single operation.

A second use of buffering is to adapt between devices that have different data transfer sizes. Such disparities are especially common in computer networking. Where buffers are used widely for fragmentation and reassembly of messages. At the sending side, a large message is fragmented into small network packets. The packets are sent over the network, and the receiving side places them in a reassembly buffer to form an image of the source data.

A third use of buffering is to support copy semantics for application I/O. A example will clarify the meaning of copy semantics. Suppose that an application has a buffer of data that it wishes to write to disk.

3) 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. For instance 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. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, where as a cache, by definition, just holds a copy an faster storage of an item that resides else where.

4) Spooling and Device reservation: A spool is a buffer that holds output for a device, such as a printer that cannot accept inter level data streams. Although a printer can serve only one job at a time, several applications may wish to print their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. 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 flies to the printer one at a time.

5)Error Handling: An operating system that uses protected memory can guard against many kinds of hardware and application errors. So that a complete system failure is not the usual result of each minor mechanical glitch. Devices and I/O transfers can fail in many ways, either for transient reasons, such a network becoming overloaded (or) for 'permanent' reasons such as a disk controller becoming detective. Operating systems can often compensate effectively for transient failures. For instance, a disk read ( ) failure results in a read ( ) retry, and a network send ( ) error results in a resend ( ), if the protocol so specifies, unfortunately, if an important component experiences a permanent failure, the operating system is unlikely to recover.

Q] Discuss Disk Formatting.

Disk formatting is the process of preparing a data storage device such as a hard disk drive, solid-state drive, floppy disk or USB flash drive for initial use. In some cases, the formatting operation may also create one or more new file systems. The first part of the formatting process that performs basic medium preparation is often referred to as "low-level formatting". Partitioning is the common term for the second part of the process, making the data storage device visible to an operating system. The third part of the process, usually termed "high-level formatting" most often refers to the process of generating a new file system. In some operating systems all or parts of these three processes can be combined or repeated at different levels and the term "format" is understood to mean an operation in which a new disk medium is fully prepared to store files. Illustrated to the right are the prompts and diagnostics printed by MS-DOS's FORMAT.COM utility as a hard drive is being formatted.

Q] Describe Directory Structure of a file system.

A file system is a section of hard disk that has been allocated to contain files. This section of hard disk is accessed by mounting the file system over a directory. After the file system is mounted, it looks just like any other directory to the end user.

However, because of the structural differences between the file systems and directories, the data within these entities can be managed separately.

When the operating system is installed for the first time, it is loaded into a directory structure, as shown in the following illustration.

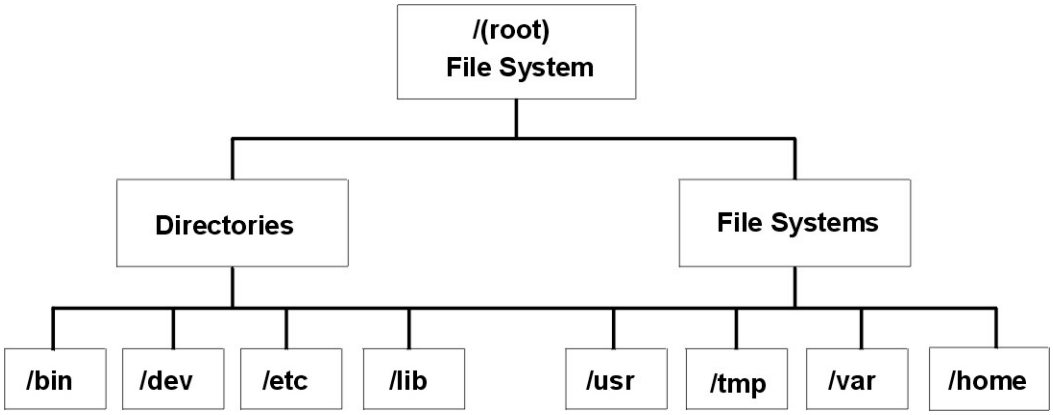


Figure 1. / (root) File System Tree. This tree chart shows a directory structure with the / (root) file system at the top, branching downward to directories and file systems. Directories branch to /bin, /dev, /etc, and /lib. File systems branch to /usr, /tmp, /var, and /home.

Operating System

Assignment

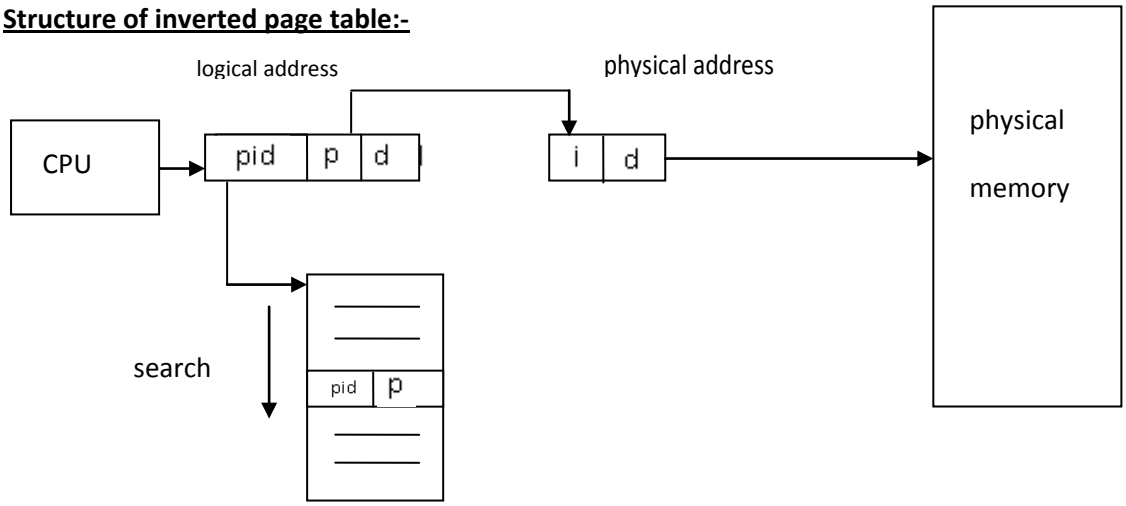
Q1) Explain the structure of inverted page table ?

Ans :

In inverted page table usually each process has an associated page table. The page table has one entry for each page that the process is using(or one slot for each virtual address , regardless of the latter's validity)

An inverted page table has one entry for each real page (or frame) of memory. Each entry consist of the virtual address of the page stored in that real memory location , with information about the process that owns the page. thus, only one page table is in the system , and it has only one entry for each page of physical memory.

Structure of inverted page table:-



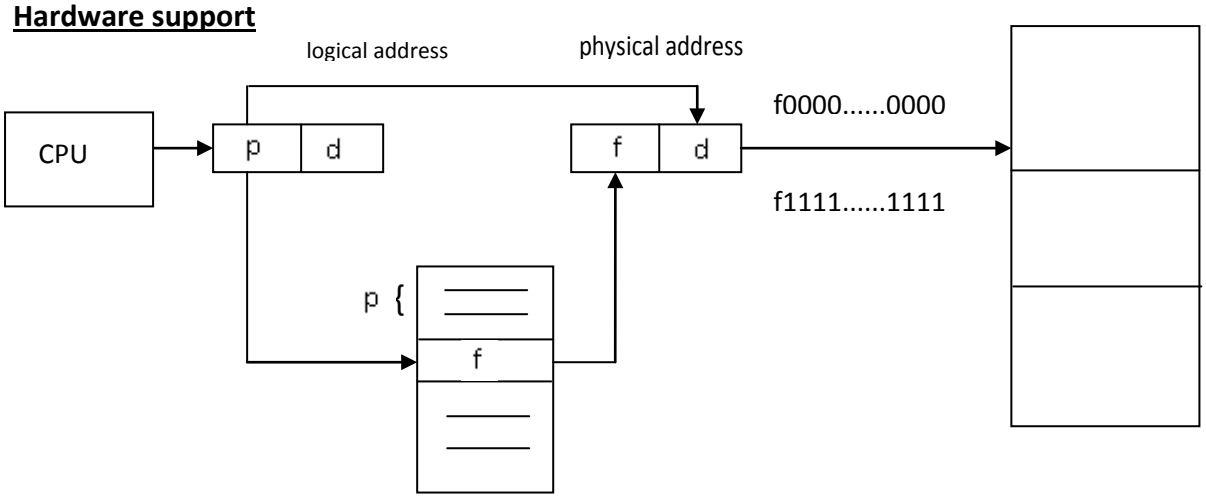
Q2) Explain the basic method of paging scheme. show the hardware support for it with a diagram ?

Ans:

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed , its pages are loaded into any available memory frames from their source(a file system or backing store). The backing store is divided into fixed-sized blocks that are of the as the memory frames.

Ishaan
07613702011
BCA (m) - V

Hardware support



Q3) Explain the concept of thrashing ?

Ans:

If the no of frames allocated to a low priority process falls below the minimum number required by the computer architecture ,we must spend that process's execution. we should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in , swap-out level of intermediate CPU scheduling.

If the process that does not have the number of frames it needs to support pages in active use , it will quickly page-fault. At this point, it must replace some page.

Since all its pages are in active use , it must replace a page that will be needed again right away. Consequently , it quickly faults again , and again , and again , replacing pages that it must bring back in immediately.

This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

Ishaan
07613702011
BCA (m) - V

OPERATING SYSTEM

ASSIGNMENT

Name: Anita Gupta

Rollno: 07813702011

Class: BCA 5Th Semester

Q-1(a)Describe the need for device management.

Answer: A process may need several resources to execute-main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files).A system with multiple users may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we are released it. These functions are similar to the open and close system calls for files. Other operating systems allow unmanaged process to devices.

(b) Explain how buffering is used with respect to storage devices.

Answer: A buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a microphone) or just before it is sent to an output device (such as speakers). However, a buffer may be used when moving data between processes within a computer. This is comparable to buffers in telecommunication. Buffers can be implemented in a fixed memory location in hardware—or by using a virtual data buffer in software, pointing at a location in the physical memory. In all cases, the data stored in a data buffer are stored on a physical storage medium. A majority of buffers are implemented in software, which typically use the faster RAM to store temporary data, due to the much faster access time compared with hard disk drives. Buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler or in online video streaming.

A buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

Applications: Buffers are often used in conjunction with I/O to hardware, such as disk drives, sending or receiving data to or from a network, or playing sound on a speaker. A line to a rollercoaster in an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded). The queue area acts as a buffer—a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a FIFO (first in, first out) method, outputting data in the order it arrived.

Q-2(a) Explain the different methods for allocating disk space to files.

Answer: Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupy a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.

- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

(b) Explain 'File Concept'. What are the different operation performed on files.

File Concept

Computers can store information on various storage media such as magnetic disks, magnetic tapes and optical disks. OS provides a uniform logical view of information storage. OS abstracts from the physical properties of its storage devices to define a logical storage unit called a file. Files are mapped by OS onto physical devices. These storage devices are non volatile so the contents are persistent through power failures and system reboots.

A file is a named collection of related information that is recorded on secondary storage. A file is the smallest allotment of logical secondary storage; that is data cannot be written to secondary storage unless they are within a file. Files represent programs and data. Data files may be numeric, alphabetic, alphanumeric or binary. Files may be free form such as text files or may be formatted rigidly. A file is a sequence of bits, bytes, lines or records.

There are different operation performed on files are as-:

- A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files.
- Six basic file operations. The OS can provide system calls to create, write, read, reposition, delete, and truncate files.
    - Creating a file. Two steps are necessary to create a file.
        1. Space in the file system must be found for the file.
        2. An entry for the new file must be made in the directory.
    - Writing a file. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
    - Reading a file. To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file

should be put. The system needs to keep a read pointer to the location in the file where the next read is to take place.

- Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current-file-position pointer.
- Both the read and write operations use this same pointer, saving space and reducing system complexity.

- Repositioning within a file. The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.
- Deleting a file. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- Truncating a file. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged (except for file length) but lets the file be reset to length zero and its file space released.

Q. Discuss banker's algorithm in detail. Also provide a example for the same

The bankers algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes a "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. The Banker's algorithm is run by the operating system whenever a process requests resources.[2] The algorithm avoidsdeadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur). When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

For the Banker's algorithm to work, it needs to know three things:

- How much of each resource each process could possibly request[CLAIMS]
- How much of each resource each process is currently holding[ALLOCATED]
- How much of each resource the system currently has available[AVAILABLE]

Resources may be allocated to a process only if it satisfies the following conditions:

1. request ≤ max, else set error condition as process has crossed maximum claim made by it.
2. request ≤ available, else process waits until resources are available.

Some of the resources that are tracked in real systems
are memory, semaphores and interface access.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.

- Max: An n×m matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- Allocation: An n×m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instance of resource type R_j.
- Need: An n×m matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j].

```
/*PROGRAM TO IMPLEMENT BANKER'S ALGORITHM
*   ----------------------------------------*/
#include <stdio.h>
int curr[5][5], maxclaim[5][5], avl[5];
int alloc[5] = {0,0,0,0,0};
int maxres[5], running[5], safe=0;
int count = 0, i, j, exec, r, p;

int main()
{
printf("\nEnter the number of processes: ");
scanf("%d",&p);

for(i=0;i<p;i++)
{
running[i]=1;
count++;
}

printf("\nEnter the number of resources: ");
scanf("%d",&r);

for(i=1;i<=r;i++)
{
printf("\nEnter the resource for instance :%d: ",i);
scanf("%d",&maxres[i]);
}

printf("\nEnter maximum resource table:\n");
for(i=0;i<p;i++)
```

```
{
for(j=0;j<r;j++)
{
scanf("%d",&maxclaim[i][j]);
}
}

printf("\nEnter allocated resource table:\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&curr[i][j]);
}
}

printf("\nThe resource of instances: ");
for(i=0;i<r;i++)
{
printf("\t%d",maxres[i]);
}

printf("\nThe allocated resource table:\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t%d",curr[i][j]);
}

printf("\n");
}

printf("\nThe maximum resource table:\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t%d",maxclaim[i][j]);
}
```

```
printf("\n");
}

for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
alloc[j]+=curr[i][j];
}
}

printf("\nAllocated resources:");
for(i=0;i<r;i++)
{
printf("\t%d",alloc[i]);
}

for(i=0;i<r;i++)
{
avl[i]=maxres[i]-alloc[i];
}

printf("\nAvailable resources:");
for(i=0;i<r;i++)
{
printf("\t%d",avl[i]);
}
printf("\n");

//Main procedure goes below to check for unsafe state.
while(count!=0)
{
safe=0;
for(i=0;i<p;i++)
{
if(running[i])
{
exec=1;
for(j=0;j<r;j++)
```

```
{
if(maxclaim[i][j] - curr[i][j] > avl[j]){
exec=0;
break;
}
}
if(exec)
{
printf("\nProcess%d is executing\n",i+1);
running[i]=0;
count--;
safe=1;

for(j=0;j<r;j++) {
avl[j]+=curr[i][j];
}

break;
}
}
}
if(!safe)
{
printf("\nThe processes are in unsafe state.\n");
break;
}
else
{
printf("\nThe process is in safe state");
printf("\nSafe sequence is:");

for(i=0;i<r;i++)
{
printf("\t%d",avl[i]);
}

printf("\n");
}
}
}
```

/*SAMPLE OUTPUT

Enter the number of resources:4

Enter the number of processes:5

Enter Claim Vector:8 5 9 7

Enter Allocated Resource Table:
2 0 1 1
0 1 2 1
4 0 0 3
0 2 1 0
1 0 3 0

Enter Maximum Claim table:
3 2 1 4
0 2 5 2
5 1 0 5
1 5 3 0
3 0 3 3

The Claim Vector is: 8 5 9 7
The Allocated Resource Table:
2 0 1 1
0 1 2 1
4 0 0 3
0 2 1 0
1 0 3 0

The Maximum Claim Table:
3 2 1 4
0 2 5 2
5 1 0 5
1 5 3 0
3 0 3 3

Allocated resources: 7 3 7 5

```
Available resources:   1      2      2      2

Process3 is executing

The process is in safe state
Available vector:      5      2      2      5
Process1 is executing

The process is in safe state
Available vector:      7      2      3      6
Process2 is executing

The process is in safe state
Available vector:      7      3      5      7
Process4 is executing

The process is in safe state
Available vector:      7      5      6      7
Process5 is executing

The process is in safe state
Available vector:      8      5      9      7

----------------------------------------------------------*/
```

Limitations

Like other algorithms, the Banker's algorithm has some limitations when implemented. Specifically, it needs to know how much of each resource a process could possibly request. In most systems, this information is unavailable, making it impossible to implement the Banker's algorithm. Also, it is unrealistic to assume that the number of processes is static since in most systems the number of processes varies dynamically. Moreover, the requirement that a process will eventually release all its resources (when the process terminates) is sufficient for the correctness of the algorithm, however it is not sufficient for a practical system. Waiting for hours (or even days) for resources to be released is usually not acceptable.

Q. **Explain following terms.**

- **Dedicated device**
- **Channel**
- **Control unit**

1) Dedicated device

– Dedicated devices are devices that are assigned to only 1 job at a time and serves the job for the entire time it is active. eg. Printers, tape devices, plotters Dedicated devices are assigned to only one job at a time. Examples may include tape drives, printers, and plotters. The disadvantage of dedicated devices is they must be allocated to a single user for the duration of a job's execution. Can be quite inefficient, especially when device isn't used 100 % of time.

2) Channels

Channel architecture avoids this problem by using a separate, independent, low-cost processor. Channel processors are simple, but self-contained, with minimal logic and sufficient on-board scratchpad memory (working storage) to handle I/O tasks. They are typically not powerful or flexible enough to be used as a computer on their own and can be construed as a form of coprocessor.

A CPU sends relatively small channel programs to the controller via the channel to handle I/O tasks, which the channel and controller can, in many cases, complete without further intervention from the CPU (exception: those channel programs which utilize 'program controlled interrupts', PCIs, to facilitate program loading, demand paging and other essential system tasks).

3) Control Unit

The control unit is a component of a computer's central processing unit (CPU) which directs operation of the processor. It controls communication and co-ordination between input/output devices. It reads and interprets instructions and determine the sequence for processing the data.It directs the operation of the other units by providing timing and control signals. All computer resources are managed by the CU (Control Unit). It directs the flow of data between the Central Processing Unit (CPU) and the other devices The control unit was historically defined as one distinct part of the 1946 reference model of Von

Neumann architecture. In modern computer designs, the control unit is typically an internal part of the CPU with its overall role and operation unchanged.

The control unit implements the instruction set of the CPU. It performs the tasks of fetching, decoding, managing execution and, finally, storing results. The CU may manage the translation of instructions (not data) to micro-instructions and manage scheduling the micro-instructions between the various execution units. On some processors, the control unit may be further broken down into other units, such as a scheduling unit to handle scheduling or a retirement unit to deal with results coming from the pipeline. The control unit handles the main functions of the CPU.

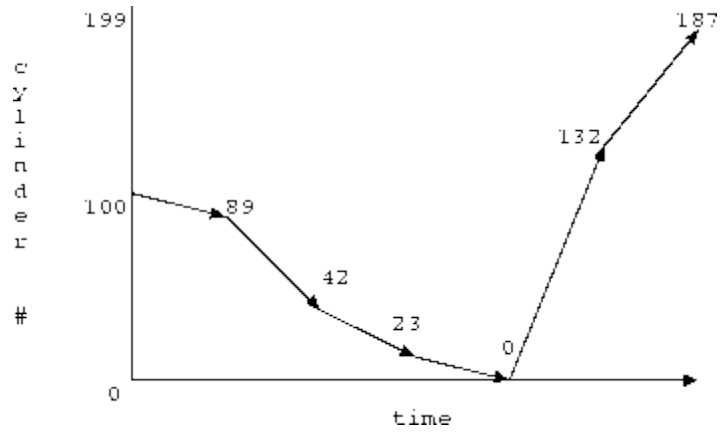
Q. Discuss the following disk scheduling algorithm with eg.

- **Scan**
- **Look**

Scan (C-SCAN)

Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.
Eg:

- assume we are going inwards (i.e., towards 0)

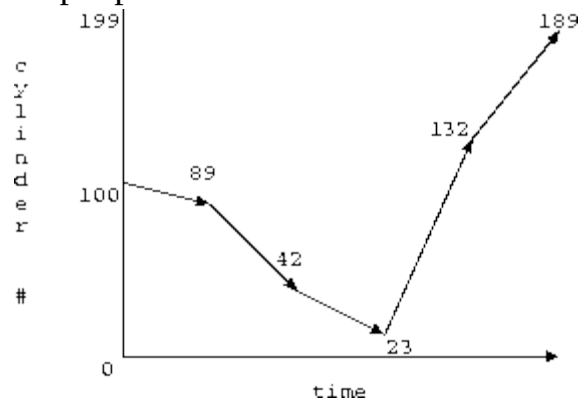


$$| 100 - 89 | + | 89 - 42 | + | 42 - 23 | + | 23 - 0 | + | 0 - 132 | + | 132 - 187 |= 11 + 47 + 19 + 23 + 1$$

Look

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.



$$| 100 - 89 | + | 89 - 42 | + | 42 - 23 | + | 23 - 132 | + | 132 - 187 |= 11 + 47 + 19 + 109 + 55 =$$

- reduce variance compared to SCAN

Examination Paper 2009

Ques. 5(a) Explain Semaphores?

Ans. A semaphore is a variable or abstract data type that is used for controlling access, by multiple processes, to a common resource in a parallel programming or a multi user environment. A useful way to think of a semaphore is as a record of how many units of a particular resource are available, coupled with operations to safely (i.e., without race conditions) adjust that record as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores.

Ques. 6(b) What is Swap Space Management?

Ans. Moving entire processes between disk and main memory. Swapping in that setting occurs when the amount of physical memory reaches a critically low point and processes (which are usually selected because they are the least active) are moved from memory to swap space to free available memory. In practice, very few modern operating systems implement swapping in this fashion. Rather, systems now combine swapping with virtual memory techniques and swap pages, not necessarily entire processes. In fact, some systems now use the terms swapping and paging interchangeably, reflecting the merging of these two concepts. 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. In this section, we discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

Ques. 7(a) What is Deadlock Detection? Explain how to recover from deadlock?

Ans. A simple way to detect a state of deadlock is with the help of wait-for graph. This graph is constructed and maintained by the system. One node is created in the wait-for graph for each transaction that is currently executing. Whenever a transaction Ti is waiting to lock an item X that is currently locked by a transaction Tj, a directed edge (Ti->Tj).is created in the wait-for graph. When Tj releases the lock(s) on the items that Ti was waiting for, the directed edge is dropped from the wait-for graph. We have a state of deadlock if and only if the wait-for graph has a cycle. Then each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait for graph, and periodically to invoke an algorithm that searches for a cycle in the graph. If deadlocks occur frequently, then the detection

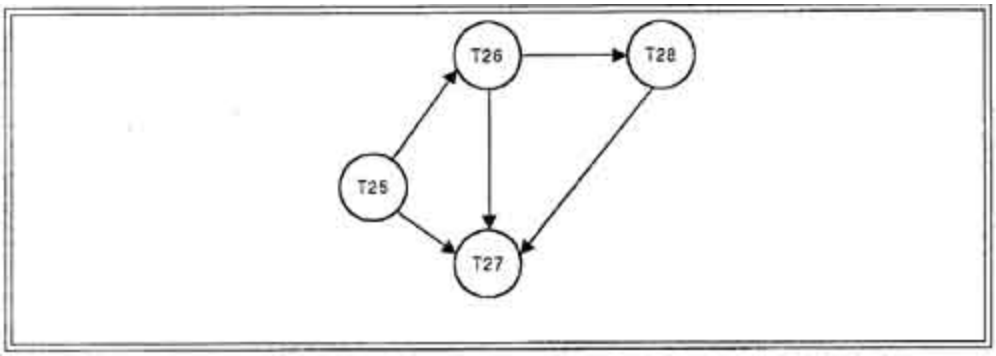
algorithm should be invoked more frequently than usual. Data items allocated to deadlocked transactions will be unavailable to other transactions until the deadlock can be broken. In the worst case, we would invoke the detection algorithm every time a request for allocation could not be granted immediately.

When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock. Choosing which transaction to abort is known as Victim Selection.

Choice of Deadlock victim

In below wait-for graph transactions T26, T28 and T27 are deadlocked. In order to remove deadlock one of the transaction out of these three transactions must be roll backed.

We should roll back those transactions that will incur the minimum cost. When a deadlock is detected, the choice of which transaction to abort can be made using following criteria:



• The transaction which have the fewest locks

• The transaction that has done the least work

• The transaction that is farthest from completion

Rollback

Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a total rollback; Abort the transaction and then restart it. However it is more effective to roll back the transaction only as far as necessary to break the deadlock. But this method requires the system to maintain additional information about the state of all the running system.

Problem of Starvation

In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result this transaction never completes

can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

Detection versus Prevention

In prevention-based schemes, the abort mechanism is used preemptively in order to avoid deadlocks. On the other hand in detection-based schemes, the transactions in a deadlock cycle hold locks that prevent other transactions from making progress. System put is reduced in case of detection because many transactions may be blocked, waiting to obtain locks currently held by deadlocked transactions.

This is the fundamental trade-off between these prevention and detection approaches to deadlocks: loss of work due to preemptive aborts versus loss of work due to blocked transactions in a deadlock cycle. We can increase the frequency with which we check for deadlock cycles, and thereby reduce the amount of work lost due to blocked transactions, but this entails a corresponding increase in the cost of the deadlock detection mechanism.

Deadlock detection is preferable, if different transactions rarely access the same items the same time. This can happen if the transactions are short and each transaction locks only few items or if the transaction load is light.

On the other hand, if transactions are long and each transaction uses many items, or if the transaction load is quite heavy, it may advantageous to use deadlock prevention scheme.

OPERATING SYSTEM

ASSIGNMENT-2

Q1.(a) What is boot strapping?

Ans-1(a) Booting is the process of starting a computer, specifically in regards to starting its software. The process involves a chain of stages, in which at each stage a smaller simpler program loads and then executes the larger more complicated program of the next stage. It is in this sense that the computer "pulls itself up by its bootstraps", i.e. it improves itself by its own efforts. Booting is a chain of events that starts with execution of hardware-based procedures and may then hand-off to firmware and software which is loaded into main memory. Booting often involves processes such as performing self-tests, loading configuration settings, loading a BIOS, resident monitors, a hypervisor, an operating system, or utility software. pressing a bootstrap button caused a hardwired program to read a bootstrap program from an input unit. The computer would then execute the bootstrap program, which caused it to read more program instructions. It became a self-sustaining process that proceeded without external help from manually entered instructions.

(b) Write four optimizing criteria for CPU scheduling.

1.CPU utilization – keep the CPU as busy as possible

2.Throughput – throughput of processes that complete their execution per time unit

3.Turnaround time – amount of time to execute a particular process

4.Waiting time – amount of time a process has been waiting in the ready queue

5.Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

(c) Explain the difference between spooling and buffering.

Spooling and buffering are both forms of storing data temporarily on a computer. Spooling can be carried out online or offline, while buffering requires an Internet connection. The main difference between spooling and buffering is that the former makes use of a computer and input/output devices simultaneously whereas the latter only uses the computer.

(d) Describe the main features of graphical user interface.

graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators. Its features are :

- Highly visual and dynamic GUI. Any number of windows can be shown at once, all of which are automatically updated (and the file re-run) whenever data changes
- Junction Diagram – shows the junction in graphical format and includes queue animation, signal states, phases, stages, and results such as delay for each stream. The junction diagram allows access to all data for traffic streams and phases etc, and enables a new junction to be built graphically
- Phase timings diagram – shows phase and stage timings graphically, along with phase delays.
- Graph Generator – Allows any result (effective green time, delay, etc) to be plotted against any input parameter (cycle time, sat flow, etc). This serves as a very flexible sensitivity analysis tool
- Report Generator – reports are generated in HTML, in a format that can easily be pasted into a word processor or spreadsheet. The main diagrams are inserted in vector format, meaning that print-outs are very high quality

Sakshi Dewangan
06813702011
BCA(M)-V

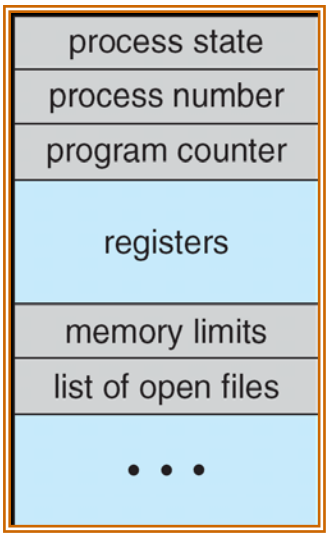
- Undo/redo allowing stepping back through up to 100 changes
- Quick edit mode – allows main data to be edited in a spreadsheet format
- All data can be accessed via a 'tree view' system and is therefore accessible from anywhere in the program

(e) What is system calls? Give several examples of system calls.

a system call is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). System calls provide an essential interface between a process and the operating system.

On Unix, Unix-like and other POSIX-compliant operating systems, popular system calls
are open, read, write, close, wait, execve, fork, exit, and kill. Many of today's operating systems have hundreds of system calls. For example, Linux has over 300 different calls, FreeBSD has over 500, while Plan 9 has 51

(f)What is Process Control Block?



Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

(g) What do you mean by process coalescing?

(h)What is disk scheduling? What are its main objectives?

With the help or Advent of the Multi-programming we can Execute Many Programs at a Time. So fir Controlling and providing the Memory to all the Processes Operating System uses the Concept of Disk Scheduling. In this the Time of CPU is divided into the various Processes and also determines that all the processes will Work Properly. So

that Disk Scheduling Will Specifies that at which time which Process will be executed by the CPU. So that the Scheduling means to Execute all the Processes those are given to a CPU at a Time.

The Scheduling is used for Divide the Total Time of the CPU between the Number or Processes So that the Processes can execute Concurrently at a Single Time.

Disk scheduling techniques:
(1)First-come, First-served
(2)Shortest-seek-time-first(SSTF)
(3)scan
4 c-scan

(i) Define mutual exclusion.

mutual exclusion refers to the requirement of ensuring that no two processes or threads (henceforth referred to only as processes) are in their critical section at the same time. Mutual Exclusion is the concept of restricting access to a shared resource. When multiple processes perform operations on a single resource then they might corrupt it. Its the operating systems' responsibility to make sure that this does not happen. There are many methods that can be used to implement mutual exclusion such as semaphores, monitors, etc. Mutual exclusion has the following properties.
Safety: No two processes must use the shared resource at the same time. (Should not be in the critical section at the same time.)
Liveliness: There should not be deadlocks and a process comes out of the critical section after some time.
Fairness: A process wanting to use critical section must only wait some time.

(j) What is file allocation table?

File Allocation Table (FAT) is the name of a computer file system architecture and a family of industry standard file systems utilizing it.The FAT file system is a legacy file system which is simple and robust. It offers good performance even in light-weight implementations, but cannot deliver the same performance, reliability and scalability as some modern file systems. It is however supported for compatibility reasons by virtually all existing operating systems for personal computers, and thus is a well-suited format for data exchange between computers and devices of almost any type and age from the early 1980s up to the present. FAT file systems are still commonly found on floppy disks, solid-state memory cards, flash memory cards, and on many portable and embedded devices. It is also utilized in the boot stage of EFI-compliant computers.

Q2. What is virtual memory? What hardware supports are needed to implement virtual memory? Explain with the help of an example that FIFO page replacement algorithm may encounter more number of page faults encountered by LRU page replacement algorithm.

Ans-2. virtual memory is a memory management technique that is implemented using both hardware and software. It maps memory addresses used by a program, called *virtual addresses*, into *physical addresses* in computer memory. The operating system manages virtual address spaces and the assignment of real memory to virtual memory. Software within the operating system may extend these capabilities to provide a virtual address space that can exceed the capacity of real memory and thus reference more memory than is physically present in the computer.

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.

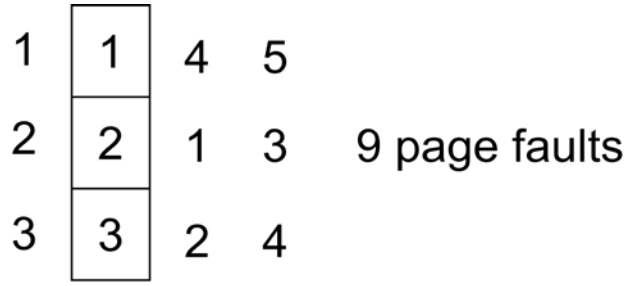
Virtual memory can be implemented via:

- Demand paging
- Demand segmentation

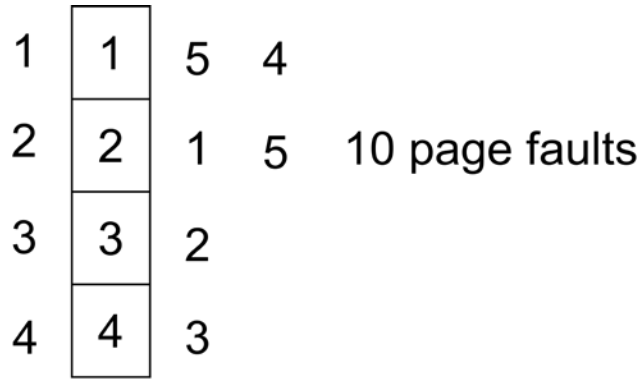
RAM is needed to implement virtual memory.

The simplest page-replacement algorithm is a FIFO algorithm. The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little book-keeping on the part of the operating system. The idea is obvious from the name – the operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the earliest arrival in front. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected. While FIFO is cheap and intuitive, it performs poorly in practical application. Thus, it is rarely used in its unmodified form.

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

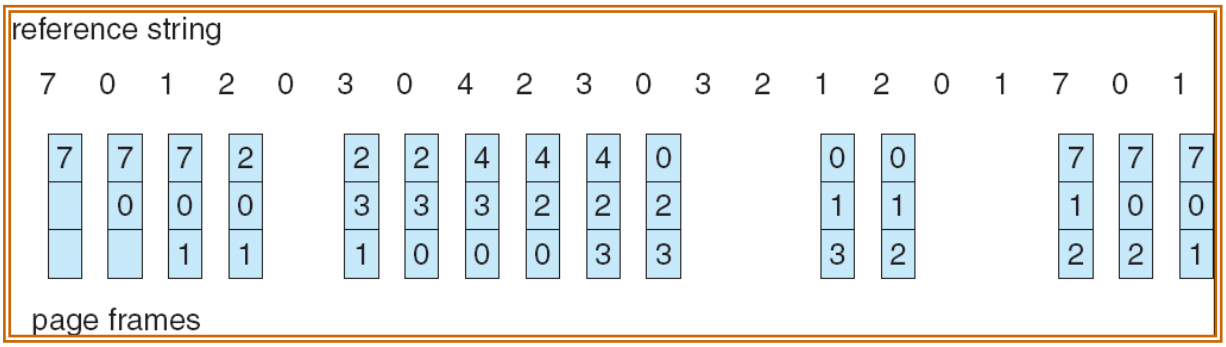


- 4 frames



FIFO Replacement – Belady's Anomaly

- more frames ⇒ more page faults



Sakshi Dewangan
06813702011
BCA(M)-V

LAKSHAY VERMA

03613702011

OS ASSIGNMENT

YEAR-2010

Q1

1. What is the difference between parallel and distributed system. Discuss brief.
Sol.

PARALLEL SYSTEM	DISTRUBUTEDSYSTEM
• A distributed system is a collection of independent computers, interconnected via a network, capable of collaborating on a task. distributed system can be characterized as collection of multiple autonomous computers that communicate over a communication network and having following features: No common Physical clock, Enhanced Reliability, Increased performance/cost ratio, Access to geographically remote data and resources, Scalability.	• A system is said to be a Parallel System in which multiple processor have direct access to shared memory which forms a common Usually tightly-coupled system are referred to as address space. Parallel System. In these systems, there is a single system wide primary memory (address space) that is shared by all the processors. On the other hand Distributed System are loosely-coupled system. Parallel computing is the use of two or more processors (cores, computers) in combination to solve a single problem.
• Distributed Operating systems are also referred to as Loosely Coupled systems. • A Loosely coupled system is one in which the processors do not share memory and each processor has its own local memory.	• Parallel processing g systems are referred to as loosely coupled systems. • In a tightly coupled system there is a single system wide primary memory shared by all the processors.
• The processors of distributed operating systems can be placed far away from each other to cover a wider geographic area. • Distributed operating system a larger no. of processors can be usefully deployed	• This is not the case with parallel processing systems. • The no. of processors that can be usefully deployed is very small in a parallel processing operating system

• Examples of Distributed System ATM(bank), Computer Networks Such as internet or intranet, Networks Distributed database and distributed database, Machines management Mobile Computing etc, Network of Workstations system .	• EXAMPLE: The Earth Simulator Supercomputer from (2002-2004) 12
• Advantages Of Distributed System Information Sharing among, Distributed Users , Resource Sharing Extensibility and Incremental growth Shorter Response Time and Higher Output Higher Reliability Better Flexibility's in meeting User's needs Better price/performance ratio Scalability Transparency	• Provide Concurrency(do multiple things Advantages of Parallel System at the Cost Taking advantage of non-local resources same time) Global Save time and money Overcoming memory constraints Savings address space provides a user-friendly programming perspective to memory
• Disadvantages of Distributed System Difficulties of developing distributed software Networking Problem Security Problems Performance Openness Reliability and Fault Tolerance 8	• Primary disadvantage is the lack of Disadvantages of Parallel System scalability Programmer responsibility for between memory and CPUs. synchronization constructs that ensure "correct" access of global It becomes increasingly difficult and expensive memory. to design and produce shared memory machines with ever increasing numbers of processors.

2. What do you mean by process? Discuss its states in brief.

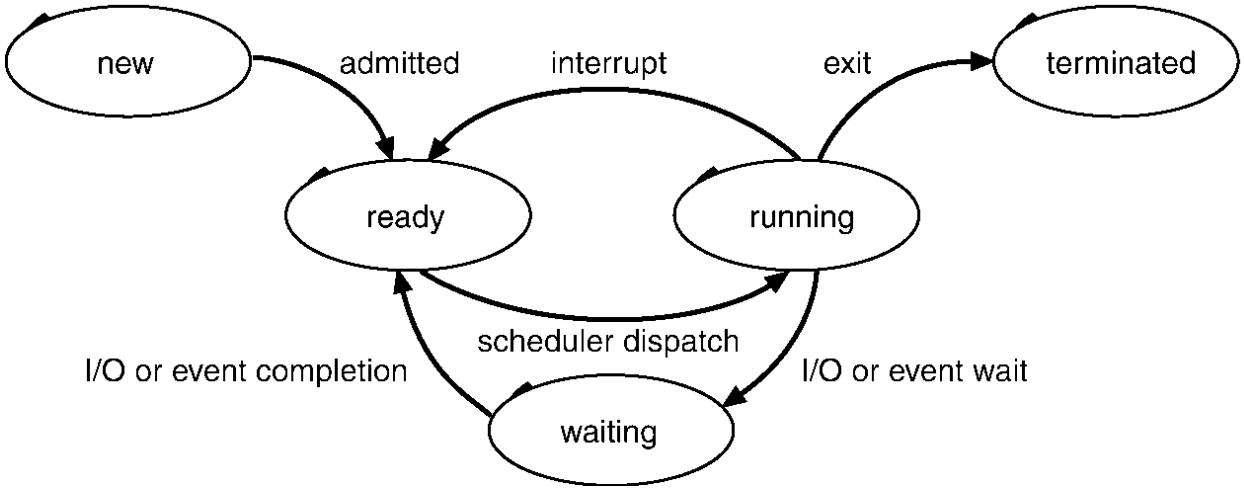
- A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
- Process creation and deletion.
- process suspension and resumption.
- Provision of mechanisms for:
- process synchronization
- process communication

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 process.
- terminated: The process has finished execution.



3. Define the difference between preemptive and non-preemptive scheduling.

Preemptive scheduling	**Non Preemptive scheduling**
• Preemptive: Preemptive algorithms are 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	**Non-Preemptive**: Non-preemptive algorithms are 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). context_switch() is called only when the process terminates or blocks.
• Preemptive: Preemptive algorithms are 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	**Non-Preemptive**: Non-preemptive algorithms are 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

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.

1. It allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process.
2. Complex to implement
3. Costly
4. High overhead
5. Process switches from running state to ready state and waiting state to ready state

processor).
context_switch() is called only when the process terminates or blocks.

It ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst

Simple to implement Cost is less Less overhead
Process switches from running state to waiting state and process terminates

4. Is it possible to have a dead lock involving only one process? Explain your answer.

1. Recall that there are four necessary conditions for deadlock.
2. Number 4 is circular-wait. Deadlock with one process is not possible, because it
3. is not possible to have circular wait with only one process, thus failing a
4. necessary condition. There is no second process to form a circle with the first
5. one. For the same reason, it is not possible to have a deadlock involving only
6. one resource. This follows directly from the hold-and-wait condition.

5. What are condition that lead to deadlock.

1. Mutual Exclusion Condition

The resources involved are non-shareable.

Explanation: At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait Condition

Requesting process hold already, resources while waiting for requested resources.

Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other

processes.

3. No-Preemptive Condition

Resources already allocated to a process cannot be preempted.

Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

4. Circular Wait Condition

The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

6. Discuss file attributes in brief

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

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

The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier.

.

7. What are the three main purposes of an operating system.

Convenience: An operating system makes a computer more convenient to use.
Efficiency: An operating system allows the computer system resources to be used in an efficient manner. **Ability to evolve:** An operating system should be constructed in such a way as to permit the effective development, testing, and

1. introduction of new system functions without interfering with service.

Ques: Explain critical section problem and discuss various algorithm to solve synchronization problem. List advantages and disadvantages of each.

Ans. Critical section is the segment of code where a process access the set of shared data. Shared data is always inside the critical section.

A **critical section problem** is to design a protocol that will give the order of execution of critical section.

Each process must get permission to execute its critical section. The section of code doing this is called **entry section**. If permission not granted then the process has to wait. The critical section is followed by the **exit section** which informs other processes that is has finished executing the critical section. The remaining code is the **remainder section**.

Solution to the critical section problem must satisfy the following characteristics:

1) **Mutual exclusion**: No two process can execute its critical section at the same time.
2) **Progress or Non-Blocking**: System should not be blocked forever due to the synchronization problem. The system must assure that some process competing for the critical section enters in finite time.
3) **Bounded waiting**: No process should be allowed to enter the critical section more than k-times when another process is waiting to execute its critical section.

Various algorithm to solve critical section problem are as follows:

1) **Two-process solutions:**
   In two process solution two processes are in critical section. These solutions are applicable to only two processes at a time. Only two processes P_0, P_1. Processes may share some common variables to synchronize their action.
   Let the processes P_0 & P_1.

   In other way the processes can be Pi and Pj. i-0; j=1-i;

   Processes may share a common variable to synchronize their action.

Algorithm

- The two processes share an integer variable turn initialized to i (or j).
- The process Pi finds the value of turn and when turn = = i, enters the critical section. Pj also finds turn as i and keeps waiting for turn = j.
- When the process Pi leaves the critical section, its sets the value of turn to j, allowing Pj to enter the critical section.

```
do
{
While (turn ! = i);   /* if true, Pi will continue in this loop. If turn=I, Pi enter in its critical section
<Critical section>;
turn =j;              /* Pi will set the value for Pj
<remainder section>
} while(1);
```

Structure of process Pi

This algorithm satisfies mutual exclusion but not progress.

2) <u>Perterson's solution</u>

- The algorithm has 3 Boolean variables flag[i], flag[j], turn. Flag variables are initialized to false. And turn to 0.
- When process Pi becomes ready for executing the critical section, it sets the flag[i] = true and turn = j (to tell the other process that it can also wish to execute its critical section) and enters the critical section.
- If both processes Pi and Pj sets the turn variable at the same time, the final value of turn variable will decide which of the two process enters the critical section.

In this processes share two variables:

Boolean flag [2];

int turn;

Initially flag [0] = flag [1] =false

turn may be 0 or 1;

```
do
 {
        flag [i] =  true;
        turn =j;                          //  Pj can also wish to enter the critical section
        while (flag [j]    &&    turn==j)    // if both values are true then Pj will execute its critical
                                            //section and if one of the value is false then Pi will
                                           //execute in its critical section
<Critical section>;
Flag [i] = false;                        // Pi will set its value to false and signaling other process can
                                            // enter in its critical section.
<Remainder section>;
} while(1);
```

Structure of Process Pi

This algorithm satisfies the 3 conditions to the critical section.

3) Solution using Test and set instruction

- The solution uses a single global Boolean variable lock initialized to false (false indicates the lock is free and true indicates lock is taken).
- The entry section contains a single while – loop statement where the process Pi sets the value of lock to true.
- If the original value of lock is false, the process breaks the loop and enters the critical section, otherwise it keeps trying to get the lock.
- In the exit section, it sets the lock value again to false.
- This solution is independent of number of processes.

```
do
{
While (TestAndSet (lock));
Critical section;
Lock = false;
Remainder section;
}while(1);
```

(Mutual exclusion implementation with TestAndSet.)

Definition of TestAndSet instruction

```
Boolean Testandset(boolean lock, Boolean newvalue)
{
  Boolean oldvalue;
  Oldvalue = lock;
  Lock = newvalue;
  return oldvalue;
    }
```

4) Solution using swap instruction

- The swap operation take the contents of two variables and interchanges them in a single operation.
- The solution uses a single global Boolean variable lock initialized to false (false indicates the lock is free and true indicates lock is taken).
- Each process uses a local Boolean variable key initialized to true.
- The entry section contains a single while – loop statement where the process Pi interchanges the values of lock and key.
- If the original value of lock is false, the process breaks the loop and enters the critical section, otherwise it keeps trying to get the lock.
- In the exit section, it sets the lock value again to false.

```
do
{
        Key =true;
        While (key==true)
        Swap (lock, key);
        Critical section;
        Lock=false;
        Remainder section;
} while(1);
```

(Mutual exclusion implementation with Swap instruction)

Definition of Swap instruction

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

Ques: What do you understand about process scheduling? Why is it important ?Explain various process scheduling techniques with example.

Ans. The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. A scheduler is a preemptive scheduler if it has the ability to get invoked by an interrupt and move a process out of a *running* state and let another process run. If a scheduler cannot take the CPU away from a process then it is a cooperative, or non-preemptive scheduler.

Process scheduling is important as it ensures efficient use of CPU. It also allows processes to run concurrently. It ensures that a deadlock should not occur. Also, it can stop a lower priority process to allow a higher priority process to execute. CPU executes all the process according to some rules or some schedule. Scheduling is that in which each process has some amount of time of CPU. Scheduling provides time of CPU to the each process.

There are three types of process scheduling:

1. Long term scheduling
2. Medium term scheduling
3. Short term scheduling

1)LONG TERM SCHEDULING : The long-term, or admission scheduler, decides which jobs or processes are to be admitted to the ready queue (in the Main Memory); that is, when an attempt is made to execute a program, its admission to the set of currently executing processes is either authorized or delayed by the long-term scheduler . Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time - i.e.: whether a high or low amount of processes are to be executed concurrently, and how the split between input output intensive and CPU intensive processes is to be handled. So long term scheduler is responsible for controlling the degree of multiprogramming. Long-term scheduling is

also important in large-scale systems such as batch processing systems, computer clusters, supercomputers and render farms. E.g. (Shortest Job first) if we want to print a page and move a mouse on the screen, So that CPU will first move the mouse on the screen. Then after that he will print a page. Because job of printing require a lots of time and moving a mouse is just requires little time of CPU.

2)MEDUIM TERM SCHEDULING: Scheduler temporarily removes processes from main memory and places them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as swapping out or swapping in The medium-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available, or when the process has been unblocked and is no longer waiting for a resource.

3)SHORT TERM SCHEDULING: The short-term scheduler (also known as the CPU scheduler) decides which of the ready, in-memory processes are to be executed (allocated a CPU) after a clock interrupt, an I/O interrupt, an operating system call or another form of signal. Thus the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers - a scheduling decision will at a minimum have to be made after every time slice, and these are very short. This scheduler can be pre-emptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process, or non-pre-emptive in which case the scheduler is unable to force processes off the CPU.

Q.What are the various security requirements for the operating system? What are different types of security policies for different types of operating system?

Some of the most common types of ***violations*** include:

Breach of Confidentiality - Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.

Breach of Integrity - Unauthorized ***modification*** of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.

Breach of Availability - Unauthorized ***destruction*** of data, often just for the "fun" of causing havoc and for bragging rights. Vandalism of web sites is a common form of this violation.

Theft of Service - Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.

Denial of Service, DOS - Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.

There are three requirements that can enhance operating system security across an enterprise network:-

First, provisioning of the servers on the network should be done once in one place, involving the roughly tens of separate configurations most organizations require. This image, or set of images, can then be downloaded across the network, with the help of software that automates this process and eliminates the pain of doing it manually for each server. Moreover, even if you had an instruction sheet for these key configurations, you wouldn't want local administrators to access these key configurations for each server, which is very dangerous. The best way to do it is once and for all. Once the network has been provisioned, administrators need to be able to verify policy compliance, which defines user access rights and ensures that all configurations are correct. An agent running on the network or remotely can monitor each server continuously, and such monitoring wouldn't interfere with normal operations.

Second, account management needs to be centralized to control access to the network and to ensure that users have appropriate access to enterprise resources. Policies, rules and intelligence should be located in one place—not on each box—and should be pushed out from there to provision user systems with correct IDs and permissions. An ID life cycle manager can be used to automate this process and reduce the pain of doing this manually.

Third, the operating system should be configured so that it can be used to monitor activity on the network easily and efficiently—revealing who is and isn't making connections, as well as pointing out potential security events coming out of the operating system. Administrators can use a central dashboard that monitors these events in real time and alerts them to serious problems based on preset correlations and filtering. Just as important, this monitoring system should be set up so that administrators aren't overwhelmed by routine events that don't jeopardize network security.

Security policies for LINUX Operating System:-

- Clean separation of policy from enforcement
- Well-defined policy interfaces
- Support for applications querying the policy and enforcing access control.
- Independent of specific policies and policy languages
- Independent of specific security label formats and contents
- Individual labels and controls for kernel objects and services
- Support for policy changes
- Separate measures for protecting system integrity (domain-type) and data confidentiality (multilevel security)
- Flexible policy
- Controls over process initialization and inheritance and program execution
- Controls over file systems, directories, files, and open file descriptors
- Controls over sockets, messages, and network interfaces
- Controls over use of "capabilities"

Security policies for UNIX Operating System:-

- Account Security
- Network Security
- Hardware (physical) Security
- File System Security

Security policies for WINDOWS Operating System:-

Active Desktop Security Policies: Active Desktop security policies include, Restrict users to change the wallpaper, restrict adding of any desktop items, Restrict deleting any desktop items, etc.

Control Panel Security Policies: Control Panel security policies include, Hide add/remove hardware applet, Hide add/remove programs applet, Hide games controller applet, etc.

Desktop Security Policies: Desktop security policies include, Hide and disable all items on the desktop, Prevent adding, dragging, dropping and closing the taskbar tool, Hide Internet Explorer icon on desktop, etc.

Internet Explorer Security Policies: Internet Explorer security policies include, Restrict changing proxy settings, Restrict changing history settings, Hide security option screen, etc.

Network Security Policies: Network security policies include, Ability to rename LAN, Ability to enable/disable LAN connections, alphanumeric password, etc.

Start Menu and Taskbar Security Policies: Start Menu and Taskbar security policies include, Prevent changes to taskbar and start menu settings, Remove run from start menu, Remove and prevent access to the shutdown command, etc.

System Security Policies: System security policies include, Restrict using registry editing tools, Restrict using change passwords page, Hide device manager page, etc.

Task Scheduler Security Policies: Task Scheduler security policies include, Prevent task run or end, Prohibit task deletion, Prohibit browse, etc.

Windows Installer Security Policies: Windows Installer security policies include, always install with elevated privileges, Prohibit rollback, and Disable media source for any install.

OPERATING SYSTEM

ASSIGNMENT

Q (a) Describe the shortest–job-first scheduling algorithm.

(b) What are the mechanisms to evaluate an algorithm related to CPU scheduling? Discuss any one of them.

ANS. (a) **Shortest-job-first scheduling:**
This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is free, it is assigned to the process of the ready queue which has smallest next CPU burst. If two processes have the same length, FCFS scheduling is used to break the tie. SJF scheduling algorithm is used frequently in long term scheduling. SJF algorithm may be either pre-emptive or non-pre-emptive.

A pre-emptive SJF algorithm will pre-empt the currently executing process whereas a non-pre-emptive SJF algorithm will allow the currently running process to finish its CPU burst. Let us consider the set of process with burst time in milliseconds.

SJF algorithm cannot be implemented at the level of short term CPU scheduling. There is no way to know the length of the next CPU burst

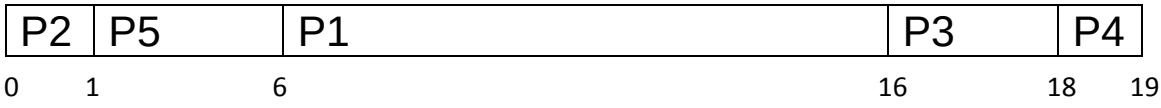
(b) There are several mechanisms to evaluate an algorithm related to CPU scheduling.

1. First-come first-serve scheduling,FCFS
2. Shortest-job-first scheduling, SJF
3. Priority scheduling
4. Round robin scheduling
5. Multilevel queue scheduling
6. Multilevel feedback-queue scheduling

Priority scheduling

- Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first(SJF uses the inverse of the next expected burst time as its priority- the smaller the expected burst, the higher the priority.)
- Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether “high” priorities use large numbers or small numbers.
- For example, the following Gantt chart is based upon these process burst times and priorities, and yields an average waiting time of 8.2ms:

PROCESS	BURST TIME	PRIORITY
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



- Priorities can be assigned either internally or externally. Internally priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.

- Priority scheduling can be either pre-emptive or non-pre-emptive.
- Priority scheduling can suffer from a major problem known as indefinite blocking, or starvation, in which a low priority task can wait forever because there are always some other jobs around that have higher priority.
    - If this problem is allowed to occur, then processs will either run eventually when the system load lightens, or will eventually get lost when the systems is shut down or crashes.
    - Other common solution to this problem is aging, in which priorities of jobs increase the longer they wait. Under this scheme a low priority job will eventually get its priority raised high enough that it gets run.

Q. (a) Describe the dining-philosopher's problem and provide its solution

(b) How will you handle synchronization problem using hardware? Discuss.

ANS. (a) There is*N* philosopher's sitting around circular table eating spaghetti and discussing philosophy. The problem is that each philosopher needs 2 forks to eat, and there are only *N* forks, one
Between each 2 philosophers. Design an algorithm that the philosophers can follow that insures that none starves as long as each philosopher eventually stops eating, and such that the maximum number of philosophers can eat at once.

Why describe problems this way? Well, the analogous situations in computers are sometimes so technical that they obscure creative thought. Thinking about philosophers makes it easier to thinkabstractly. And many of the early students of this field were theoreticians who like abstract problems.

Rashi Sharma
07313702011
Bca 5th sem (m)

Q3) section -c

a) what constitutes a state of a system ? what is safe state describe an algorithm to check whether a given state is safe or not ?

sol : A state is safe if the system can allocate resources to each processes in some order (safe sequence**)** and still avoid a deadlock

If no such sequence exists, then the system state is said to be unsafe

To illustrate, we consider a system has 12 magnetic tape drives and 3 processes. P0 needs 10 tapes, P1 needs 4 and P2 needs 9.

Currently, P0 has 5, P1 has 2 and P2 has 2

	Max needs	Current needs
P0	10	5
P1	4	2
P2	9	2

At T0, the system is in safe state, since <P1,P0,P2> satisfied safe state condition

safe state describe an algorithm

For the Banker's algorithm to work, it needs to know three things:

- How much of each resource each process could possibly request[CLAIMS]
- How much of each resource each process is currently holding[ALLOCATED]
- How much of each resource the system currently has available[AVAILABLE]

Resources may be allocated to a process only if it satisfies the following conditions:

1. request ≤ max, else set error condition as process has crossed maximum claim made by it.
2. request ≤ available, else process waits until resources are available.

Some of the resources that are tracked in real systems are memory, semaphores and interface access.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would allocate its money in such a way that it can no longer satisfy the needs of all its customers.

By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Max: An *n*×*m* matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- Allocation: An *n*×*m* matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instance of resource type R_j.
- Need: An *n*×*m* matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j].

EXAMPLE

1. P1 acquires 2 A, 1 B and 1 D more resources, achieving its maximum
   - [available resource: <3 1 1 2> - <2 1 0 1> = <1 0 1 1>]
   - The system now still has 1 A, no B, 1 C and 1 D resource available
2. P1 terminates, returning 3 A, 3 B, 2 C and 2 D resources to the system
   - [available resource: <1 0 1 1> + <3 3 2 2> = <4 3 3 3>]
   - The system now has 4 A, 3 B, 3 C and 3 D resources available
3. P2 acquires 2 B and 1 D extra resources, then terminates, returning all its resources
   - [available resource: <4 3 3 3>-<0 2 0 1>+<1 2 3 4> = <5 3 6 6>]
   - The system now has 5 A, 3 B, 6 C and 6 D resources
4. P3 acquires 1 B and 4 C resources and terminates
   - [available resource: <5 3 6 6> - <0 1 4 0> + <1 3 5 0> = <6 5 7 6>
   - The system now has all resources: 6 A, 5 B, 7 C and 6 D
5. Because all processes were able to terminate, this state is safe

b) describe the necessary condition for dead lock occurrence ? how can u prevent a system from deadlock ?

1. mutual exclusion
   The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.

2. hold and wait or partial allocation
   The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.

3. no pre-emption
   The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.

4. resource waiting or circular wait
   A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

Prevention

1. Deadlock prevention is the process of making it logically impossible for one of the 4 deadlock conditions to hold.
2. Relaxing mutual exclusion requires making all relevant resources sharable. Some resources can be made sharable. Spooling can make devices like printers or tape drives sharable. Spooling is storing the output on a shared medium, like disk, and using a single process to coordinate access to the shared resource. Print spooling is the cannonical example.
3. There are generally some resources that cannot be spooled - semaphores, for example. Removing mutual exclusion is not always an option.
4. Relaxing Hold and Wait requires processes to acquire all their needed resources at once. To acquire new resources in such a system requires a process to relinquish all the processes it holds and try to reacquire all the resources it needs atomically.

5. Reasonable systems can be programmed this way, but in practice it's almost never done. Starvation is a real possibility, and its probability increases with the number of resources concurrently acquired.

6. Relaing non-preemptability for resources other than the CPU is very non intuitive. The idea that a program might lose a lock at any time and have to reacquire it is very counterintuitive. If this is done with physical resources, things can be even more confusing. Other than very specialized cases, relaxing non-preemptability is almost never done.

7. The most common method of preventing deadlock is to prevent the circular wait. A simple way to do this, when possible, is to order the resources and always acquire them in order. Because a process can't be waiting on a lower numbered process while holding a higher numbered one, a cycle is impossible.

a) Discuss Contiguous , Linked , Indexed disk block allocation method with their merits and demerits.

SOL :

contiguous allocation

In contiguous allocation, files are assigned to contiguous areas of secondary storage. A user specifies in advance the size of the area needed to hold a file to be created. If the desired amount of contiguous space is not available, the file cannot be created.
Two most common strategies are:

i. First - fit
ii. Best - fit

First - fit - In this case as soon as the first hole (that is big enough) is encountered, searching is stopped and memory is allocated for creating a file. Searching can start either at the beginning of the set of holes or where the previous first - fit search ended.

Best - fit - In this case the entire list is searched for and the smallest hole, that is big enough, is allocated for creating a file.

Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first-fit is generally faster.

advantages

1. It supports fast sequential and direct access
2. It provides a good performance
3. the number of disk seek required is minimal

Disadvantage

1. fragmentation.
2. Space wastage and inflexibility

OPERATING SYSTEM ASSIGNMENT

Rashi Sharma
07313702011
Bca 5th sem (m)

Linked Allocation

Linked allocation is essentially a disk-based version of the linked list. With linked list allocation each file is linked list of disk blocks. These disk blocks may be scattered through the disk. A few bytes of each disk block contains the address of the next block. The directory contains a pointer to the first (and last) blocks of the file.

The advantage

(i) its simplicity

ii) no disk compaction required. Because of noncontiguous nature of allocation, the linking does not produce any external disk fragmentation. Any disk block on the free space can be used to satisfy a request, since all blocks are linked together. There is also no need of declaration of the size of a file in linked allocation while it is created. A file can continue to grow as there are free blocks. Consequently, it is never necessary to have disk compaction.

The disadvantages :.

i. Slow direct accessing of any disk block - To find out the Nth block of a file, we must start at the beginning of that file and follow the Dointer until we get to the Nth block.

ii. Space requirement for pointers - If a pointer requires two words out of a 512 words block, then around 39 percent of the disk is being pond only for pointers, not for information. Each file therefore requires more space.

iii. Reliability - Since disk blocks are linked by pointers, a single damaged pointer can make thousands of disk blocks inaccessible. Some operating systems address this problem by storing pointers in a dedicated file and making redundant copies of it. The idea is to copy the list of pointers into the main memory and thus facilitate faster access of disk blocks. Redundancy of pointers files is employed for safer recovery.

Indexed allocation

One disadvantage with linked allocation method is that it does not support direct accessing since blocks are scattered all over the disk. This problem is solved by indexed allocation by placing all of the pointers together into a index block.

advantages

the absence of external fragmentation and the efficiency of random accessing. Moreover, indexing can easily map around the bad disk blocks. Indexing of free space can be accomplished by means of the bit map.

disadvantage

the number of disk accesses necessary to retrieve the address of the target block on disk. This overhead can be reduced by keeping some index blocks in memory.

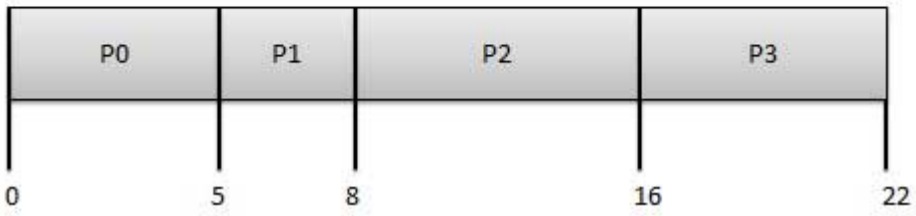
indexed allocation : it requires lots of space for keeping pointers. The pointer overhead of the index block is generally worse than the pointer overhead of linked allocation. Most files are generally small. With linked allocation for a file of only one or two blocks, we only need a space for one pointer per block. With indexed allocation, an entire index block must be allocated even if one or two pointers will be non-nil.

b) Discuss FCFS , SCAN , CSCAN DISK scheduling algorithm

first Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Rashi Sharma
07313702011
Bca 5th sem (m)

Wait time of each process is following

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.55

SCAN Disk Scheduling

- An elevator is designed to visit floors that have people waiting. In general, an elevator moves from one extreme to the other (say, the top of the building to the bottom), servicing requests as appropriate.

The scan algorithm has the head start at track 0 and move towards the highest numbered track, servicing all requests for a track as it passes the track. In another word, the disk«arm is required to move in one direction only satisfying all outstanding requests, until it readies the last track in that direction. The service direction is then reversed and the scan proceeds in the opposite direction, again picking up all requests in ord?r. SCAN algorithm is guaranteed to service every request in one complete pass through the disk. SCAN algorithm behaves almost identically with the SSTF algorithm. The SCAN algorithm is sometimes called the elevator algorithm. Fig.3 shows the SCAN algorithm. For example. The disk request queue contains a set of references for blocks on tracks 76, 124, 17, 269, 201, 29 and 137 and 12. inner track to outer track is t, then the expected service interval for sectors at the periphery is 2t. With SCAN if the excepted time for a scan from inner track to outer track is t, then the expected service interval for sectors at the periphery is 2t. With C-SCAN, the interval is on the order of t + S_{nm}_ Where S^ is the maximum seek time.

C-Scan Scheduling

- In the **C-Scan** all the Processes are Arranged by using Some Circular List. Circular List is that in which there is no start and end point of the list means the End of the List is the Starting Point of the list. In the C-Scan Scheduling the CPU will search for the Process from Start to end and if an End has Found then this again start from the Starting Process.

- Because Many Times When a CPU is executing the processes then may a user wants to enter some data means a user wants to enter Some data So that at that Situation the CPU will Again Execute that Process after the Input Operation. So that C-Scan Scheduling is used for Processing Same Processes again and Again

Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling, reverse direction and repeat. This algorithm is similar to SCAN, but unlike SCAN, the head does not unnecessarily travel to the innermost and outermost track -on each circuit. Fig.5 shows the look scheduling algorithm. For example, the disk request queue contains a set of references for blocks on tracks 76, 124, 17, 269, 201, 29, 137 and 12.

OPERATING SYSTEM

Assignment

Q1. What is round robin scheduling? Explain taking an example. Can it be useful for a single user system? If yes, then explain. If no, then why not?

Ans1. **Round Robin Scheduling**

- Designed especially for time-sharing systems.
- A small unit of time (time-quantum / time-slice) is defined generally from 10 to 100 milliseconds.
- Ready queue is treated as circular queue. CPU scheduler goes around the queue allocating the CPU for period of 1 time quantum.
- If the process has execution time less than 1 time quantum, the process voluntarily releases the CPU.
- If the process executing time exceeds the time quantum, the interrupt occurs, and process put at the tail of the ready queue.
- The performance of Round Robin algorithm depends on the time quantum. If too large, similar to FCFS and too small, the CPU would keep switching between processes.

Example:

PROCESS	CPU BURST TIME(MILLISECOND)
P1	24
P2	3
P3	3

Time Quantum = 4 milliseconds

GANTT CHART:

P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22 26 30

Turn-Around and Response times are P1=30, P2=7, P3=10 milliseconds

Waiting times are P1=6(10-4), P2=4, P3=7 milliseconds

Average Waiting Time = ((0+ [10-4] + 4+7))/3 = 17/3 = 5.66 milliseconds.Q2.List the difference between process and program.

Q2. What is the difference between dedicated and virtual devices?

Ans2.

- **Dedicated Devices**: Dedicated devices are devices that are assigned to only one job at a time and serves the job for the entire time it is active.
  - They serve the job for the entire time the job is active or until it releases them.
  - Some devices demand this kind of allocation scheme, because it would be awkward to let several users share them.
    - Example: tape drives, printers, and plotters
  - **Disadvantages**

They must be allocated to a single user for the duration of a job's execution, which can be quite inefficient, even though the device is not used 100% of the time.

- **Virtual Devices:** These are the peripheral device simulated by the operating system. If the device is offline, the operating system stores the output from the application until the device becomes available.
  - It is the combination of dedicated and shared devices.
  - They're dedicated devices that have been transformed into shared devices.
  - Example: printer
  - Converted into a shareable device through a spooling program that reroutes all print requests to a disk.
  - Only when all of a job's output is complete, and the printer is ready to print out the entire document, is the output sent to the printer for printing.
  - Because disks are shareable devices, this technique can convert one printer into several virtual printers, thus improving both its performance and use.
  - Example: universal serial bus (USB)

Q3. What is deadlock? What are the conditions for a deadlock to occur? How can it be avoided?

Ans4. In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; and if the resources are not available at that time1 the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

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

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

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

3. **No pre-emption**. Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. **Circular wait**. A set { $P_{0,}$ $P_{1.....}P_n$ } processes must exist such that is for a resource is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2……p_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held by P_0.

Deadlock avoidance:

Deadlock avoidance requires additional information about how resources are to be requested.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.