UNIT 3 I/O AND FILE MANAGEMENT

Structure **Page Nos.**

3.0 INTRODUCTION

Input and output devices are components that form part of the computer system. These devices are controlled by the operating system. Input devices such as keyboard, mouse, and sensors provide input signals such as commands to the operating system. These commands received from input devices instruct the operating system to perform some task or control its behaviour. Output devices such as monitors, printers and speakers are the devices that receive commands or information from the operating system.

In the earlier unit, we had studied the memory management of primary memory. The physical memory, as we have already seen, is not large enough to accommodate all of the needs of a computer system. Also, it is not permanent. Secondary storage consists of disk units and tape drives onto which data can be moved for permanent storage. Though there are actual physical differences between tapes and disks, the principles involved in controlling them are the same, so we shall only consider disk management here in this unit.

The operating system implements the abstract concept of the file by managing mass storage devices, such as types and disks. For convenient use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage Unit, the **file**. Files are mapped by the operating system, onto physical devices.

Definition: A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic or alphanumeric. Files may be free-form, such as text

files, or may be rigidly formatted. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user.

File management is one of the most visible services of an operating system. Computers can store information in several different physical forms among which magnetic tape, disk, and drum are the most common forms. Each of these devices has their own characteristics and physical organisation.

Normally files are organised into directories to ease their use. When multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed. The operating system is responsible for the following activities in connection with file management:

- The creation and deletion of files.
- The creation and deletion of directory.
- The support of primitives for manipulating files and directories.
- The mapping of files onto disk storage.
- Backup of files on stable (non volatile) storage.

The most significant problem in I/O system is the speed mismatch between I/O devices and the memory and also with the processor. This is because I/O system involves both H/W and S/W support and there is large variation in the nature of I/O devices, so they cannot compete with the speed of the processor and memory.

A well-designed file management structure makes the file access quick and easily movable to a new machine. Also it facilitates sharing of files and protection of non-public files. For security and privacy, file system may also provide encryption and decryption capabilities. This makes information accessible to the intended user only.

In this unit we will study the I/O and the file management techniques used by the operating system in order to manage them efficiently.

3.1 OBJECTIVES

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

- describe the management of the I/O activities independently and simultaneously with processor activities;
- summarise the full range of views that support file systems, especially the operating system view;
- compare and contrast different approaches to file organisations;
- discuss the disk scheduling techniques, and
- know how to implement the file system and its protection against unauthorised usage.

3.2 ORGANISATION OF THE I/O FUNCTION

The range of I/O devices and the large variation in their nature, speed, design, functioning, usage etc. makes it difficult for the operating system to handle them with any generality. The key concept in I/O software designing is device independence achieved by using uniform naming.

The name of the file or device should simply be a string or an integer and not dependent on the device in any way. Another key issue is sharable versus dedicated devices. Many users can share some I/O devices such as disks, at any instance of time. The devices like printers have to be dedicated to a single user until that user has finished an operation.

The basic idea is to organise the I/O software as a series of layers with the lower ones hiding the physical H/W and other complexities from the upper ones that present

simple, regular interface interaction with users. Based on this I/O software can be structured in the following four layers given below with brief descriptions:

(i) **Interrupt handlers**: The CPU starts the transfer and goes off to do something else until the interrupt arrives. The I/O device performs its activity independently and simultaneously with CPU activity. This enables the I/O devices to run asynchronously with the processor. The device sends an interrupt to the processor when it has completed the task, enabling CPU to initiate a further task. These interrupts can be hidden with the help of device drivers discussed below as the next I/O software layer.

(ii) **Device Drivers**: Each device driver handles one device type or group of closely related devices and contains a device dependent code. It accepts individual requests from device independent software and checks that the request is carried out. A device driver manages communication with a specific I/O device by converting a logical request from a user into specific commands directed to the device.

(iii) **Device-independent Operating System Software**: Its basic responsibility is to perform the I/O functions common to all devices and to provide interface with user-level software. It also takes care of mapping symbolic device names onto the proper driver. This layer supports device naming, device protection, error reporting, allocating and releasing dedicated devices etc.

(iv) **User level software**: It consists of library procedures linked together with user programs. These libraries make system calls. It makes I/O call, format I/O and also support spooling. Spooling is a way of dealing with dedicated I/O devices like printers in a multiprogramming environment.

3.3 I/O BUFFERING

A buffer is an intermediate memory area under operating system control that stores data in transit between two devices or between user's work area and device. It allows computation to proceed in parallel with I/O.

In a typical unbuffered transfer situation the processor is idle for most of the time, waiting for data transfer to complete and total read-processing time is the sum of all the transfer/read time and processor time as shown in *Figure 1*.

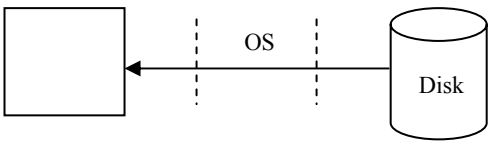

Figure 1: Unbuffered Transfers

In case of single-buffered transfer, blocks are first read into a buffer and then moved to the user's work area. When the move is complete, the next block is read into the buffer and processed in parallel with the first block. This helps in minimizing speed mismatch between devices and the processor. Also, this allows process computation in parallel with input/output as shown in *Figure 2*.

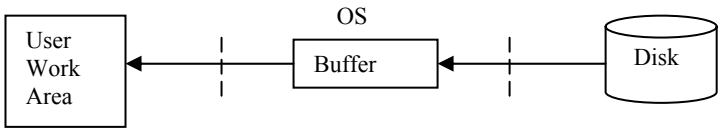

Figure 2: Single Buffering

Double buffering is an improvement over this. A pair of buffers is used; blocks/records generated by a running process are initially stored in the first buffer until it is full. Then from this buffer it is transferred to the secondary storage. During this transfer the other blocks generated are deposited in the second buffer and when this second buffer is also full and first buffer transfer is complete, then transfer from

the second buffer is initiated. This process of alternation between buffers continues which allows I/O to occur in parallel with a process's computation. This scheme increases the complexity but yields improved performance as shown in *Figure 3*.

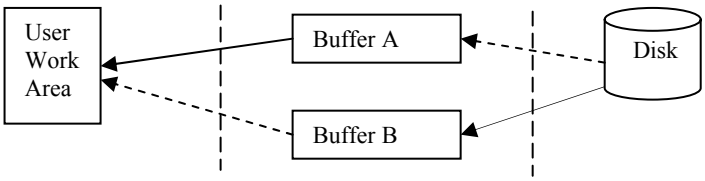

Figure 3: Double Buffering

3.4 DISK ORGANISATION

Disks come in different shapes and sizes. The most obvious distinction between floppy disks, diskettes and hard disks is: floppy disks and diskettes consist, of a single disk of magnetic material, while hard-disks normally consist of several stacked on top of one another. Hard disks are totally enclosed devices which are much more finely engineered and therefore require protection from dust. A hard disk spins at a constant speed, while the rotation of floppy drives is switched on and off. On the Macintosh machine, floppy drives have a variable speed operation, whereas most floppy drives have only a single speed of rotation. As hard drives and tape units become more efficient and cheaper to produce, the role of the floppy disk is diminishing. We look therefore mainly at hard drives.

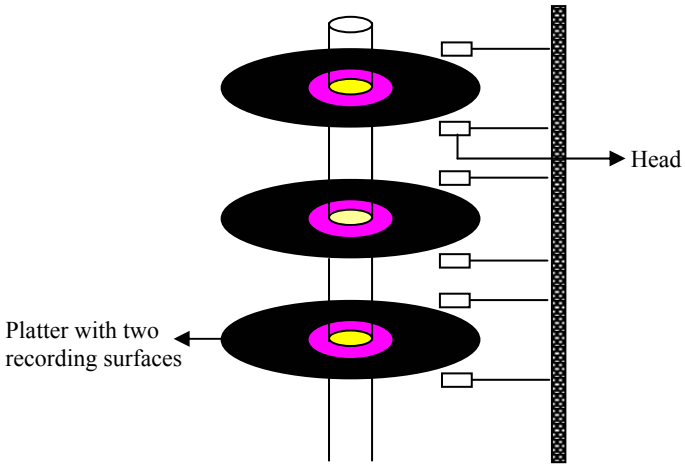

Figure 4: Hard Disk with 3 platters

Looking at the *Figure 4*, we see that a hard disk is composed of several physical disks stacked on top of each other. The disk shown in the *Figure 4* has 3 platters and 6 recording surfaces (two on each platter). A separate read *head* is provided for each *surface*. Although the disks are made of continuous magnetic material, there is a limit to the *density* of information which can be stored on the disk. The heads are controlled by a *stepper motor* which moves them in fixed-distance intervals across each surface. i.e., there is a fixed number of *tracks* on each surface. The tracks on all the surfaces are aligned, and the sum of all the tracks at a fixed distance from the edge of the disk is called a *cylinder*. To make the disk access quicker, tracks are usually divided up into *sectors* – or fixed size regions which lie along tracks. When writing to a disk, data are written in units of a whole number of sectors. (In this respect, they are similar to pages or frames in physical memory). On some disks, the sizes of sectors are decided by the manufacturers in hardware. On other systems (often microcomputers) it might be chosen in software when the disk is prepared for use (*formatting*). Because the heads of the disk move together on all surfaces, we can increase read-write efficiency by allocating blocks in parallel across all surfaces. Thus, if a file is stored in consecutive blocks, on a disk with *n* surfaces and *n* heads, it could read *n* sectors *per-track* without any head movement. When a disk is supplied by a manufacturer, the

physical properties of the disk (number of tracks, number of heads, sectors per track, speed of revolution) are provided with the disk. An operating system must be able to adjust to different types of disk. Clearly *sectors per track* is not a constant, nor is the number of tracks. The numbers given are just a convention used to work out a consistent set of addresses on a disk and may not have anything to do with the hard and fast physical limits of the disk. To address any portion of a disk, we need a three component address consisting of (*surface, track and sector*).

The *seek time* is the time required for the disk arm to move the head to the cylinder with the desired sector. The *rotational latency* is the time required for the disk to rotate the desired sector until it is under the read-write head.

3.4.1 Device drivers and IDs

A hard-disk is a *device*, and as such, an operating system must use a *device controller* to talk to it. Some device controllers are simple microprocessors which translate numerical addresses into head motor movements, while others contain small decision making computers of their own. The most popular type of drive for larger personal computers and workstations is the SCSI drive. SCSI (pronounced scuzzy) (Small Computer System Interface) is a protocol and now exists in four variants SCSI 1, SCSI 2, fast SCSI 2, and SCSI 3. SCSI disks live on a *data bus* which is a fast parallel data link to the CPU and memory, rather like a very short network. Each drive coupled to the bus identifies itself by a SCSI address and each SCSI controller can address up to seven units. If more disks are required, a second controller must be added. SCSI is more efficient at multiple accesses sharing than other disk types for microcomputers. In order to talk to a SCSI disk, an operating system must have a SCSI device driver. This is a layer of software which translates disk requests from the operating system's abstract command-layer into the language of signals which the SCSI controller understands.

3.4.2 Checking Data Consistency and Formatting

Hard drives are not perfect: they develop defects due to magnetic dropout and imperfect manufacturing. On more primitive disks, this is checked when the disk is *formatted* and these damaged sectors are avoided. If the sector becomes damaged under operation, the structure of the disk must be patched up by some repair program. Usually the data are lost.

On more intelligent drives, like the SCSI drives, the disk itself keeps a *defect list* which contains a list of all bad sectors. A new disk from the manufacturer contains a starting list and this is updated as time goes by, if more defects occur. Formatting is a process by which the sectors of the disk are:

- (If necessary) created by setting out 'signposts' along the tracks,
- Labelled with an address, so that the disk controller knows when it has found the correct sector.

On simple disks used by microcomputers, formatting is done manually. On other types, like SCSI drives, there is a low-level formatting already on the disk when it comes from the manufacturer. This is part of the SCSI protocol, in a sense. High level formatting on top of this is not necessary, since an advanced enough *filesystem* will be able to manage the hardware sectors. *Data consistency* is checked by writing to disk and reading back the result. If there is disagreement, an error occurs. This procedure can best be implemented inside the hardware of the disk−modern disk drives are small computers in their own right. Another cheaper way of checking data consistency is to calculate a number for each sector, based on what data are in the sector and store it in the sector. When the data are read back, the number is recalculated and if there is disagreement then an error is signalled. This is called a *cyclic redundancy check* (CRC) or *error correcting code*. Some device controllers are intelligent enough to be able to detect bad sectors and move data to a spare 'good' sector if there is an error. Disk design is still a subject of considerable research and disks are improving both in speed and reliability by leaps and bounds.

3.5 DISK SCHEDULING

The disk is a resource which has to be shared. It is therefore has to be scheduled for use, according to some kind of scheduling system. The secondary storage media structure is one of the vital parts of the file system. Disks are the one, providing lot of the secondary storage. As compared to magnetic tapes, disks have very fast access time and disk bandwidth. The access time has two major constituents: seek time and the rotational latency.

The *seek time* is the time required for the disk arm to move the head to the cylinder with the desired sector. The *rotational latency* is the time required for the disk to rotate the desired sector until it is under the read-write head. The *disk bandwidth* is the total number of bytes transferred per unit time.

Both the access time and the bandwidth can be improved by efficient disk I/O requests scheduling. Disk drivers are large single dimensional arrays of logical blocks to be transferred. Because of large usage of disks, proper scheduling algorithms are required.

A scheduling policy should attempt to maximize throughput (defined as the number of requests serviced per unit time) and also to minimize mean response time (i.e., average waiting time plus service time). These scheduling algorithms are discussed below:

3.5.1 FCFS Scheduling

First-Come, First-Served (FCFS) is the basis of this simplest disk scheduling technique. There is no reordering of the queue. Suppose the requests for inputting/ outputting to blocks on the cylinders have arrived, forming the following disk queue:

50, 91, 150, 42, 130, 18, 140, 70, 60

Also assume that the disk head is initially at cylinder 50 then it moves to 91, then to 150 and so on. The total head movement in this scheme is 610 cylinders, which makes the system slow because of wild swings. Proper scheduling while moving towards a particular direction could decrease this. This will further improve performance. FCFS scheme is clearly depicted in *Figure 5*.

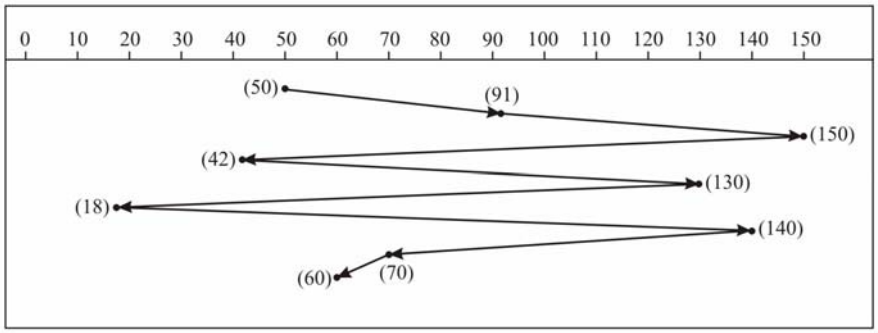

Figure 5: FCFS Scheduling

3.5.2 SSTF Scheduling

The basis for this algorithm is Shortest-Seek-Time-First (SSTF) i.e., service all the requests close to the current head position and with minimum seeks time from current head position.

In the previous disk queue sample the cylinder close to critical head position i.e., 50, is 42 cylinder, next closest request is at 60. From there, the closest one is 70, then 91,130,140,150 and then finally 18-cylinder. This scheme has reduced the total head movements to 248 cylinders and hence improved the performance. Like SJF (Shortest Job First) for CPU scheduling SSTF also suffers from starvation problem. This is

because requests may arrive at any time. Suppose we have the requests in disk queue for cylinders 18 and 150, and while servicing the 18-cylinder request, some other request closest to it arrives and it will be serviced next. This can continue further also making the request at 150-cylinder wait for long. Thus a continual stream of requests near one another could arrive and keep the far away request waiting indefinitely. The SSTF is not the optimal scheduling due to the starvation problem. This whole scheduling is shown in *Figure 6*.

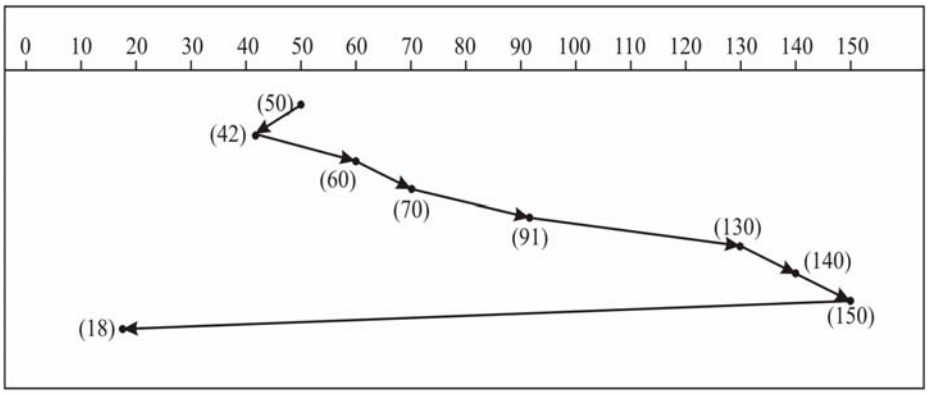

Figure 6: SSTF Scheduling

3.5.3 SCAN Scheduling

The disk arm starts at one end of the disk and service all the requests in its way towards the other end, i.e., until it reaches the other end of the disk where the head movement is reversed and continue servicing in this reverse direction. This scanning is done back and forth by the head continuously.

In the example problem two things must be known before starting the scanning process. Firstly, the initial head position i.e., 50 and then the head movement direction (let it towards 0, starting cylinder). Consider the disk queue again:

91, 150, 42, 130, 18, 140, 70, 60

Starting from 50 it will move towards 0, servicing requests 42 and 18 in between. At cylinder 0 the direction is reversed and the arm will move towards the other end of the disk servicing the requests at 60, 70, 91,130,140 and then finally 150.

As the arm acts like an elevator in a building, the SCAN algorithm is also known as elevator algorithm sometimes. The limitation of this scheme is that few requests need to wait for a long time because of reversal of head direction. This scheduling algorithm results in a total head movement of only 200 cylinders. *Figure 7* shows this scheme:

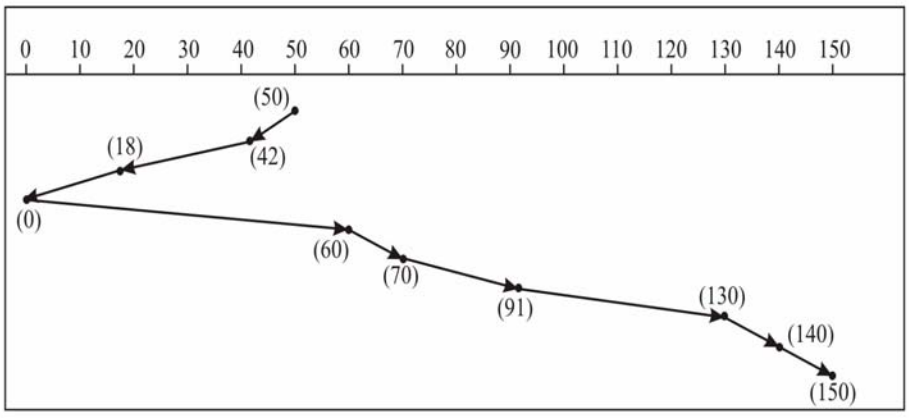

Figure 7: SCAN Scheduling

3.5.4 C-SCAN Scheduling

Similar to SCAN algorithm, C-SCAN also moves head from one end to the other servicing all the requests in its way. The difference here is that after the head reaches

the end it immediately returns to beginning, skipping all the requests on the return trip. The servicing of the requests is done only along one path. Thus comparatively this scheme gives uniform wait time because cylinders are like circular lists that wrap around from the last cylinder to the first one.

3.5.5 LOOK and C-LOOK Scheduling

These are just improvements of SCAN and C-SCAN but difficult to implement. Here the head moves only till final request in each direction (first and last ones), and immediately reverses direction without going to end of the disk. Before moving towards any direction the requests are looked, avoiding the full width disk movement by the arm.

The performance and choice of all these scheduling algorithms depend heavily on the number and type of requests and on the nature of disk usage. The file allocation methods like contiguous, linked or indexed, also affect the requests. For example, a contiguously allocated file will generate nearby requests and hence reduce head movements whereas linked or indexed files may generate requests from blocks that are scattered throughout the disk and hence increase the head movements. While searching for files the directories will be frequently accessed, hence location of directories and also blocks of data in them are also important criteria. All these peculiarities force the disk scheduling algorithms to be written as a separate module of the operating system, so that these can easily be replaced. For heavily used disks the SCAN / LOOK algorithms are well suited because they take care of the hardware and access requests in a reasonable order. There is no real danger of starvation, especially in the C-SCAN case. The arrangement of data on a disk plays an important role in deciding the efficiency of data-retrieval.

3.6 RAID

Disks have high failure rates and hence there is the risk of loss of data and lots of downtime for restoring and disk replacement. To improve disk usage many techniques have been implemented. One such technology is RAID (Redundant Array of Inexpensive Disks). Its organisation is based on disk striping (or interleaving), which uses a group of disks as one storage unit. Disk striping is a way of increasing the disk transfer rate up to a factor of *N*, by splitting files across *N* different disks. Instead of saving all the data from a given file on one disk, it is split across many. Since the *N* heads can now search independently, the speed of transfer is, in principle, increased manifold. Logical disk data/blocks can be written on two or more separate physical disks which can further transfer their sub-blocks in parallel. The total transfer rate system is directly proportional to the number of disks. The larger the number of physical disks striped together, the larger the total transfer rate of the system. Hence, the overall performance and disk accessing speed is also enhanced. The enhanced version of this scheme is mirroring or shadowing. In this RAID organisation a duplicate copy of each disk is kept. It is costly but a much faster and more reliable approach. The disadvantage with disk striping is that, if one of the *N* disks becomes damaged, then the data on all *N* disks is lost. Thus striping needs to be combined with a reliable form of backup in order to be successful.

Another RAID scheme uses some disk space for holding parity blocks. Suppose, three or more disks are used, then one of the disks will act as a parity block, which contains corresponding bit positions in all blocks. In case some error occurs or the disk develops a problems all its data bits can be reconstructed. This technique is known as disk striping with parity or block interleaved parity, which increases speed. But writing or updating any data on a disk requires corresponding recalculations and changes in parity block. To overcome this the parity blocks can be distributed over all disks.

3.7 DISK CACHE

Disk caching is an extension of buffering. Cache is derived from the French word cacher, meaning to hide. In this context, a ***cache*** is a collection of blocks that logically belong on the disk, but are kept in memory for performance reasons. It is used in multiprogramming environment or in disk file servers, which maintain a separate section of main memory called disk cache. These are sets of buffers (cache) that contain the blocks that are recently used. The cached buffers in memory are copies of the disk blocks and if any data here is modified only its local copy is updated. So, to maintain integrity, updated blocks must be transferred back to the disk. Caching is based on the assumption that most shortly accessed blocks are likely to be accessed again soon. In case some new block is required in the cache buffer, one block already there can be selected for "flushing" to the disk. Also to avoid loss of updated information in case of failures or loss of power, the system can periodically flush cache blocks to the disk. The key to disk caching is keeping frequently accessed records in the disk cache buffer in primary storage.

☞ **Check Your Progress 1**

1) Indicate the major characteristics which differentiate I/O devices.

……

……

……

2) Explain the term device independence. What is the role of device drivers in this context?

……

……

……

3) Describe the ways in which a device driver can be implemented?

……

……

……

4) What is the advantage of the double buffering scheme over single buffering?

……

……

……

5) What are the key objectives of the I/O system?

……

……

……

……

3.8 COMMAND LANGUAGE USER'S VIEW OF THE FILE SYSTEM

The most important aspect of a file system is its appearance from the user's point of view. The user prefers to view the naming scheme of files, the constituents of a file, what the directory tree looks like, protection specifications, file operations allowed on them and many other interface issues. The internal details like, the data structure used

for free storage management, number of sectors in a logical block etc. are of less interest to the users. From a user's perspective, a file is the smallest allotment of logical secondary storage. Users should be able to refer to their files by symbolic names rather than having to use physical device names.

The operating system allows users to define named objects called files which can hold interrelated data, programs or any other thing that the user wants to store/save.

3.9 THE SYSTEM PROGRAMMER'S VIEW OF THE FILE SYSTEM

As discussed earlier, the system programmer's and designer's view of the file system is mainly concerned with the details/issues like whether linked lists or simple arrays are used for keeping track of free storage and also the number of sectors useful in any logical block. But it is rare that physical record size will exactly match the length of desired logical record. The designers are mainly interested in seeing how disk space is managed, how files are stored and how to make everything work efficiently and reliably.

3.10 THE OPERATING SYSTEM'S VIEW OF FILE MANAGEMENT

As discussed earlier, the operating system abstracts (maps) from the physical properties of its storage devices to define a logical storage unit i.e., the file. The operating system provides various system calls for file management like creating, deleting files, read and write, truncate operations etc. All operating systems focus on achieving device-independence by making the access easy regardless of the place of storage (file or device). The files are mapped by the operating system onto physical devices. Many factors are considered for file management by the operating system like directory structure, disk scheduling and management, file related system services, input/output etc. Most operating systems take a different approach to storing information. Three common file organisations are byte sequence, record sequence and tree of disk blocks. UNIX files are structured in simple byte sequence form. In record sequence, arbitrary records can be read or written, but a record cannot be inserted or deleted in the middle of a file. CP/M works according to this scheme. In tree organisation each block hold ***n*** keyed records and a new record can be inserted anywhere in the tree. The mainframes use this approach. The OS is responsible for the following activities in regard to the file system:

- The creation and deletion of files
- The creation and deletion of directory
- The support of system calls for files and directories manipulation
- The mapping of files onto disk
- Backup of files on stable storage media (non-volatile).

The coming sub-sections cover these details as viewed by the operating system.

3.10.1 Directories

A file directory is a group of files organised together. An entry within a directory refers to the file or another directory. Hence, a tree structure/hierarchy can be formed. The directories are used to group files belonging to different applications/users. Large-scale time sharing systems and distributed systems store thousands of files and bulk of data. For this type of environment a file system must be organised properly. A File system can be broken into partitions or volumes. They provide separate areas within one disk, each treated as separate storage devices in which files and directories reside. Thus directories enable files to be separated on the basis of user and user applications, thus simplifying system management issues like backups, recovery,

security, integrity, name-collision problem (file name clashes), housekeeping of files etc.

The device directory records information like name, location, size, and type for all the files on partition. A root refers to the part of the disk from where the root directory begins, which points to the user directories. The root directory is distinct from sub-directories in that it is in a fixed position and of fixed size. So, the directory is like a symbol table that converts file names into their corresponding directory entries. The operations performed on a directory or file system are:

1) Create, delete and modify files.
2) Search for a file.
3) Mechanisms for sharing files should provide controlled access like read, write, execute or various combinations.
4) List the files in a directory and also contents of the directory entry.
5) Renaming a file when its contents or uses change or file position needs to be changed.
6) Backup and recovery capabilities must be provided to prevent accidental loss or malicious destruction of information.
7) Traverse the file system.

The most common schemes for describing logical directory structure are:

(i) **Single-level directory**

All the files are inside the same directory, which is simple and easy to understand; but the limitation is that all files must have unique names. Also, even with a single user as the number of files increases, it is difficult to remember and to track the names of all the files. This hierarchy is depicted in *Figure 8.*

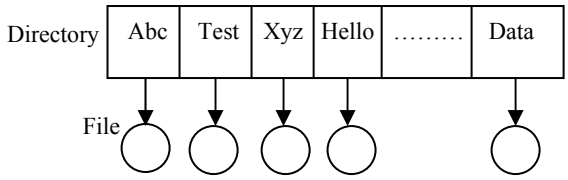

Figure 8: Single-level directory

(ii) **Two-level directory**

We can overcome the limitations of the previous scheme by creating a separate directory for each user, called User File Directory (UFD). Initially when the user logs in, the system's Master File Directory (MFD) is searched which is indexed with respect to username/account and UFD reference for that user. Thus different users may have same file names but within each UFD they should be unique. This resolves name-collision problem up to some extent but this directory structure isolates one user from another, which is not desired sometimes when users need to share or cooperate on some task. *Figure 9* shows this scheme clearly.

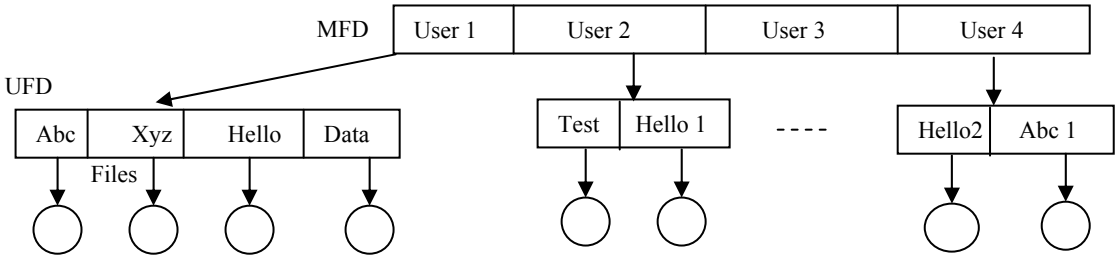

Figure 9: Two-level directory

(iii) **Tree-structured directory**

The two-level directory structure is like a 2-level tree. Thus to generalise, we can extend the directory structure to a tree of arbitrary height. Thus the user can create his/her own directory and subdirectories and can also organise files. One bit in each directory entry defines entry as a file (0) or as a subdirectory (1).

The tree has a root directory and every file in it has a unique path name (path from root, through all subdirectories, to a specified file). The pathname prefixes the filename, helps to reach the required file traversed from a base directory. The pathnames can be of 2 types: absolute path names or relative path names, depending on the base directory. An absolute path name begins at the root and follows a path to a particular file. It is a full pathname and uses the root directory. Relative defines the path from the current directory. For example, if we assume that the current directory is */Hello2* then the file *F4.doc* has the absolute pathname */Hello/Hello2/Test2/F4.doc* and the relative pathname is */Test2/F4.doc*. The pathname is used to simplify the searching of a file in a tree-structured directory hierarchy. *Figure 10* shows the layout:

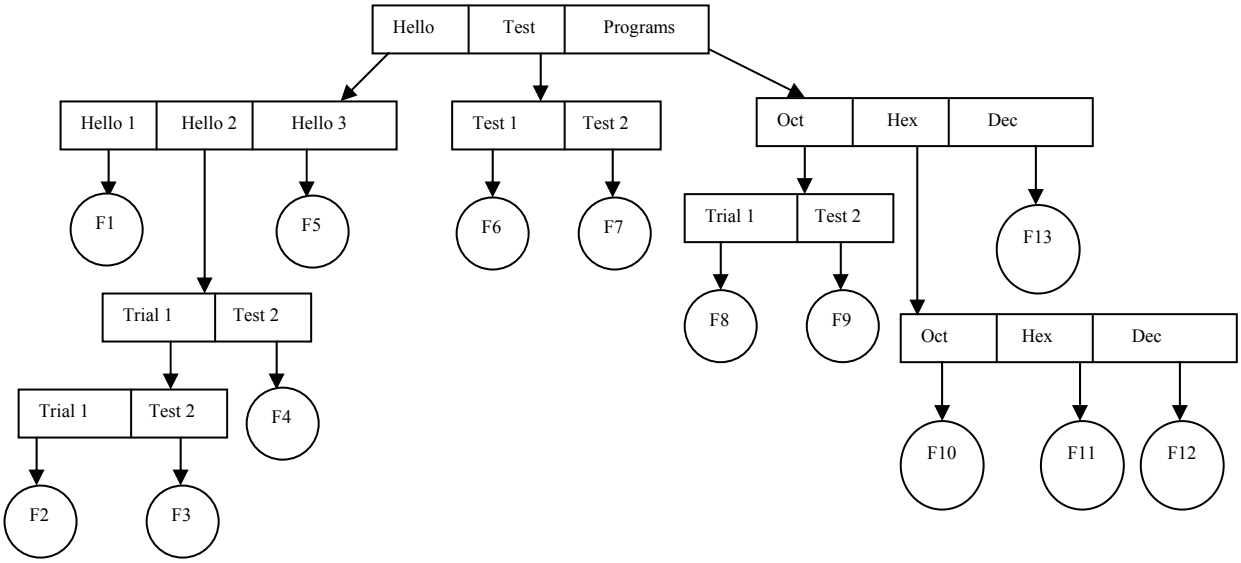

Figure 10: Tree-structured directory

(iv) **Acyclic-graph directory**:

As the name suggests, this scheme has a graph with no cycles allowed in it. This scheme added the concept of shared common subdirectory / file which exists in file system in two (or more) places at once. Having two copies of a file does not reflect changes in one copy corresponding to changes made in the other copy.

But in a shared file, only one actual file is used and hence changes are visible. Thus an acyclic graph is a generalisation of a tree-structured directory scheme. This is useful in a situation where several people are working as a team, and need access to shared files kept together in one directory. This scheme can be implemented by creating a new directory entry known as a link which points to another file or subdirectory. *Figure 11* depicts this structure for directories.

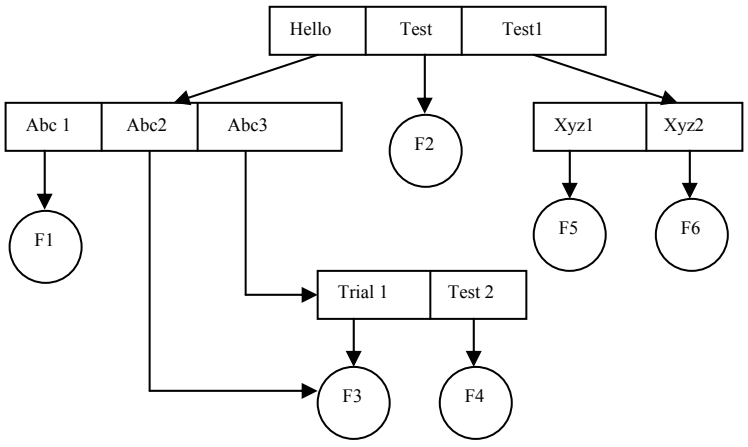

Figure 11: Acyclic-graph directory

The limitations of this approach are the difficulties in traversing an entire file system because of multiple absolute path names. Another issue is the presence of dangling pointers to the files that are already deleted, though we can overcome this by preserving the file until all references to it are deleted. For this, every time a link or a copy of directory is established, a new entry is added to the file-reference list. But in reality as the list is too lengthy, only a count of the number of references is kept. This count is then incremented or decremented when the reference to the file is added or it is deleted respectively.

(v) **General graph Directory**:

Acyclic-graph does not allow cycles in it. However, when cycles exist, the reference count may be non-zero, even when the directory or file is not referenced anymore. In such situation garbage collection is useful. This scheme requires the traversal of the whole file system and marking accessible entries only. The second pass then collects everything that is unmarked on a free-space list. This is depicted in *Figure 12*.

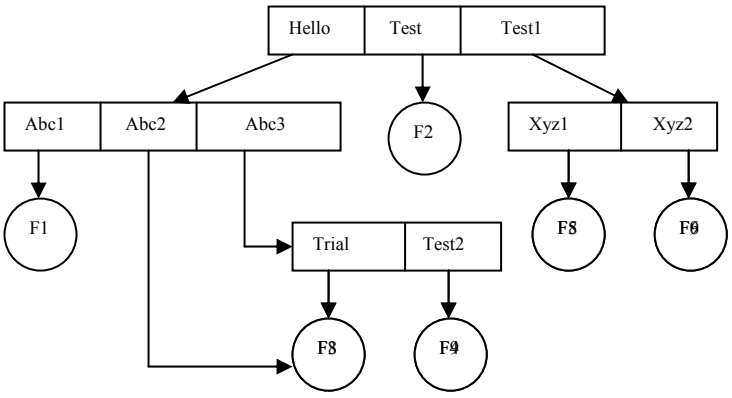

Figure 12: General-graph directory

3.10.2 Disk Space Management

The direct-access of disks and keeping files in adjacent areas of the disk is highly desirable**.** But the problem is how to allocate space to files for effective disk space utilisation and quick access. Also, as files are allocated and freed, the space on a disk become fragmented. The major methods of allocating disk space are:

i) Continuous
ii) Non-continuous (Indexing and Chaining)

i) **Continuous**

This is also known as contiguous allocation as each file in this scheme occupies a set of contiguous blocks on the disk. A linear ordering of disk addresses is seen on the disk. It is used in VM/CMS– an old interactive system. The advantage of this approach is that successive logical records are physically adjacent and require no head movement. So disk seek time is minimal and speeds up access of records. Also, this scheme is relatively simple to implement. The technique in which the operating system provides units of file space on demand by user running processes, is known as dynamic allocation of disk space. Generally space is allocated in units of a fixed size, called an allocation unit or a 'cluster' in MS-DOS. Cluster is a simple multiple of the disk physical sector size, usually 512 bytes. Disk space can be considered as a one-dimensional array of data stores, each store being a cluster. A larger cluster size reduces the potential for fragmentation, but increases the likelihood that clusters will have unused space. Using clusters larger than one sector reduces fragmentation, and reduces the amount of disk space needed to store the information about the used and unused areas on the disk.

Contiguous allocation merely retains the disk address (start of file) and length (in block units) of the first block. If a file is n blocks long and it begins with location b (blocks), then it occupies b, b+1, b+2,…, b+n-1 blocks. First-fit and best-fit strategies

can be used to select a free hole from available ones. But the major problem here is searching for sufficient space for a new file. *Figure 13* depicts this scheme:

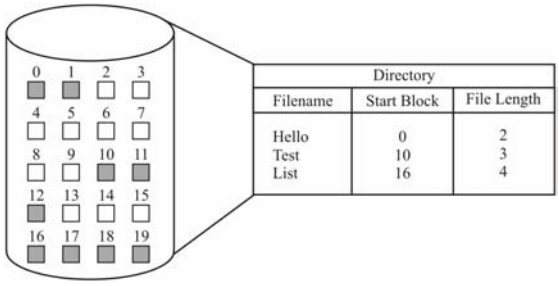

Figure 13: Contiguous Allocation on the Disk

This scheme exhibits similar fragmentation problems as in variable memory partitioning. This is because allocation and deal location could result in regions of free disk space broken into chunks (pieces) within active space, which is called external fragmentation. A repacking routine called compaction can solve this problem. In this routine, an entire file system is copied on to tape or another disk and the original disk is then freed completely. Then from the copied disk, files are again stored back using contiguous space on the original disk. But this approach can be very expensive in terms of time. Also, size-declaration in advance is a problem because each time, the size of file is not predictable. But it supports both sequential and direct accessing. For sequential access, almost no seeks are required. Even direct access with seek and read is fast. Also, calculation of blocks holding data is quick and easy as we need just offset from the start of the file.

ii) Non-Continuous (Chaining and Indexing)

This scheme has replaced the previous ones. The popular non-contiguous storages allocation schemes are:

- Linked/Chained allocation
- Indexed Allocation.

Linked/Chained allocation: All files are stored in fixed size blocks and adjacent blocks are linked together like a linked list. The disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last block of the file. Also each block contains pointers to the next block, which are not made available to the user. There is no external fragmentation in this as any free block can be utilised for storage. So, compaction and relocation is not required. But the disadvantage here is that it is potentially inefficient for direct-accessible files since blocks are scattered over the disk and have to follow pointers from one disk block to the next. It can be used effectively for sequential access only but there also it may generate long seeks between blocks. Another issue is the extra storage space required for pointers. Yet the reliability problem is also there due to loss/damage of any pointer. The use of doubly linked lists could be a solution to this problem but it would add more overheads for each file. A doubly linked list also facilitates searching as blocks are threaded both forward and backward. The *Figure 14* depicts linked /chained allocation where each block contains the information about the next block (i.e., pointer to next block).

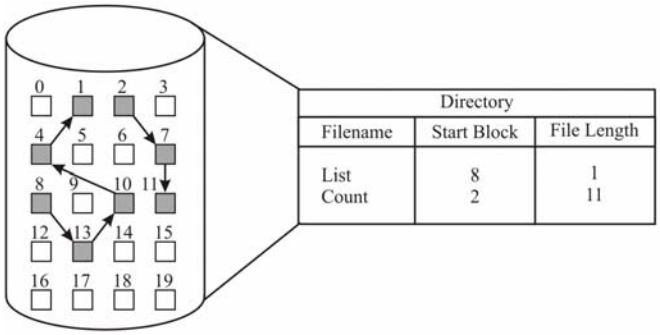

Figure 14: Linked Allocation on the Disk

MS-DOS and OS/2 use another variation on linked list called FAT (File Allocation Table). The beginning of each partition contains a table having one entry for each disk block and is indexed by the block number. The directory entry contains the block number of the first block of file. The table entry indexed by block number contains the block number of the next block in the file. The Table pointer of the last block in the file has EOF pointer value. This chain continues until EOF (end of file) table entry is encountered. We still have to linearly traverse next pointers, but at least we don't have to go to the disk for each of them. 0(Zero) table value indicates an unused block. So, allocation of free blocks with FAT scheme is straightforward, just search for the first block with 0 table pointer. MS-DOS and OS/2 use this scheme. This scheme is depicted in *Figure 15.*

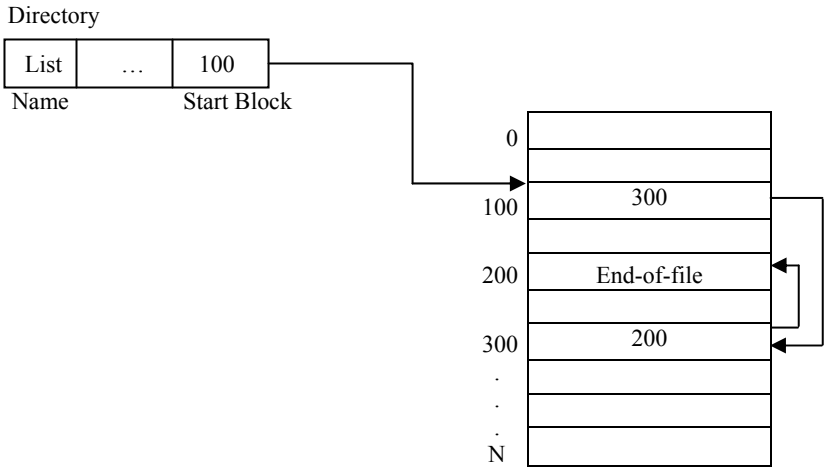

Figure 15: File-Allocation Table (FAT)

Indexed Allocation: In this each file has its own index block. Each entry of the index points to the disk blocks containing actual file data i.e., the index keeps an array of block pointers for each file. So, index block is an array of disk block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. Also, the main directory contains the address where the index block is on the disk. Initially, all the pointers in index block are set to ***NIL***. The advantage of this scheme is that it supports both sequential and random access. The searching may take place in index blocks themselves. The index blocks may be kept close together in secondary storage to minimize seek time. Also space is wasted only on the index which is not very large and there's no external fragmentation. But a few limitations of the previous scheme still exists in this, like, we still need to set maximum file length and we can have overflow scheme of the file larger than the predicted value. Insertions can require complete reconstruction of index blocks also. The indexed allocation scheme is diagrammatically shown in *Figure 16.*

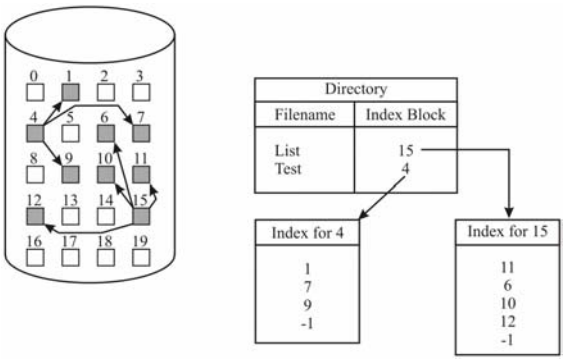

Figure 16: Indexed Allocation on the Disk

3.10.3 Disk Address Translation

We have seen in Unit-1 memory management that the virtual addresses generated by a program is different from the physical. The translation of virtual addresses to physical

addresses is performed by MMU. Disk address translation considers the aspects of data storage on the disk. Hard disks are totally enclosed devices, which are more finely engineered and therefore require protection from dust. A hard disk spins at a constant speed. Briefly, hard disks consist of one or more rotating platters. A read-write head is positioned above the rotating surface and is able to read or write the data underneath the current head position. The hard drives are able to present the "geometry" or "addressing scheme" of the drive. Consider the disk internals first. Each track of the disk is divided into sections called sectors. A sector is the smallest physical storage unit on the disk. The size of a sector is always a power of two, and is almost always 512 bytes. A sector is the part of a slice that lies within a track. The position of the head determines which track is being read. A cylinder is almost the same as a track, except that it means all tracks lining up under each other on all the surfaces. The head is equivalent to side(s). It simply means one of the rotating platters or one of the sides on one of the platters. If a hard disk has three rotating platters, it usually has 5 or 6 readable sides, each with its own read-write head.

The MS-DOS file systems allocate storage in clusters, where a cluster is one or more contiguous sectors. MS-DOS bases the cluster size on the size of the partition. As a file is written on the disk, the file system allocates the appropriate number of clusters to store the file's data. For the purposes of isolating special areas of the disk, most operating systems allow the disk surface to be divided into partitions. A partition (also called a cylinder group) is just that: a group of cylinders, which lie next to each other. By defining partitions we divide up the storage of data to special areas, for convenience. Each partition is assigned a separate logical device and each device can only write to the cylinders, which are defined as being its own. To access the disk the computer needs to convert physical disk geometry (the number of cylinders on the disk, number of heads per cylinder, and sectors per track) to a logical configuration that is compatible with the operating system. This conversion is called translation. Since sector translation works between the disk itself and the system BIOS or firmware, the operating system is unaware of the actual characteristics of the disk, if the number of cylinders, heads, and sectors per track the computer needs is within the range supported by the disk. MS-DOS presents disk devices as logical volumes that are associated with a drive code (A, B, C, and so on) and have a volume name (optional), a root directory, and from zero to many additional directories and files.

3.10.4 File Related System Services

A file system enables applications to store and retrieve files on storage devices. Files are placed in a hierarchical structure. The file system specifies naming conventions for files and the format for specifying the path to a file in the tree structure. OS provides the ability to perform input and output (I/O) operations on storage components located on local and remote computers. In this section we briefly describe the system services, which relate to file management. We can broadly classify these under categories:

i) Online services
ii) Programming services.

i) **Online-services**: Most operating systems provide interactive facilities to enable the on-line users to work with files. Few of these facilities are built-in commands of the system while others are provided by separate utility programs. But basic operating systems like MS-DOS with limited security provisions can be potentially risky because of these user owned powers. So, these must be used by technical support staff or experienced users only. For example: DEL *. * Command can erase all the files in the current directory. Also, FORMAT c: can erase the entire contents of the mentioned drive/disk. Many such services provided by the operating system related to directory operations are listed below:

- Create a file
- Delete a file

- Copy a file
- Rename a file
- Display a file
- Create a directory
- Remove an empty directory
- List the contents of a directory
- Search for a file
- Traverse the file system.

ii) **Programming services**: The complexity of the file services offered by the operating system vary from one operating system to another but the basic set of operations like: open (make the file ready for processing), close (make a file unavailable for processing), read (input data from the file), write (output data to the file), seek (select a position in file for data transfer).

All these operations are used in the form of language syntax procedures or built-in library routines, of high-level language used like C, Pascal, and Basic etc. More complicated file operations supported by the operating system provide wider range of facilities/services. These include facilities like reading and writing records, locating a record with respect to a primary key value etc. The software interrupts can also be used for activating operating system functions. For example, Interrupt 21(hex) function call request in MS-DOS helps in opening, reading and writing operations on a file.

In addition to file functions described above the operating system must provide directory operation support also like:

- Create or remove directory
- Change directory
- Read a directory entry
- Change a directory entry etc.

These are not always implemented in a high level language but language can be supplied with these procedure libraries. For example, UNIX uses C language as system programming language, so that all system calls requests are implemented as C procedures.

3.10.5 Asynchronous Input/Output

Synchronous I/O is based on blocking concept while asynchronous is interrupt-driven transfer. If an user program is doing several things simultaneously and request for I/O operation, two possibilities arise. The simplest one is that the I/O is started, then after its completion, control is transferred back to the user process. This is known as synchronous I/O where you make an I/O request and you have to wait for it to finish. This could be a problem where you would like to do some background processing and wait for a key press. Asynchronous I/O solves this problem, which is the second possibility. In this, control is returned back to the user program without waiting for the I/O completion. The I/O then continues while other system operations occur. The CPU starts the transfer and goes off to do something else until the interrupt arrives.

Asynchronous I/O operations run in the background and do not block user applications. This improves performance, because I/O operations and applications processing can run simultaneously. Many applications, such as databases and file servers, take advantage of the ability to overlap processing and I/O. To manage asynchronous I/O, each asynchronous I/O request has a corresponding control block in the application's address space. This control block contains the control and status information for the request. It can be used again when the I/O operation is completed. Most physical I/O is asynchronous.

After issuing an asynchronous I/O request, the user application can determine when and how the I/O operation is completed. This information is provided in any of three ways:

- The application can poll the status of the I/O operation.
- The system can asynchronously notify the application when the I/O operation is done.
- The application can block until the I/O operation is complete.

Each I/O is handled by using a device-status table. This table holds entry for each I/O device indicating device's type, address, and its current status like bust or idle. When any I/O device needs service, it interrupts the operating system. After determining the device, the Operating System checks its status and modifies table entry reflecting the interrupt occurrence. Control is then returned back to the user process.

☞ Check Your Progress 2

1) What is the meaning of the term 'virtual device'? Give an Example.

……

……

2) What are the advantages of using directories?

……

……

3) Explain the advantages of organising file directory structure into a tree structure?

……

……

4) List few file attributes?

……

……

5) Why is SCAN scheduling also called Elevator Algorithm?

……

……

6) In an MS-DOS disk system, calculate the number of entries (i.e., No. of clusters) required in the FAT table. Assume the following parameters:

Disk Capacity - 40 Mbytes
Block Size - 512 Bytes
Blocks/Cluster- 4

……

……

7) Assuming a cluster size of 512 bytes calculate the percentage wastage in file space due to incomplete filling of last cluster, for the file sizes below:

(i) 1000 bytes (ii) 20,000 bytes

……

……

8) What is meant by an 'alias filename' and explain its UNIX implementation.

……

……

3.11 SUMMARY

This unit briefly describes the aspects of I/O and File Management. We started by looking at I/O controllers, organisation and I/O buffering. We briefly described various buffering approaches and how buffering is effective in smoothing out the speed mismatch between I/O rates and processor speed. We also looked at the four levels of I/O software: the interrupt handlers, device drivers, the device independent I/O software, and the I/O libraries and user-level software.

A well-designed file system should provide a user-friendly interface. The file system generally includes access methods, file management issues like file integrity, storage, sharing, security and protection etc. We have discussed the services provided by the operating system to the user to enable fast access and processing of files.

The important concepts related to the file system are also introduced in this unit like file concepts, attributes, directories, tree structure, root directory, pathnames, file services etc. Also a number of techniques applied to improve disk system performance have been discussed and in summary these are: disk caching, disk scheduling algorithms (FIFO, SSTF, SCAN, CSCAN, LOOK etc.), types of disk space management (contiguous and non-contiguous-linked and indexed), disk address translation, RAID based on interleaving concept etc. Auxiliary storage management is also considered as it is mainly concerned with allocation of space for files:

3.12 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The major characteristics are:

Data Rate	Disk-2Mbytes/sec
	Keyboard-10-15 bytes/sec
Units of transfer Operation	Disk-read, write, seek
	Printer-write, move, select font
Error conditions	Disk-read errors
	Printer-paper out etc.

2) Device independence refers to making the operating system software and user application independent of the devices attached. This enables the devices to be changed at different executions. Output to the printer can be sent to a disk file. Device drivers act as software interface between these I/O devices and user-level software.

3) In some operating systems like UNIX, the driver code has to be compiled and linked with the kernel object code while in some others, like MS-DOS, device drivers are installed and loaded dynamically. The advantage of the former way is its simplicity and run-time efficiency but its limitation is that addition of a new device requires regeneration of kernel, which is eliminated in the latter technique.

4) In double - buffering the transfers occur at maximum rate hence it sustains device activity at maximum speed, while single buffering is slowed down by buffer to transfer times.

5) The key objectives are to maximize the utilisation of the processor, to operate the devices at their maximum speed and to achieve device independence.

Check Your Progress 2

1) A virtual device is a simulation of an actual device by the operating system. It responds to the system calls and helps in achieving device independence. Example: Print Spooler.

2) Directories enable files to be separated on the basis of users and their applications. Also, they simplify the security and system management problems.

3) Major advantages are:

- This helps to avoid possible file name clashes
- Simplifies system management and installations
- Facilitates file management and security of files
- Simplifies running of different versions of same application.

4) File attributes are: Name, Type (in UNIX), Location at which file is stored on disk, size, protection bits, date and time, user identification etc.

5) Since an elevator continues in one direction until there are no more requests pending and then it reverses direction just like SCAN scheduling.

6) Cluster size= 4*512=2048 bytes

Number of clusters = (40*1,000,000)/2048=19531 approximately

7) (i) File size = 1000 bytes
No. of clusters = 1000/512 = 2(approximately)
Total cluster capacity = 2*512 = 1024 bytes
Wasted space = 1024-1000 = 24 bytes
Percentage Wasted space = (24 / 1024)*100
= 2.3%

(ii) File size = 20,000 bytes
No. of clusters = 20,000/512 = 40(approximately)
Total cluster capacity = 40 * 512 = 20480 bytes
Wasted space = 20480-20,000 = 480 bytes
Percentage Wasted space = (480 / 20480) * 100
= 2.3%

8) An alias is an alternative name for a file, possibly in a different directory. In UNIX, a single inode is associated with each physical file. Alias names are stored as separate items in directories but point to the same inode. It allows one physical file to be referenced by two or more different names or same name in different directory.

3.13 FURTHER READINGS

1) Abraham Silberschatz and Peter Baer Galvin, *Operating System Concepts*, Wiley and Sons (Asia) Publications, New Delhi.

2) H.M.Deitel, *Operating Systems*, Pearson Education Asia Place, New Delhi.

3) Andrew S.Tanenbaum, *Operating Systems- Design and Implementation*, Prentice Hall of India Pvt. Ltd., New Delhi.

4) Colin Ritchie, *Operating Systems incorporating UNIX and Windows*, BPB Publications, New Delhi.

5) J. Archer Harris, *Operating Systems*, Schaum's Outlines, TMGH, 2002, New Delhi.

6) Achyut S Godbole, *Operating Systems*, Second Edition, TMGH, 2005, New Delhi.

7) Milan Milenkovic, *Operating Systems*, TMGH, 2000, New Delhi.

8) D. M. Dhamdhere, *Operating Systems – A Concept Based Approach*, TMGH, 2002, New Delhi.

9) William Stallings, *Operating Systems*, Pearson Education, 2003, New Delhi.

10) Gary Nutt, *Operating Systems – A Modern Perspective*, Pearson Education, 2003, New Delhi.