NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNOLOGY

COURSE CODE: CIT381

COURSE TITLE: File Processing and Management

COURSE GUIDE

CIT381
FILE PROCESSING AND MANAGEMENT

Course Team Ismaila O. Mudasiru (Developer/Writer) - OAU

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Published By:
National Open University of Nigeria

First Printed 2011

ISBN: 978-058-525-7

All Rights Reserved

CONTENTS **PAGE**

Introduction

File Processing and Management is a second semester course. It is a 2-credit course that is available to students offering Bachelor of Science, B. Sc., Computer Science, Information Systems and Allied degrees.

Computers can store information on several different types of physical media. Magnetic tape, magnetic disk and optical disk are the most common media. Each of these media has its own characteristics and physical organisation.

For convenience use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating system maps files onto physical media and accesses these files via the storage devices.

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. File processing refers to an environment in which data are physically organised into files.

The operating system implements the abstract concept of a file by managing mass storage media and the devices which control them. Also, files are normally organised into directories to ease their use. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed. File management is software processes concerned with the overall management of files.

What You Will Learn in this Course

This course consists of units and a course guide. This course guide tells you briefly what the course is about, what course material you will be using and how you can work through these materials. In addition, it advocates some general guidelines for the amount of time you are likely to spend on each unit of the course in order to complete it successfully.

It gives you guidance in respect of your Tutor-Marked Assignments which will be made available in the assignment file. There will be regular tutorial classes that are related to the course. It is advisable for you to attend these tutorial sessions. The course will prepare you for the challenges you will meet in the understanding and application of file processing and management principles.

Course Aims

The aim of the course is not a complex one. CIT381 aims to furnish you with enough knowledge so as to understand the basic principles underlining file processing and management, both from the technical and end-user points of view.

Course Objectives

To achieve the aims set out, the course has a set of objectives. Each unit has specific objectives which are included at the beginning of the unit. You may wish to refer to them during your study to check on your progress. You should always look at the unit objectives after completion of each unit. By doing so, you would know whether you have followed the instruction in the unit.

Below are the comprehensive objectives of the course as a whole. By meeting these objectives, you should have achieved the aims of the course as a whole. In addition to the aims earlier stated, this course sets to achieve some objectives. Thus, after going through the course, you should be able to:

- differentiate between file management and file processing concepts
- explain file naming, extensions and attributes
- explain the relationship between computer or electronic files and operating system
- describe file management architecture, operations, and functions
- explain various file organisation and access methods
- know the relationship between files and directories
- explain techniques of record blocking
- explain how operating systems manage used space in computer memory
- describe the various techniques in improving system performance, reliability, and security
- explain various data validation techniques
- discuss and identify many file processing and management products
- manage files and directories on a Microsoft Windows-based system
- discuss various file sorting, searching, and merging algorithms and applications
- have idea of how programming languages handle or process files for input and output.

Working through this Course

To complete this course, you are required to read each study unit, read the textbooks and read other materials which may be provided by the National Open University of Nigeria.

Each unit contains self-assessment exercises and at certain point in the course you would be required to submit assignments for assessment purposes. At the end of the course there is a final examination. The course should take you about a total of 17 weeks to complete. Below you will find listed all the components of the course, what you have to do and how you should allocate your time to each unit in order to complete the course on time and successfully.

This course entails that you spend a lot time reading. I would advise that you avail yourself the opportunity of comparing your knowledge with that of other learners.

Course Materials

The major components of the course are:

1. Course Guide
2. Study Units
3. Presentation Schedule
4. Tutor-Marked Assignments
5. References/Further Reading

Study Units

The study units in this course are as follows:

Module 1	**File Fundamentals**
Unit 1	Basic File Concepts
Unit 2	File Organisation and Access Methods
Unit 3	File Management
Unit 4	File Directories
Unit 5	File and Directory Operations
Module 2	**File Storage Management**
Unit 1	File Allocation
Unit 2	Record Blocking
Unit 3	Free Space Management
Unit 4	File System Performance and Reliability
Unit 5	File System Security and Integrity

Module 3	**File Processing and Applications**
Unit 1	Data Validation
Unit 2	File Managers
Unit 3	Managing Files in Windows
Unit 4	File Sorting, Searching, and Merging
Unit 5	File Handling in High Level Languages

The first module teaches some basic concepts about files and directories, like file naming, essence of file extensions, attributes, how files are organised on disks, how to access files. The various access rights granted to a user and standard operations that can be performed on electronic files and directories are also discussed.

Module Two discusses how files are allocated in computer memory. It teaches that file allocation method will determine the kind of access, and that a file may be scattered all over the storage disk but the operating systems know how to logically harmonize the pieces together. The methods used to manage unused spaces in storage are also discussed. The techniques used by file system designers in improving system performance, reliability, security and integrity are extensively discussed in this module.

The last module tries to look at file processing and management from a subtle angle. It discusses few of the ways end-users interact with computer file system, from data validation through using commercial software to manage files, operations like sorting, searching, and merging of files and finally to using high level computer programming codes to manipulate electronic files.

Each unit consists of one or two weeks' work and include an introduction, objectives, reading materials, exercises, conclusion, summary, tutor-marked assignments (TMAs), references and other resources. The units direct you to work on exercises related to the required reading. In general, these exercises test you on the materials you have just covered or require you to apply it in some way and thereby assist you to evaluate your progress and to reinforce your comprehension of the material. Together with TMAs, these exercises will help you in achieving the stated learning objectives of the individual units and of the course as a whole.

Presentation Schedule

Your course materials have important dates for the early and timely completion and submission of your TMAs and attending tutorials. You should remember that you are required to submit all your assignments by the stipulated time and date. You should guide against falling behind in your work.

Assessment

There are three aspects to the assessment of the course. First is made up of self-assessment exercises. Second, consists of the tutor-marked assignments and third is the written examination/end of course examination.

You are advised to do the exercises. In tackling the assignments, you are expected to apply information, knowledge and techniques you have gathered during the course. The assignments must be submitted to your facilitator for formal assessment in accordance with the deadline stated in the presentation schedule and the assessment file. The work you submit to your tutor for assessment will count for 30% of your total course mark. At the end of the course, you will need to sit for a final or end of course examination of about three hour duration. This examination will count for 70% of your total course mark.

Tutor-Marked Assignment (TMAs)

The TMA is a continuous assessment component of your course. It accounts for 30% of the total score. You will be given four TMAs to answer. Three of these must be answered before you are allowed to sit for end of course examination. The TMAs would be given to you by your facilitator and should be returned after you have done the assignment. Assignment questions for the units in this course are contained in the assignment file. You will be able to complete your assignments from the information and material contained in your reading, references and study units. However, it is desirable in all degree level of education to demonstrate that you have read and researched more into your references, which will give a wider view point and may provide you with a deeper understanding of the subject.

Make sure that each assignment reaches your facilitator on or before the deadline given in the presentation schedule and assignment file. If for any reason you cannot complete your work on time, contact your facilitator before the assignment is due to discuss the possibility of an extension. Extension will not be granted after the due date unless in exceptional circumstances.

Final Examination and Grading

The end of course examination for File Processing and Management (CIT381) will be for three (3) hours and it has a value of 70% of the total course score. The examination will consist of questions, which will reflect the type of self-testing, practice exercise and tutor-marked assignment problems you have previously encountered. All areas of the course will be assessed.

Use the time between finishing the last unit and sitting for the examination to revise the whole course. You might find it useful to review your self-test, TMAs and comments on them before the examination. The end of course examination covers information from all parts of the course.

Course Marking Scheme

Assignment	**Marks**
Assignment 1 - 4	For assignment, best three marks of the four counts at 10% each, i.e., 30% of Course Marks.
End of Course Examination	70% 0f the overall Course Marks
Total	100% of Course Material

Facilitators/Tutors and Tutorials

There are 16 hours of tutorials provided in support of this course. You will be notified of the dates, time, and location of these tutorials as well as the name and phone number of your facilitator, as soon as you are allocated to a tutorial group.

Your facilitator will mark and comment on your assignments, keep a close watch on your progress and any difficulties you might face and provide assistance to you during the course. You are expected to mail your Tutor-Marked Assignments to your facilitator before the schedule date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not delay to contact your facilitator by telephone or e-mail if you need assistance.

The following might be circumstances in which you would find assistance necessary, hence you would have to contact your facilitator if:

- You do not understand any part of the study or assigned readings
- You have difficulty with self-tests
- You have question or problem with an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only chance to have face to face contact with your course facilitator and to ask questions which may be answered instantly. You can raise any problem encountered in the course of your study.

To have more benefits from course tutorials, you are advised to prepare a list of questions before attending them. You will learn a lot from participating actively in discussions.

Summary

File processing and management is a course that intends to intimate the learner with basic facts on file systems both from technical and end-user point of view. Upon completing this course, you will be equipped with the knowledge of file fundamentals, how operating systems process and manage files, and also the techniques users can use in manipulating files through many commercially available file management software. In addition, you will be able to answer the following types of questions:

- What is the meaning of file management?
- How does operating system distinguish one file from another?
- Explain different operations supported by file management systems.
- What is the relationship between file organisation and file access method?
- Explain directory structure and its examples.
- What is the difference between absolute pathname and relative pathname?
- Outline and discuss briefly the various access rights that can be granted to a file user.
- List some of the strategies that can be used in free space fragmentation
- Compare and contrast different file allocation methods
- Compare and contrast the different techniques of blocking records
- Why is managing space so important?
- List different techniques that can be used in improving system performance
- What are the measures available in improving system performance?

- State some of the design principles for securing a computing environment
- What do you understand by system and program threats?
- What do you understand by “Safe Computing”?
- What is the difference between data validity and data accuracy?
- What are the useful tips that can assist in efficient information storage and retrieval?
- Why merge algorithm is not as popular as sort and search algorithm?
- Explain how memory problem can be averted during sorting operation

Of course, the list of questions that you can answer is not limited to the above list. To gain the most from this course, you should endeavour to apply the principles you have learnt to your understanding of file processing and management in computing.

I wish you success in the course and I hope you find it very interesting.

Course Code	CIT381
Course Title	File Processing and Management

Course Team	Ismaila O. Mudasiru (Developer/Writer) - OAU

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Published By:
National Open University of Nigeria

First Printed 2011

ISBN: 978-058-525-7

All Rights Reserved

CONTENTS PAGE

MODULE 1 FILE FUNDAMENTALS

UNIT 1 BASIC FILE CONCEPTS

CONTENTS

1.0 INTRODUCTION

In most applications, the file is the central element. Before data can be processed by a Computer-Based Information System (CBIS), it must be systematically organised. The most common method is to arrange data into fields, records, files and databases. Files can be considered to be the framework around which data processing revolves. The purpose of this unit is to look at the general concepts before going on to discuss the different methods of organising and accessing them.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define a file
- describe the structure of file system
- list the techniques of naming files and the importance of various extensions
- discuss various attributes of a file.

3.0 MAIN CONTENT

3.1 Elements of a File

A file consists of a number of records. Each record is made up of a number of fields and each field consists of a number of characters.

3.1.1 Logical Components of File

The logical components deal with the real-world objects the data represent. These are field, record and file. However, in today's an information system, files most often exist as parts of database, or organised collections of interrelated data.

3.1.2 Field

A **field** is the basic element of data. An individual field contains a single value, such as an employee's last name, a date, or the value of a sensor reading. It is characterised by its length and data type (e.g., ASCII, string, decimal). Depending on the file design, fields may be fixed length or variable length. In the latter case, the field often consists of two or three subfields: the actual value to be stored, the name of the field, and, in some cases, the length of the field. In other cases of variable-length fields, the length of the field is indicated by the use of special demarcation symbols between fields.

3.1.3 Record

A **record** is a collection of related fields that can be treated as a unit by some application program. For example, an employee record would contain such fields as name, identification number, job designation, date of employment, and so on. Again, depending on design, records may be of fixed length or variable length. A record will be of variable length if some of its fields are of variable length or if the number of fields may vary. In the latter case, each field is usually accompanied by a field name. In either case, the entire record usually includes a length field.

3.1.4 File

A **file** is a collection of related records. The file is treated as a single entity by users and applications and may be referenced by name. Files have names and may be created and deleted. Access control restrictions usually apply at the file level. That is, in a shared system, users and programs are granted or denied access to entire files. In some more sophisticated systems, such controls are enforced at the record or even the field level.

3.2 File Naming

Files are abstraction mechanisms. They provide a way to store information and read it back later. This must be done in a way as to shield the user from the details of how and where the information is stored, and how the disks actually work. When a process creates a file, it gives the file a name. When the process terminates, the file continue to exist, and can be accessed by other processes using its name.

The exact rules for file naming vary somewhat from system to system, but all operating systems allow strings of one to eight letters as legal file names. The file name is chosen by the person creating it, usually to reflect its contents. There are few constraints on the format of the file name: It can comprise the letters A-Z, numbers 0-9 and special characters $ # & + @ ! ( ) - { } ' ` _ ~ as well as space. The only symbols that cannot be used to identify a file are * | < > \ ^ = ? / [ ] ' ; , plus control characters. The main caveat on chosen a file name is that there are different rules for different operating systems that can present problems when files are moved from computer to another. For example, Microsoft Windows is case insensitive, so files like MYEBOOKS, myebooks, MyEbooks are all the same to Microsoft Windows. However, under the UNIX operating system, all three would be different files as, in this instance, file names are case sensitive.

3.2.1 Naming Convention

Usually a file would have two parts with "." separating them. The part on the left side of the period character is called the **main name** while the part on the right side is called the **extension**. A good example of a file name is "course.doc." The main name is **course** while the extension is **doc**. File extension differentiates between different types of files. We can have files with same names but different extensions and therefore we generally refer to a file with its name along with its extension and that forms a complete file name.

3.2.2 File Name Extension

A filename extension is a suffix to the name of a computer file applied to indicate the encoding convention or file format of its contents. In some operating systems (for example UNIX) it is optional, while in some others (such as DOS) it is a requirement. Some operating systems limit the length of the extension (such as DOS and OS/2, to three characters) while others (such as UNIX) do not. Some operating systems (for example RISC OS) do not use file extensions.

The following tables, which are extracted from Microsoft® Encarta (2007), show examples of some common filename extensions:

Table 1: Filename extension of Textual Files

TEXT		
FILE TYPE	**CONTENT**	**APPLICATION**
.html	Hypertext Mark-Up Language, the code of simple Web pages. Usually plain texts file with embedded formatting instructions.	Internet browser such as Internet Explorer, Crazy Browser, Mozilla Firefox and Opera.
.pdf	Portable Document Format, a document presentation format, downloads as binary.	Adobe Acrobat
.rtf	Rich Text Format, a document format that can be shared between different word processors.	Any word processing application
.txt	A plain and simple text file	Any word processing application
.doc **.dot** **.abw** **.lwp**	Word processing files created with popular packages.	Microsoft Word (.doc), the related .dot extension for Microsoft Word Template, Abiword (.abw), and Lotus WordPro (.lwp)

Table 2: Filename extension of Image Files

IMAGES		
FILE TYPE	**CONTENT**	**APPLICATION**
.gif	General Interchange Format, though not the most economical, the most common graphics format not found on the Internet.	Lview and many others
.jpg **.jpeg**	Joint Picture Experts Group, a 24 bit graphic format	Lview and many others
.mpg **.mpeg**	Moving Picture Experts Group, a standard internet movie platform	Sparle, Windows Media Player, Quick Time, and many others
.mov	Quick time Movie, apple Macintosh, native movie platform	Sparle, Windows Media Player, Quick Time, and many others

Table 3: Filename extension of Sound Files

SOUND		
FILE TYPE	**CONTENT**	**APPLICATION**
.mp3	Audio Files on both PC and Mac	Windows Media Player
.wav	Audio Files on PC	Real Player
.ra	Real Audio, a proprietary system for delivering and playing streaming audio on the Web	
.aiff	Audio Files on Mac.	

Table 4: Filename extension of Utility type programme

UTILITIES		
FILE TYPE	**CONTENT**	**APPLICATION**
.ppt	A presentation file (for slide shows)	Microsoft Powerpoint
.xls **.123**	Spreadsheet files	Microsoft Excel, Lotus 123
.mdb	A database file	Microsoft Access

Table 5: Filename extension of other types of files

OTHERS		
FILE TYPE	**CONTENT**	**APPLICATION**
.dll	Dynamic Link Library. This is a compiled set of procedures and/or drivers called by another program.	This is a compiled system file-one that should not be moved or altered
.exe	A DOS/ Windows program or a DOS/ windows Self Extracting Archive	Downloads and launches it in its own temporary directory
.zip .sit .tar	Various popular compression formats for the PC, Macintosh, and UNIX respectively	WinZip, ZipIt, PKzip, and others

SELF-ASSESSMENT EXERCISE 1

1. What is a file?
2. What are the terms commonly used in discussing structure of a file?
3. How can you distinguish one file from another?

3.3 File Attributes

The particular information kept for each file varies from operating system to operating system. No matter what operating system one might be using, files always have certain attributes or characteristics. Different file attributes are discussed as follow.

3.3.1 File Name

The symbolic file name is the only information kept in human-read form. As it is obvious, a file name helps users to differentiate between various files.

3.3.2 File Type

A file type is required for the systems that support different types of files. As discussed earlier, file type is a part of the complete file name. We might have two different files; say “cit381.doc” and “cit381.txt”. Therefore the file type is an important attribute which helps in differentiating between files based on their types. File types indicate which application should be used to open a particular file.

3.3.3 Location

This is a pointer to the device and location on that device of the file. As it is clear from the attribute name, it specifies where the file is stored.

3.3.4 Size

Size attribute keeps track of the current size of a file in bytes, words or blocks. The size of a file is measured in bytes. A floppy disk holds about 1.44 Mb; a Zip disk holds 100 Mb or 250 Mb; a CD holds about 800 Mb; a DVD holds about 4.7 Gb.

3.3.5 Protection

Protection attribute of a file keeps track of the access-control information that controls who can do reading, writing, executing, and so on.

3.3.6 Usage Count

This value indicates the number of processes that are currently using (have opened) a particular file.

3.3.7 Time, Date and Process Identification

This information may be kept for creation, last modification, and last use. Data provided by this attribute is often helpful for protection and usage monitoring. Each process has its own identification number which contains information about file hierarchy.

3.4 Attribute Values

In addition, all operating systems associate other information with each file. The list of attributes varies considerably from system to system. The table below shows some of the possibilities, but other ones also exist. No existing system has all of these, but each is present in some system.

Table 6: Fields and various attribute values

FIELD	MEANING
Protection	Who can access the file and in what way?
Password	Password needed to access the file
Creator	Identity of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write, 1 for read only
Hidden flag	0 for normal, 1 for do not display in listing
System flag	0 for normal file, 1 for system file
Archive flag	0 has been backed up, 1 for needs to be backed up
ASCII/binary file	0 for ASCII file, 1 for binary file
Random access file	0 for sequential access only, 1 for random access
Temporary flag	0 for normal, 1 for delete on process exit
Lock flags	0 for unlocked, nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time file was created
Time of last access	Date and time file was last accessed
Time of last change	Date and time file was last changed
Current size	Number of bytes in the file
Maximum size	Maximum size file may grow

Source: *Modern Operating Systems, 2nd ed. by Andrew S. Tanenbaum (2006).*

The first four attributes relate to the file's protection and tell who may access it and who may not. All kinds of scheme are possible; in some systems the user must present a password to access a file, in which case the password must be one of the attributes.

The flags are bits or short fields that control or enable some specific property. Hidden files, for example, do not appear in listing of the files. The archive flag is a bit that keeps track of whether the file has been backed up. The backup program clears it, and the operating system sets it whenever a file is changed. In this way, the backup program can tell which files need backing up. The temporary flag allows a file to be marked for automatic deletion when the process that created it terminates.

The record length, key position, and key length fields are only present in files whose records can be looked up using a key. They provide the information required to find the keys.

The various times keep track of when the file was created, most recently accessed and most recently modified. These are useful for a variety of purposes. For example, a source file that has been modified after the creation of the corresponding object file needs to be recompiled. These fields provide the necessary information.

The current size tells how big the file is at present. Some mainframe operating systems require the maximum size to be specified when the file is created, to let the operating system reserve the maximum amount of storage in advance. Minicomputers and personal computer systems are clever enough to do without this item.

SELF-ASSESSMENT EXERCISE 2

1. What do you understand by file attributes?
2. List out some attributes a file could possess.

4.0 CONCLUSION

This unit has shed light on the importance of files in computing. The relevance of the use of files cannot be over emphasized and the study on files is a worthwhile one.

5.0 SUMMARY

In this unit, you have learnt that:

- File is the basic unit of storage that enables a computer to distinguish one set of information from another.
- In naming a file, a file would have two parts with a period character separating them. The part on the left side of the period character is called the *main name* while the part on the right side is called the *extension*.
- File extension shows the type of file and the application that the Operating System will use in opening it.
- Files have attributes which vary considerably from system to system. No existing operating system has all of these, but each is present in some systems.
- A file might or might not be stored in human-readable form, but it is invariably the “glue” that binds a conglomeration of instructions, numbers, words, or images into a coherent unit that a

user can retrieve, delete, save, sometimes change, or send to an output device.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain how Operating System differentiates one file from another.
2. What types of file extensions will the following contain:

a. A movie in a Video CD
b. A word processed file
c. A file in a Musical CD
d. A Scanned image
e. A picture taken with a digital camera.

3. Discuss how to name a file and give examples of valid and invalid filenames.

7.0 REFERENCES/FURTHER READING

Davis, W. S. & Rajkumar, T. M. (2001). *Operating System:* A Systematic View (5th Edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). Operating Systems, (3rd Edition). New Jersey: Prentice Hall.

Microsoft® Encarta (2007) http://en.www.wikipedia.com.

Stallings, Williams. (2004). Operating System; *Internal and Design Principles,* (5th Edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd Edition). New Jersey: Prentice Hall.

Williams, B. K. (1999). *Using Information Technology: A Practical Information to Computers and Communications*, (3rd Edition). Boston: McGraw Hill.

UNIT 2 FILE ORGANISATION AND ACCESS METHODS

CONTENTS

1.0 INTRODUCTION

Data files are organised so as to facilitate access to records and to ensure their efficient storage. A trade-off between these two requirements generally exists: if rapid access is required, more storage must be expended to make it possible (for example, by providing indexes to the data records). Access to a record for reading it (and sometimes updating it) is the essential operation on data. On secondary storage devices where files are kept, these are two types of access: sequential and direct.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define file organisation
- list various file organisation methods available
- explain different file organisation methods
- state the differences between one file organisation method from another
- list the merits and demerits of each file organisation style.

3.0 MAIN CONTENT

3.1 File Organisation and Access Methods

In this unit, we use the term *file organisation* to refer to the structure of a file (especially a data file) defined in terms of its components and how they are mapped onto backing store. Any given file organisation supports one or more file access methods. Organisation is thus closely related to but conceptually distinct from access methods. Access method is any algorithm used for the storage and retrieval of records from a data file by determining the structural characteristics of the file on which it is used.

3.1.1 File Organisation Criteria

In choosing a file organisation, several criteria are important:

- Short access time
- Ease of update
- Economy of storage
- Simple maintenance
- Reliability.

The relative priority of these criteria will depend on the applications that will use the file. For example, if a file is only to be processed in batch mode, with all of the records accessed every time, then rapid access for retrieval of a single record is of minimal concern. A file stored on CD-ROM will never be updated, and so ease of update is not an issue. These criteria may conflict. For example, for economy of storage, there should be minimum redundancy in the data. On the other hand, redundancy is a primary means of increasing the speed of access to data. An example of this is the use of indexes.

3.2 File Organisation Methods

The number of alternative file organisations that have been implemented or just proposed is unmanageably large. In this brief survey, we will outline five fundamental organisations. Most structures used in actual systems either fall into one of these categories or can be implemented or a combination of these organisations. The five organisations, the first four of which are depicted in Figure 01, are:

- The pile/serial
- The sequential file
- The indexed sequential file
- The indexed file
- The direct, or hashed, file

Table 7 Summarises relative performance aspects of these five organisations.

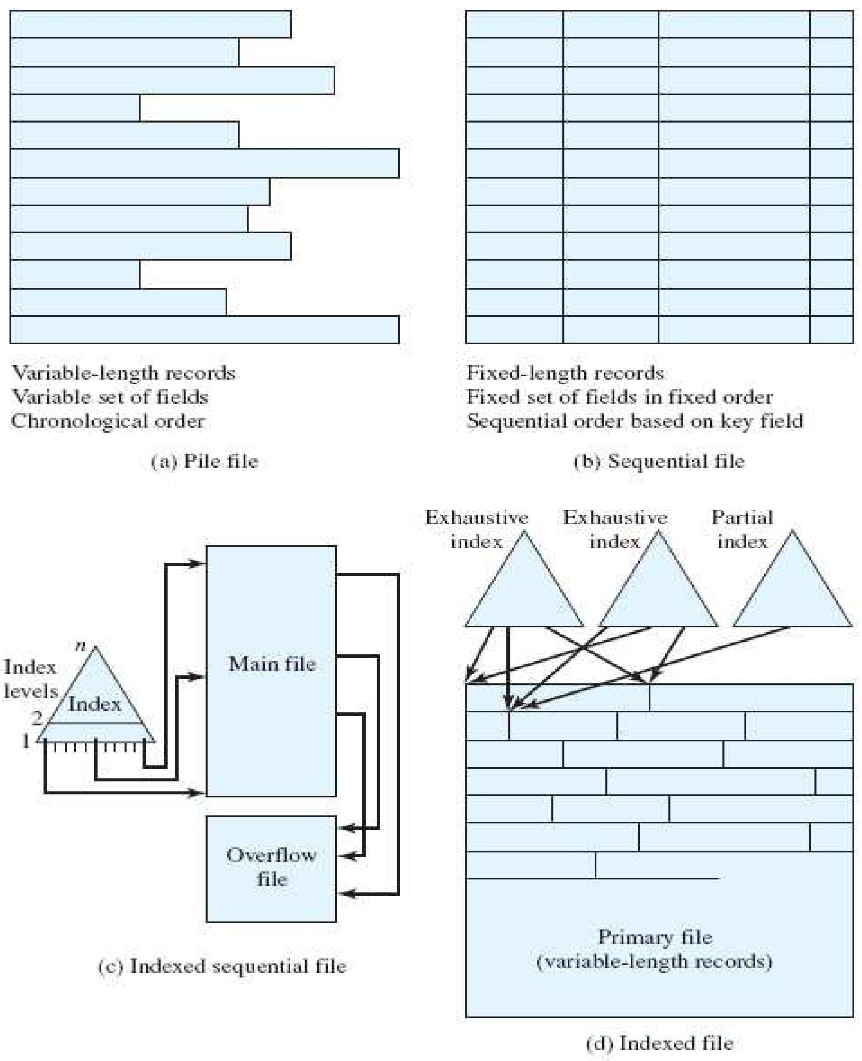
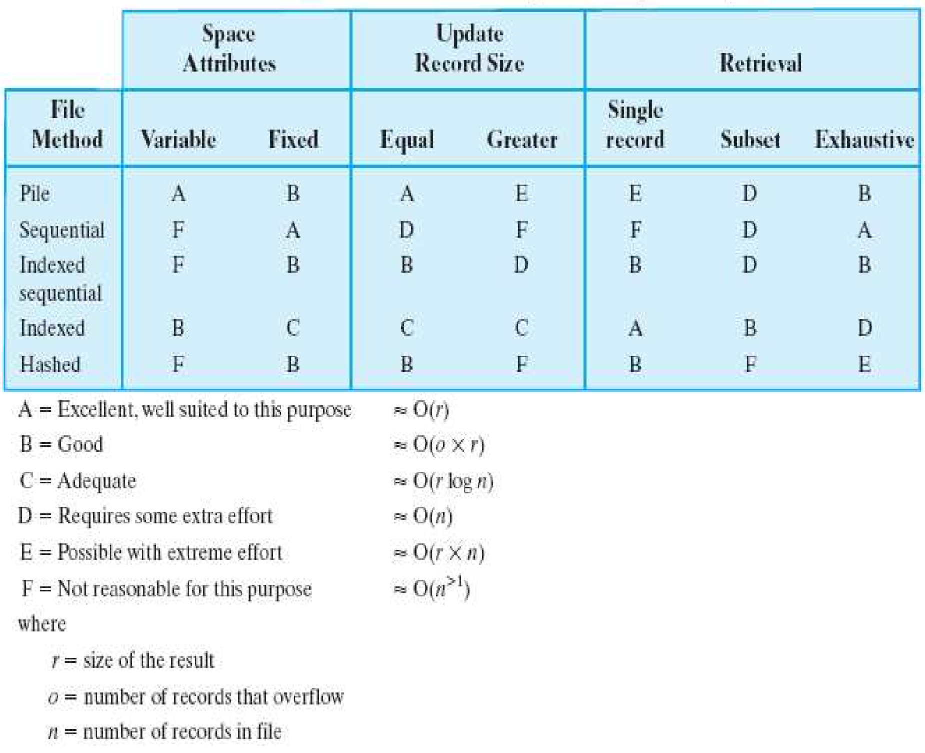


Fig 3: **Common File Organisation (*Source: Operating Systems by Stalling)*.**

Source: *Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

Table 7: Grade of Performances for Five Basic File Organisation



	Space Attributes		Update Record Size		Retrieval		
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excellent, well suited to this purpose	$\approx O(r)$
B = Good	$\approx O(o \times r)$
C = Adequate	$\approx O(r \log n)$
D = Requires some extra effort	$\approx O(n)$
E = Possible with extreme effort	$\approx O(r \times n)$
F = Not reasonable for this purpose	$\approx O(n^{>1})$

where

r = size of the result

o = number of records that overflow

n = number of records in file

Source: *Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

SELF-ASSESSMENT EXERCISE 1

1. What is file organisation?
2. What is the relationship between file organisation and access method?
3. What are the important issues to consider when selecting a file organisation?

3.2.1 The Pile/Serial

The least-complicated form of file organisation may be termed the *pile/serial*. Data are collected in the order in which they arrive. Each record consists of one burst of data. The purpose of the pile/serial is simply to accumulate the mass of data and save it. Records may have different fields, or similar fields in different orders. Thus, each field should be self-describing, including a field name as well as a value. The length of each field must be implicitly indicated by delimiters, explicitly included as a subfield, or known as default for that field type. Because there is no structure to the pile/serial file, record access is by exhaustive search. That is, if we wish to find a record that contains a particular field with a particular value, it is necessary to examine each record in the pile until the desired record is found or the entire file has been searched. If we wish to find all records that contain a particular field or contain that field with a particular value, then the entire file must be searched.

Pile/serial files are encountered when data are collected and stored prior to processing or when data are not easy to organise. This type of file uses space well when the stored data vary in size and structure; is perfectly adequate for exhaustive searches, and is easy to update. However, beyond these limited uses, this type of file is unsuitable for most applications.

3.2.2 The Sequential File

The most common form of file structure is the sequential file. In this file organisation, a fixed format is used for records. All records are of the same length, consisting of the same number of fixed-length fields in a particular order. Because the length and position of each field are known, only the values of fields need to be stored; the field name and length for each field are attributes of the file structure. One particular field, usually the first field in each record, is referred to as the **key field**. The key field uniquely identifies the record; thus key values for different records are always different. Further, the records are stored in key sequence: alphabetical order for a text key, and numerical order for a numerical key.

Sequential files are typically used in batch applications and are generally optimum for such applications if they involve the processing of all the records (e.g., a billing or payroll application).The sequential file organisation is the only one that is easily stored on tape as well as disk. For interactive applications that involve queries and/or updates of individual records, the sequential file provides poor performance. Access requires the sequential search of the file for a key match. If the entire file, or a large portion of the file, can be brought into main memory at one time, more efficient search techniques are possible.

Nevertheless, considerable processing and delay are encountered to access a record in a large sequential file. Additions to the file also present problems. Typically, a sequential file is stored in simple sequential ordering of the records within blocks. That is, the physical organisation of the file on tape or disk directly matches the logical organisation of the file. In this case, the usual procedure is to place new records in a separate pile file, called a log file or transaction file. Periodically, a batch update is performed that merges the log file with the master file to produce a new file in correct key sequence.

An alternative is to organize the sequential file physically as a linked list. One or more records are stored in each physical block. Each block on disk contains a pointer to the next block. The insertion of new records involves pointer manipulation but does not require that the new records occupy a particular physical block position. Thus, some added convenience is obtained at the cost of additional processing and overhead.

3.2.3 The Indexed Sequential File

A popular approach to overcoming the disadvantages of the sequential file is the indexed sequential file. The indexed sequential file maintains the key characteristic of the sequential file: records are organised in sequence based on a key field. Two features are added:

- an **index** to the file to support random access, and
- an **overflow file**.

The index provides a lookup capability to quickly reach the vicinity of a desired record. The overflow file is similar to the log file used with a sequential file but is integrated so that a record in the overflow file is located by following a pointer from its predecessor record.

In the simplest indexed sequential structure, a single level of indexing is used. The index in this case is a simple sequential file. Each record in the index file consists of two fields: a key field, which is the same as the key field in the main file, and a pointer into the main file. To find a specific record, the index is searched to find the highest key value that is equal to or precedes the desired key value. The search continues in the main file at the location indicated by the pointer.

To see the effectiveness of this approach, consider a sequential file with 1 million records. To search for a particular key value will require on average one-half million record accesses. Now suppose that an index containing 1000 entries is constructed, with the keys in the index more or less evenly distributed over the main file. Now it will take on average 500 accesses to the index file followed by 500 accesses to the main file to find the record. The average search length is reduced from 500,000 to 1000.

Additions to the file are handled in the following manner: Each record in the main file contains an additional field not visible to the application, which is a pointer to the overflow file. When a new record is to be inserted into the file, it is added to the overflow file. The record in the main file that immediately precedes the new record in logical sequence is updated to contain a pointer to the new record in the overflow file. If the immediately preceding record is itself in the overflow file, then the pointer in that record is updated. As with the sequential file, the indexed sequential file is occasionally merged with the overflow file in batch mode.

The indexed sequential file greatly reduces the time required to access a single record, without sacrificing the sequential nature of the file. To process the entire file sequentially, the records of the main file are processed in sequence until a pointer to the overflow file is found, then accessing continues in the overflow file until a null pointer is encountered, at which time accessing of the main file is resumed where it left off.

To provide even greater efficiency in access, multiple levels of indexing can be used. Thus the lowest level of index file is treated as a sequential file and a higher-level index file is created for that file. Consider again a file with 1 million records. A lower-level index with 10,000 entries is constructed. A higher-level index into the lower level index of 100 entries can then be constructed. The search begins at the higher-level index (average length = 50 accesses) to find an entry point into the lower-level index. This index is then searched (average length = 50) to find an entry point into the main file, which is then searched (average length = 50). Thus the average length of search has been reduced from 500,000 to 1000 to 150.

3.2.4 The Indexed File

The indexed sequential file retains one limitation of the sequential file: effective processing is limited to that which is based on a single field of the file. For example, when it is necessary to search for a record on the basis of some other attributes than the key field, both forms of sequential file are inadequate. In some applications, the flexibility of efficiently searching by various attributes is desirable.

To achieve this flexibility, a structure is needed that employs multiple indexes, one for each type of field that may be the subject of a search. In the general indexed file, the concept of sequentiality and a single key are abandoned. Records are accessed only through their indexes. The result is that there is now no restriction on the placement of records as long as a pointer in at least one index refers to that record. Furthermore, variable-length records can be employed.

Two types of indexes are used. An exhaustive index contains one entry for every record in the main file. The index itself is organized as a sequential file for ease of searching. A partial index contains entries to records where the field of interest exists. With variable-length records, some records will not contain all fields. When a new record is added to the main file, all of the index files must be updated. Indexed files are used mostly in applications where timeliness of information is critical and where data are rarely processed exhaustively. Examples are airline reservation systems and inventory control systems.

3.2.5 The Direct or Hashed File

The direct or hashed file exploits the capability found on disks to access directly any block of a known address. As with sequential and indexed sequential files, a key field is required in each record. However, there is no concept of sequential ordering here. The direct file makes use of hashing on the key value. Direct files are often used where very rapid access is required, where fixed length records are used, and where records are always accessed one at a time. Examples are directories, pricing tables, schedules, and name lists.

SELF-ASSESSMENT EXERCISE 2

1. List the different file organisation methods.
2. In what situation will a particular file access method be useful?
3. Outline the shortfalls of each of the file organisation.

4.0 CONCLUSION

Although there are two principal types of access: Sequential and direct, many more ways to organise data files exist namely: piles/serial, sequential, indexed sequential, indexed and direct.

5.0 SUMMARY

In this unit, you have learnt that:

- File organisation refers to the logical structuring of the records as determined by the way in which they are accessed.
- Short access time, ease of update, economy of storage, simple maintenance and reliability are important criteria in choosing a file organisation
- Major types of file organisation methods are pile, sequential file, indexed sequential file, indexed file, direct/hashed file.
- File organisation determines the applicable access methods. Access methods are principally sequential and direct.
- Each file organisation method has its peculiar advantages and disadvantages.

6.0 TUTOR-MARKED ASSIGNMENT

1. Make a diagrammatic representation of the different types of file organisation methods.
2. What is the importance of index and overflow file in indexed sequential file?
3. Evaluate the performance of each file organisation method.

7.0 REFERENCES/FURTHER READING

Davis, W. S. & Rajkumar, T.M. (2001). *Operating Systems: A Systematic View* (5th Edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems*, (3rd Edition). New Jersey: Prentice Hall.

Microsoft® Encarta (2007) http://en.www.wikipedia.com.

Stallings, Williams (2004). Operating *Systems; Internal and Design Principles,* (5th Edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd Edition). New Jersey: Prentice Hall.

Williams, B. K. (1999). *Using Information Technology: A Practical Information to* Computers *and Communications*. (3rd Edition). Boston: McGraw Hill.

ftp2.bentley.com/dist/collateral/userstory/Bamberg-1008.pdf

www.encyclopedia.com/doc/1O11-fileorganisation.html

UNIT 3 FILE MANAGEMENT

CONTENTS

1.0 INTRODUCTION

The operating system implements the abstract concept of a file by managing mass storage media and the devices which control them. Also files are normally organized into directories to ease their use. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed. File management is software processes concern with the overall management of files.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define file management
- list the various objectives of file management
- explain the various functions a file management system can perform
- describe file system architecture
- explain the operations that must be supported by file management system.

3.0 MAIN CONTENT

3.1 File Management System

The *file management system, FMS* is the subsystem of an operating system that manages the data storage organisation on secondary storage, and provides services to processes related to their access. In this sense, it interfaces the application programs with the low-level media-I/O (e.g. disk I/O) subsystem, freeing on the application programmers from having to deal with low-level intricacies and allowing them to implement I/O using convenient data-organisational abstractions such as files and records. On the other hand, the FMS services often are the only ways through which applications can access the data stored in the files, thus achieving an encapsulation of the data themselves which can be usefully exploited for the purposes of data protection, maintenance and control.

Typically, the only way that a user or application may access files is through the file management system. This relieves the user or programmer of the necessity of developing special-purpose software for each application and provides the system with a consistent, well-defined means of controlling its most important asset.

3.2 Objectives of File Management System

We can summarise the objectives of a File Management System as follows:

- **Data Management**. An FMS should provide data management services to applications through convenient abstractions, simplifying and making device-independent of the common operations involved in data access and modification.
- **Generality with respect to storage devices**. The FMS data abstractions and access methods should remain unchanged irrespective of the devices involved in data storage.
- **Validity.** An FMS should guarantee that at any given moment the stored data reflect the operations performed on them, regardless of the time delays involved in actually performing those operations. Appropriate access synchronization mechanism should be used to enforce validity when multiple accesses from independent processes are possible.
- **Protection**. Illegal or potentially dangerous operations on the data should be controlled by the FMS, by enforcing a well defined data protection policy.
- **Concurrency**. In multiprogramming systems, concurrent access to the data should be allowed with minimal differences with

respect to single-process access, save for access synchronization enforcement.

- **Performance**. The above functionalities should be offered achieving at the same a good compromise in terms of data access speed and data transferring rate.

3.3 File Management Functions

With respect to meeting user requirements, the extent of such requirements depends on the variety of applications and the environment in which the computer system will be used. For an interactive, general-purpose system, the under listed constitutes a minimal set of requirements:

- Each user should be able to create, delete, read, write, and modify files.
- Each user may have controlled access to other users' files.
- Each user may control what types of accesses are allowed to the user's files.
- Each user should be able to restructure the user's files in a form appropriate to the problem.
- Each user should be able to move data between files.
- Each user should be able to back up and recover the user's files in case of damage.
- Each user should be able to access his or her files by name rather than by numeric identifier.

3.4 File System Architecture

One way of getting a feel for the scope of file management is to look at a depiction of a typical software organisation, as suggested in Figure 2. Of course, different systems will be organised differently, but this organisation is reasonably representative.

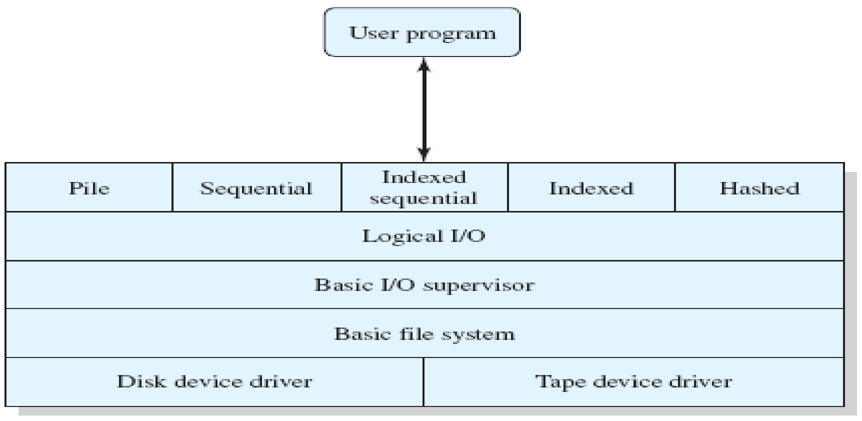


Fig 2: File System Software Architecture

Source: *Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

3.4.1 Device Drivers

At the lowest level, **device drivers** communicate directly with peripheral devices or their controllers or channels. A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request. For file operations, the typical devices controlled are disk and tape drives. Device drivers are usually considered to be part of the operating system.

3.4.2 Basic File System

The next level is referred to as the **basic file system**, or the **physical I/O** level. This is the primary interface with the environment outside of the computer system. It deals with blocks of data that are exchanged with disk or tape systems. Thus, it is concerned with the placement of those blocks on the secondary storage device and on the buffering of those blocks in main memory. It does not understand the contents of the data or the structure of the files involved. The basic file system is often considered part of the operating system.

3.4.3 Basic I/O Supervisor

The **basic I/O supervisor** is responsible for all file I/O initiation and termination. At this level, control structures are maintained that deal with device I/O, scheduling, and file status. The basic I/O supervisor selects the device on which file I/O is to be performed, based on the particular file selected. It is also concerned with scheduling disk and tape accesses to optimize performance. I/O buffers are assigned and secondary memory is allocated at this level. The basic I/O supervisor is part of the operating system.

3.4.4 Logical I/O

Logical I/O enables users and applications to access records. Thus, whereas the basic file system deals with blocks of data, the logical I/O module deals with file records. Logical I/O provides a general-purpose record I/O capability and maintains basic data about files. The level of the file system closest to the user is often termed the **access method**. It provides a standard interface between applications and the file systems and devices that hold the data. Different access methods reflect different file structures and different ways of accessing and processing the data. Some of the most common access methods are shown in Figure 01, and they have been described in the previous unit.

SELF-ASSESSMENT EXERCISE 1

1. What is a file management system?
2. Give examples of devices controlled by device drivers.
3. What are the different functions performed by file management systems?

3.5 Elements of File Management

Another way of viewing the functions of a file system is shown in Figure 03. Let us follow this diagram from left to right. Users and application programs interact with the file system by means of commands for creating and deleting files and for performing operations on files. Before performing any operation, the file system must identify and locate the selected file. This requires the use of some sort of directory that serves to describe the location of all files, plus their attributes. In addition, most shared systems enforce user access control: Only authorised users are allowed to access particular files in particular ways.

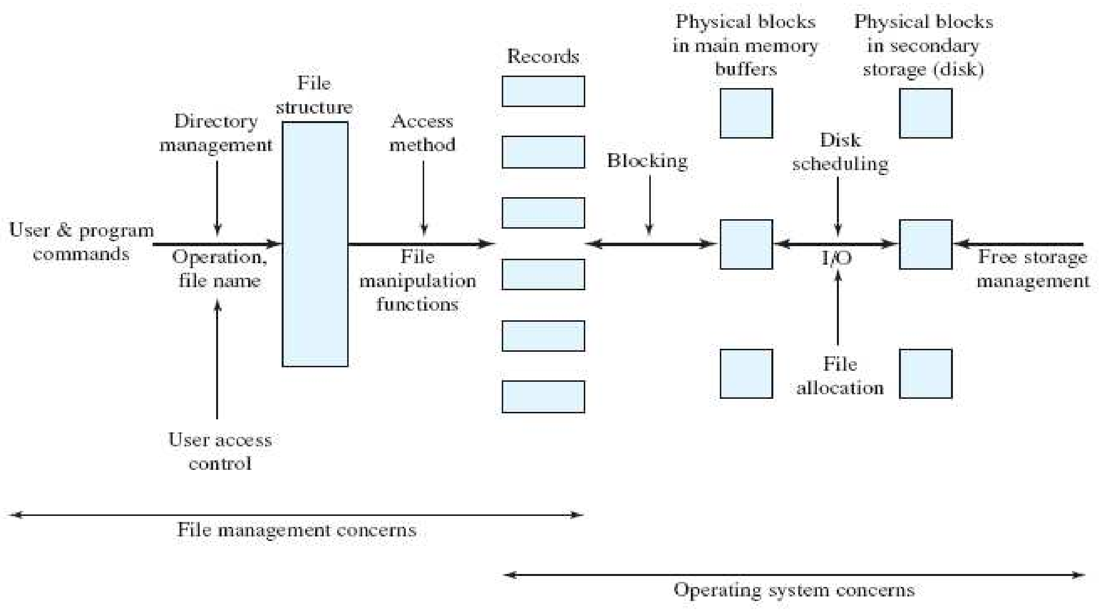


Fig 3: **Elements of File Management (*Source: Operating System by Stalling*)**

***Source**: Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

The basic operations that a user or application may perform on a file are performed at the record level. The user or application views the file as having some structure that organises the records, such as a sequential structure (e.g., personnel records are stored alphabetically by last name). Thus, to translate user commands into specific file manipulation commands, the access method appropriate to this file structure must be employed. Whereas users and applications are concerned with records or fields, I/O is done on a block basis. Thus, the records or fields of a file must be organised as a sequence of blocks for output and unblocked after input. To support block I/O of files, several functions are needed. The secondary storage must be managed. This involves allocating files to free blocks on secondary storage and managing free storage so as to know what blocks are available for new files and growth in existing files. In addition, individual block I/O requests must be scheduled. Both disk scheduling and file allocation are concerned with optimising performance. As might be expected, these functions therefore need to be considered together. Furthermore, the optimisation will depend on the structure of the files and the access patterns. Accordingly, developing an optimum file management system from the point of view of performance is an exceedingly complicated task.

Figure 3 suggests a division between what might be considered the concerns of the file management system as a separate system utility and the concerns of the operating system, with the point of intersection being record processing. This division is arbitrary; various approaches are taken in various systems

3.6 Operations Supported by File Management System

Users and applications wish to make use of files. Typical operations that must be supported include the following:

- **Retrieve _All**

Retrieve all the records of a file. This will be required for an application that must process all of the information in the file at one time. For example, an application that produces a summary of the information in the file would need to retrieve all records. This operation is often equated with the term *sequential processing*, because all of the records are accessed in sequence.

- **Retrieve _One**

This requires the retrieval of just a single record. Interactive, transaction-oriented applications need this operation.

- **Retrieve _Next**

This requires the retrieval of the record that is "next" in some logical sequence to the most recently retrieved record. Some interactive applications, such as filling in forms, may require such an operation. A program that is performing a search may also use this operation.

- **Retrieve _Previous**

Similar to Retrieve_Next, but in this case the record that is "previous" to the currently accessed record is retrieved.

- **Insert _One**

Insert a new record into the file. It may be necessary that the new record fit into a particular position to preserve a sequencing of the file.

- **Delete_One**

Delete an existing record. Certain linkages or other data structures may need to be updated to preserve the sequencing of the file.

- **Update_One**

Retrieve a record, update one or more of its fields, and rewrite the updated record back into the file. Again, it may be necessary to preserve sequencing with this operation. If the length of the record has changed, the update operation is generally more difficult than if the length is preserved.

- **Retrieve_Few**

Retrieve a number of records. For example, an application or user may wish to retrieve all records that satisfy a certain set of criteria.

The nature of the operations that are most commonly performed on a file will influence the way the file is organized, as discussed under file organisation, which in the next unit. It should be noted that not all file systems exhibit the sort of structure discussed in this subsection. On UNIX and UNIX-like systems, the basic file structure is just a *stream* of bytes. For example, a C program is stored as a file but does not have physical fields, records, and so on.

SELF-ASSESSMENT EXERCISE 2

1. List different operations supported by a file management system.
2. What are the objectives of a file management system?

4.0 CONCLUSION

A file management system is that set of system software that provides services to users and applications in the use of files. Typically, the only way that a user or application may access files is through the file management system. This relieves the user or programmer of the necessity of developing special-purpose software for each application and provides the system with a consistent, well-defined means of controlling its most important asset.

5.0 SUMMARY

In this unit, you have learnt that:

- File management system is the subsystem of an operating system that manages the data storage organisation on secondary storage, and provides services to processes related to file access.
- FMS objectives include data management, protection of files against dangerous operations, control over, etc.

- Different operations are supported by FMS which include retrieve_one, retrieve_all, and so on.
- To a user, FMS serves as an interface to file creation and deletion, file ownership and access control, logical identification of data and technical failure prevention as a result of data redundancy.

6.0 TUTOR-MARKED ASSIGNMENT

1. Draw a diagram to represent the file system software architecture.
2. List and explain different operations that are supported by file management system.
3. Discuss in detail the objectives of file management system.

7.0 REFERENCES/FURTHER READING

Davis, W. S. & Rajkumar, T.M. (2001). *Operating Systems: A Systematic View*, (5th edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems*, (3rd edition). New Jersey: Prentice Hall.

Microsoft® Encarta (2007) http://en.www.wikipedia.com.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles*, (5th edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd edition). New Jersey: Prentice Hall.

Williams, B. K. (1999). *Using Information Technology: A Practical Information to Computers and Communications*, (3rd edition). Boston: McGraw Hill.

rpu.dumgal.gov.uk/xpedio/help/esearch/searching_c.htm.

www.ecolore.leeds.ac.uk/xml/materials/file_management.xml

www.webopedia.com/TERM/F/file_management_system.html

UNIT 4 FILE DIRECTORIES

CONTENTS

1.0 INTRODUCTION

Associated with any file management system and collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially those that concern storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines. Although some of the information in directories is available to users and applications, this is generally provided indirectly by system routines.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- state and explain the meaning and importance of file directory
- illustrate the different directory structure possible
- demonstrate how a file can be referenced through different pathways.

3.0 MAIN CONTENT

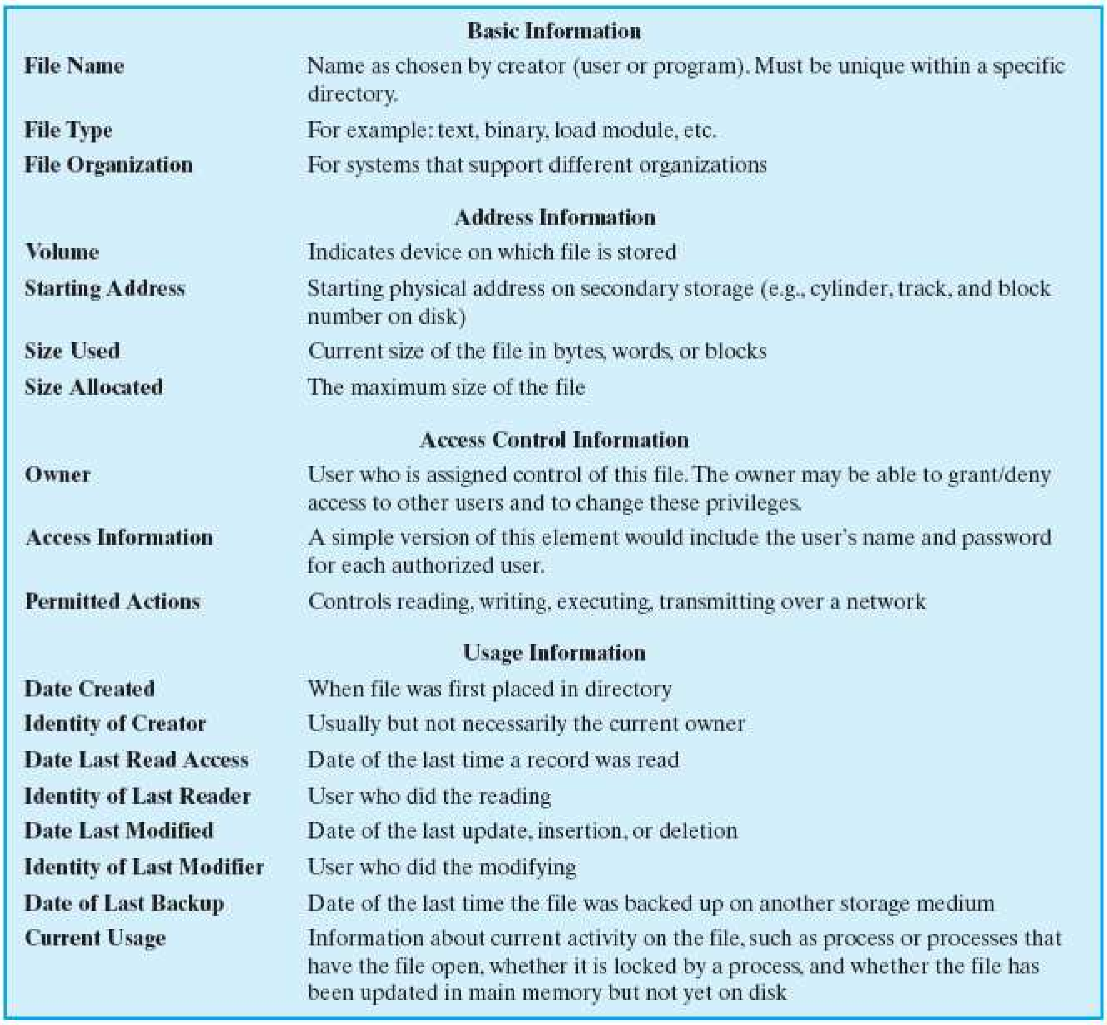
3.1 Concept of File Directory

To keep track of files, the file system normally provides directories, which, in many systems are themselves files. The structure of the directories and the relationship among them are the main areas where file systems tend to differ, and it is also the area that has the most significant effect on the user interface provided by the file system.

3.2 Contents of File Directory

Table 8 on next page suggests the information typically stored in the directory for each file in the system. From the user's point of view, the directory provides a mapping between file names, known to users and applications, and the files themselves. Thus, each file entry includes the name of the file. Virtually all systems deal with different types of files and different file organisations, and this information is also provided. An important category of information about each file concerns its storage, including its location and size. In shared systems, it is also important to provide information that is used to control access to the file. Typically, one user is the owner of the file and may grant certain access privileges to other users. Finally, usage information is needed to manage the current use of the file and to record the history of its usage.

Table 8: Information Elements of a File Directory



	Basic Information
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
	Address Information
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
	Access Control Information
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
	Usage Information
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Source: *Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

3.2.1 File Directory Structure

The number of directories varies from one operating system to another. In this section, we describe the most common schemes for defining the logical structure of a directory. These are:

- Single-Level Directory
- Two-Level Directory
- Tree-Structured Directory
- Acyclic Graph Directory

3.2.2 Single-Level Directory

In a single-level directory system, all the files are placed in one directory. This is very common on single-user operating systems. A single-level directory has significant limitations when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If there are two users who call their data file “cit381note.doc”, then the unique-name rule is violated. Even with a single user, as the number of files increases, it becomes difficult to remember the names of all the files in order to create only files with unique names.

The Figure 5 below shows the structure of a single-level directory system.

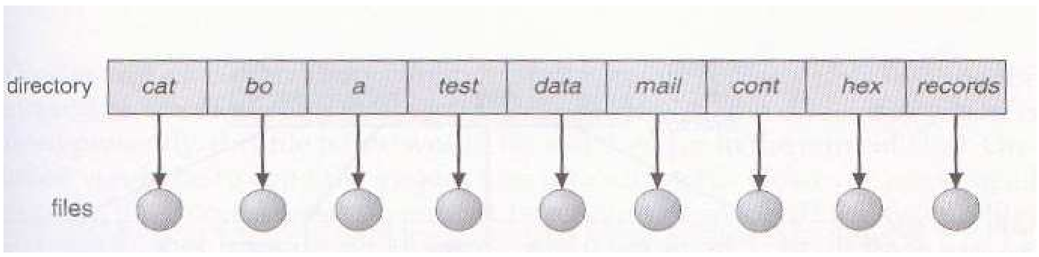


Fig 5: Single Level Directory

Source: *Operating System Concepts with Java, 6th ed. by Abraham Silberschatz and Others. (2004)*

3.2.3 Two-Level Directory

In the two-level directory system, the system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. There are still problems with two-level directory structure. This structure effectively isolates one user from another. This design eliminates name conflicts among users and this is an advantage because users are completely independent, but a disadvantage when the users want to cooperate on some task and access files of other users. Some systems simply do not allow local files to be accessed by other users. It is also unsatisfactory for users with many files because it is quite common for users to want to group their files together in a logical way.

Figure 6 below shows the double-level directory.

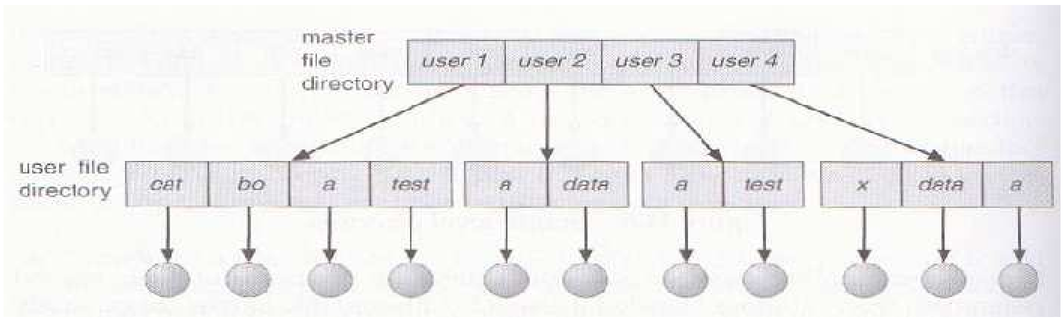


Fig 6: Two-Level Directory

Source: *Operating System Concepts with Java, 6th ed. by Abraham Silberschatz and Others. (2004).*

3.2.4 Tree-Level Structural Directories

In the tree-structured directory, the directory themselves are considered as files. This leads to the possibility of having sub-directories that can contain files and sub-subdirectories. An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory. If a directory is empty, its entry in its containing directory can simply be deleted. However, suppose the directory to be deleted is not empty, but contains several files or sub-directories then it becomes a bit problematic. Some systems will not delete a directory unless it is empty. Thus, to delete a directory, someone must first delete all the files in that directory. If there are any subdirectories, this procedure must be applied recursively to them so that they can be deleted too. This approach may result in a substantial amount of work. An alternative approach is just to assume that when a request is made to delete a directory, all of that directory's files and sub-directories are also to be deleted. This is the most common directory structure.

A typical tree-structured directory system is shown in Figure 7.

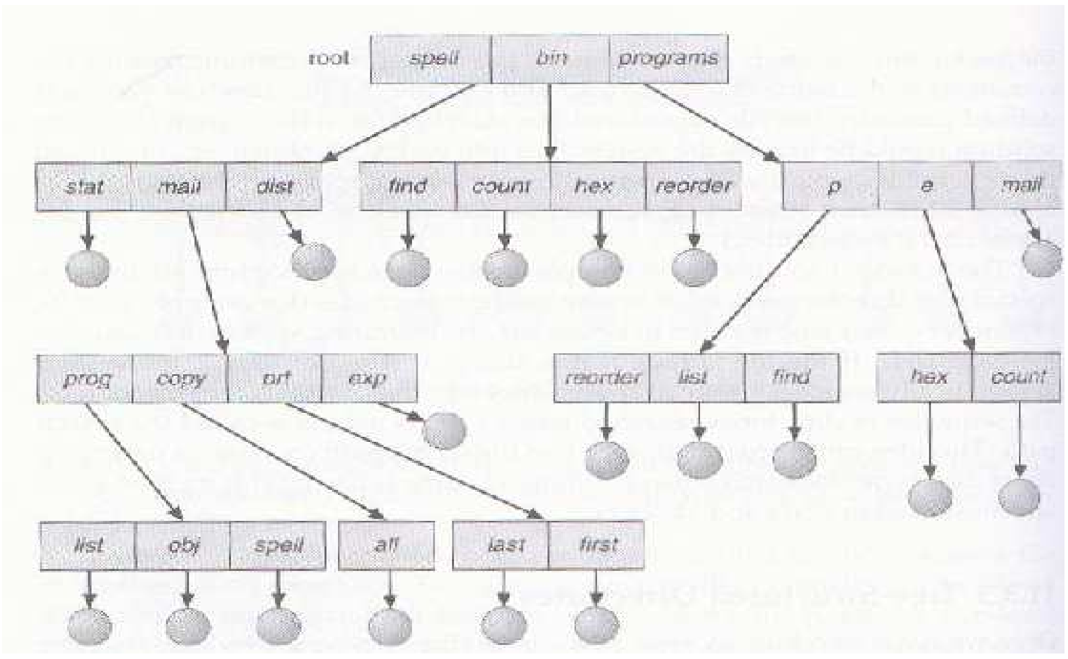


Fig. 7: Tree-Structure System

Source: *Operating System Concepts with Java, 6th ed. by Abraham Silberschatz and Others. (2004)*

3.2.5 Acyclic-Graph Directories

The acyclic directory structure is an extension of the tree-structured directory structure. In the tree-structured directory, files and directories starting from some fixed directory are owned by one particular user. In the acyclic structure, this prohibition is taken out and thus a directory or file under directory can be owned by several users.

The figure 8 shows an acyclic-graph directory structure.

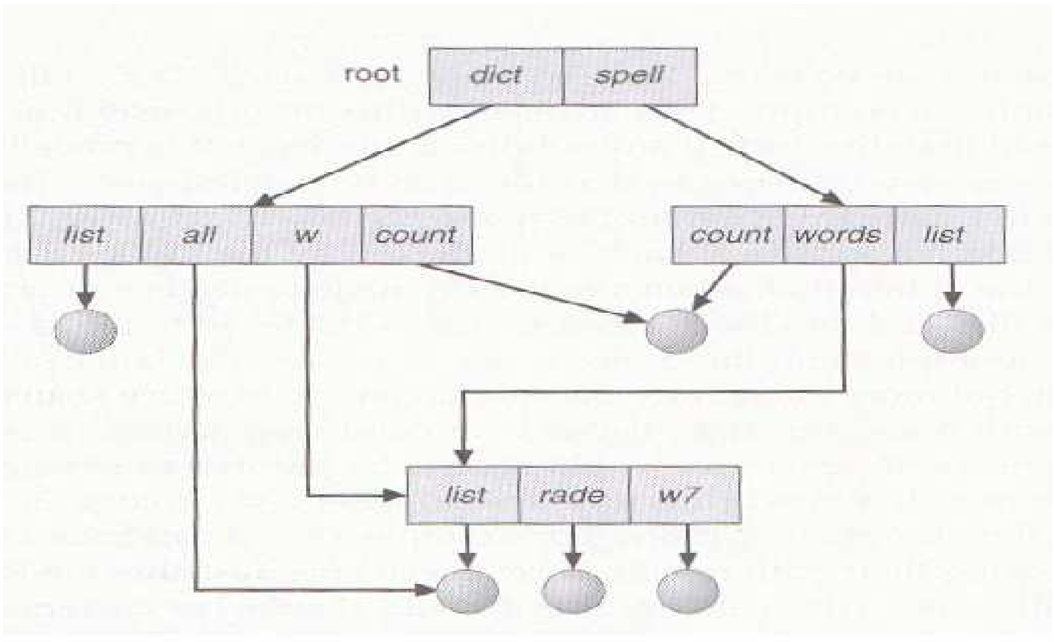


Fig 8: **Acyclic Graph Directory**

Source: *Operating System Concepts with Java, 6th ed. by Abraham Silberschatz and others. (2004).*

SELF-ASSESSMENT EXERCISE 1

1. Explain briefly what you understand by directory structure.
2. Outline the different directory structure.

3.3 Path Names

When a file system is organised as a directory tree, some way is needed for specifying the filenames. The use of a tree-structured directory minimizes the difficulty in assigning unique names. Any file in the system can be located by following a path from the root or master directory down various branches until the file is reached. The series of directory names, culminating in the file name itself, constitutes a **pathname** for the file. Two different methods commonly used are:

- Absolute Path name
- Relative Path name

3.3.1 Absolute Path Name

With this path name, each file is given a path consist of the path from the root directory to the file. As an example, the file in the lower left-hand corner of Figure 09 has the pathname User_B/Word/Unit_A/ABC. The slash is used to delimit names in the sequence. The name of the master directory is implicit, because all paths start at that directory. Note that it is perfectly acceptable to have several files with the same file name, as long as they have unique pathnames, which is equivalent to saying that the same file name may be used in different directories. In this example, there is another file in the system with the file name ABC, but that has the pathname /User_B/Draw/ABC.

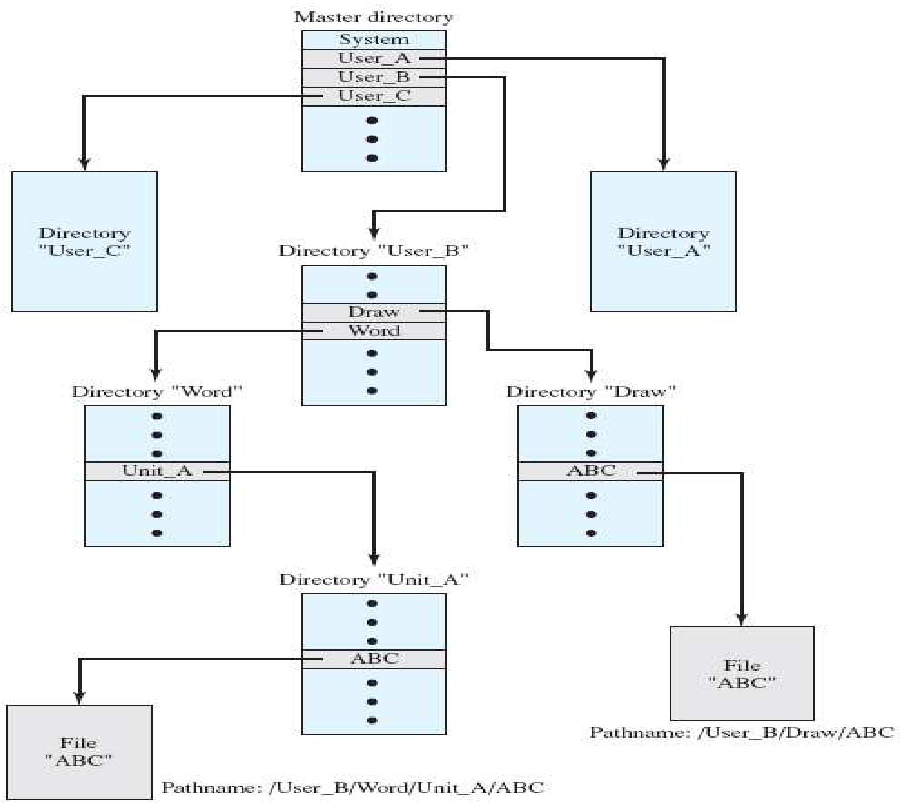


Fig 9: Another Representation of a Tree-Structure Directory

Source: *Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

Note that absolute file names always start at the root directory and are unique. In UNIX the file components of the path are separated by /. In MS-DOS the separator is \. In MULTICS it is >. No matter which character is used, if the first character of the path name is the separator, then the path is absolute.

3.3.2 Relative Path Name

Although the pathname facilitates the selection of file names, it would be awkward for a user to have to spell out the entire pathname every time a reference is made to a file. Typically, an interactive user or a process has associated with it a current directory, often referred to as the **working directory** or **current directory**. Files are then referenced relative to the working directory. For example, if the working directory for user B is “Word,” then the pathname Unit_A/ABC is sufficient to identify the file in the lower left-hand corner of Figure 09. When an interactive user logs on, or when a process is created, the default for the working directory is the user home directory. During execution, the user can navigate up or down in the tree to change to a different working directory.

SELF-ASSESSMENT EXERCISE 2

1. What is a path name?
2. What is the difference between relative and absolute path names?

4.0 CONCLUSION

Directories are used to keep track of files and there are different types of directory structure in use by different operating systems. Each directory structure has its own peculiar way through which a file can be referenced.

5.0 SUMMARY

In this unit, you have learnt that:

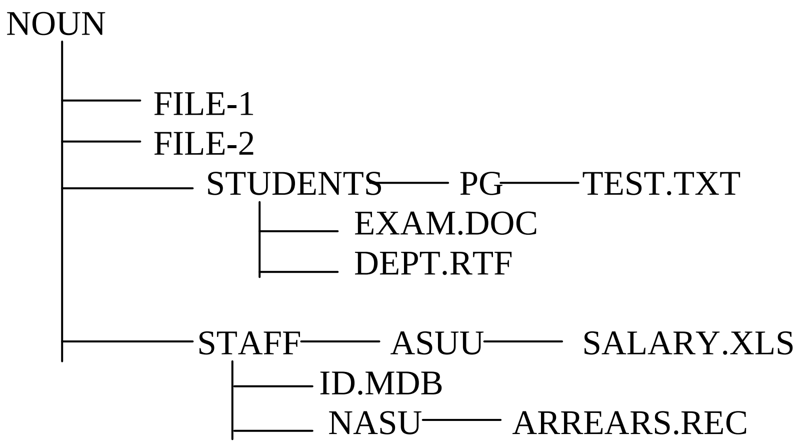
- That directories are to keep track of files
- The directory contains information about the files, including attributes, location, and ownership
- The most common directory structures in use are single-level, two-level, tree-structured, and acyclic graph directories
- A file can be referenced through absolute or relative path name.

6.0 TUTOR-MARKED ASSIGNMENT

1. Construct directory diagrams for the following:

a. Single-Level Directory
b. Two-Level Directory
c. Tree-Structured Directory
d. Acyclic Graph Directory

2. How is it possible to have same file name in a tree directory structure? What is the condition for such?
3. Differentiate between absolute and relative path names with appropriate examples.
4. Use the diagram below to answer the following questions:



a. Write out the absolute pathname for EXAM.DOC, TEST.TXT, SALARY.XLS, and ARREARS.REC
b. Write out the relative pathname for DEPT.RTF and ID.MDB using STUDENTS and STAFF as the working directory respectively.

7.0 REFERENCES/FURTHER READING

Davis, W. S. & Rajkumar, T.M. (2001). *Operating Systems: A Systematic View*, (5th edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems*, (3rd edition). New Jersey: Prentice Hall.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles*, (5th edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd edition). New Jersey: Prentice Hall.

Williams, Brian K. (1999). *Using Information Technology: A Practical Information to Computers and Communications*, (3rd edition). Boston: McGraw Hill.

UNIT 5 FILE AND DIRECTORY OPERATIONS

CONTENTS

1.0 INTRODUCTION

A file is an abstract data type. To define a file properly we need to consider the operations that can be performed on files. In this unit, we discuss the most common system calls relating to files and directories. File sharing is also discussed with regards to access rights and simultaneous access to file.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- list the different types of operations that can be performed on a file
- describe operations on directories
- distinguish between the two sets of operations
- identify and explain the different types of access rights a user of a file may be granted.

3.0 MAIN CONTENT

3.1 Operations on Files and Directories

The operating system provides systems calls to create, write, read, reposition, truncate and delete files. The following sub-units discuss the specific duties a file system must do for each of the following basic file operations.

File Operations

The following are various operations that can take place on file:

a. Creating a File

When creating a file, a space in the file system must be found for the file and then an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the file system.

b. Opening a File

Before using a file, a process must open it. The purpose of the OPEN call is to allow the system to fetch the attributes and list of secondary storage disk addresses into main memory for rapid access on subsequent calls.

c. Closing a File

When all the accesses are finished, the attributes and secondary storage addresses are no longer needed, so the file should be closed to free up internal table space. Many systems encourage this by imposing a maximum number of open files on processes.

d. Writing a File

To write a file, a system call is made specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the location of the file. The directory entry will need to store a pointer to the current block of the file (usually the beginning of the file). Using this pointer, the address of the next block can be computed where the information will be written. The write pointer must be updated ensuring successive writes that can be used to write a sequence of blocks to the file. It is also important to make sure that the file is not overwritten in case of an append operation, i.e. when we are adding a block of data at the end of an already existing file.

e. Reading a File

To read a file, a system call is made that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated directory entry, and the directory will need a pointer to the next block to be read. Once the block is read, the pointer is updated.

3.1.1 Repositioning a File

When repositioning a file, the directory is searched for the appropriate entry, and the current file position is set to a given value. This file operation is also called *file seek.*

a. Truncating a File

The user may erase some contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this operation allows all the attributes to remain unchanged, except the file size.

b. Deleting a File

To delete a file, the directory is searched for the named file. Having found the associated directory entry, the space allocated to the file is released (so it can be reused by other files) and invalidates the directory entry.

c. Renaming a File

It frequently happens that user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can always be copied to a new file with the new name, and the old file then deleted.

d. Appending a File

This call is a restricted form of WRITE call. It can only add data to the end of the file. System that provide a minima set of system calls do not generally have APPEND, but many systems provide multiple ways of doing the same thing, and these systems sometimes have APPEND.

The ten operations described comprise only the minimal set of required file operations. Others may include copying, and executing a file. Also of use are facilities to lock sections of an open file for multiprogramming access, to share sections, and even to map sections into memory or virtual-memory systems. This last function allows a part of the virtual address to be logically associated with section of a file. Reads and writes to that memory region are then treated as reads and writes to the file.

SELF-ASSESSMENT EXERCISE 1

1. Write briefly on the five types of operation that can be performed on a file.
2. What is the difference between appending and writing a file?

3.2 Directory Operations

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory.

a. Create a File

New files need to be created and added to the directory.

b. Delete a File

When a file is no longer needed, we want to remove it from the directory. Only an empty directory can be deleted.

c. Open a File

Directories can be read. For example, to list all files in a directory, a listing program opens the directory to read out the names of all the files it contains. Before a directory can be read, it must be opened.

d. Close a File

When a directory has been read, it should be closed to free up internal table space.

e. Read a File

This call returns the next entry in an open directory. Formerly, it was possible to read directories using the usual READ system call, but that approach has the disadvantage of forcing the programmer to know and deal with the internal structure of directories. In contrast, READDIR always returns one entry in a standard format, no matter which of the possible directory structure is being used.

f. Rename a File

Because the name of a file represents its contents to its uses, the name must be changeable when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.

g. Search for a File

We need to be able to search a directory structure to find the entry for a particular file.

h. List a Directory

We need to list the files in a directory and the contents of the directory entry for each file in the list.

Note that the above list gives the most important operations, but there are a few others as well, for example, for managing the protection information associated with a directory.

3.3 File Sharing

In a multiuser system, there is almost always a requirement for allowing files to be shared among a number of users. Two issues arise: access rights and the management of simultaneous access.

Access Right

The file system should provide a flexible tool for allowing extensive file sharing among users. The file system should provide a number of options so that the way in which a particular file is accessed can be controlled. Typically, users or groups of users are granted certain access rights to a file. A wide range of access rights are in use. The following list is representative of access rights that can be assigned to a particular user for a particular file:

- **None:** The user may not even know of the existence of the file, not to talk of accessing it. To enforce this restriction, the user would not be allowed to read the user directory that contains this file.
- **Knowledge:** The user can determine that the file exists and who its owner is. The user is then able to petition the owner for additional access rights.
- **Execution:** The user can load and execute a program but cannot copy it. Proprietary programs are often made accessible with this restriction.
- **Reading:** The user can read the file for any purpose, including copying and execution. Some systems are able to enforce a distinction between viewing and copying. In the former case, the contents of the file can be displayed to the user, but the user has no means for making a copy.
- **Appending:** The user can add data to the file, often only at the end, but cannot modify or delete any of the file’s contents. This right is useful in collecting data from a number of sources.
- **Updating:** The user can modify, delete, and add to the file’s data. This normally includes writing the file initially, rewriting it

completely or in part, and removing all or a portion of the data. Some systems distinguish among different degrees of updating.

- **Changing protection:** The user can change the access rights granted to other users. Typically, this right is held only by the owner of the file. In some systems, the owner can extend this right to others. To prevent abuse of this mechanism, the file owner will typically be able to specify which rights can be changed by the holder of this right.
- **Deletion:** The user can delete the file from the file system.

These rights can be considered to constitute a hierarchy, with each right implying those that precede it. Thus, if a particular user is granted the updating right for a particular file, then that user is also granted the following rights: knowledge, execution, reading, and appending.

One user is designated as owner of a given file, usually the person who initially created the file. The owner has all of the access rights listed previously and may grant rights to others. Access can be provided to different classes of users:

- **Specific user:** Individual users who are designated by user ID.
- **User groups:** A set of users who are not individually defined. The system must have some way of keeping track of the membership of user groups.
- **All:** All users who have access to this system. These are public files.

Simultaneous Access

When access is granted to append or update a file to more than one user, the operating system or file management system must enforce discipline. A brute-force approach is to allow a user to lock the entire file when it is to be updated. A finer grain of control is to lock individual records during update.

SELF-ASSESSMENT EXERCISE 2

1. How can access to files in networked environment be controlled?
2. What are the different groups of people access can be granted to?

4.0 CONCLUSION

User programs communicate with the operating system and request services from it by making system calls. Corresponding to each call is a library procedure that user program can call. So when a user wants to perform an operation on a file or directory, appropriate system call is invoked. Some rights are put in place by file management system with regards to access to files in a networked environment.

5.0 SUMMARY

In this unit, you have learnt that:

- Different operations can be performed on files and directories
- Examples of operations on files include creating, reading, closing, writing, opening, repositioning, renaming, and others
- Creating, deleting, opening, closing and reading are some of the operations that can be performed on directories
- File system provides a number of options with regards to right to accessing a file so as to control the way in which a particular file is accessed
- More than one user can be granted access to a file but with some level of discipline.

6.0 TUTOR-MARKED ASSIGNMENT

1. Discuss exhaustively different types of operations that a user can perform on directories.
2. Differentiate between the following with regards to files operations

a. Truncating and deleting
b. Writing and appending
c. Reading and repositioning
d. Creating and opening.

3. Outline and discuss briefly the various access rights that can be granted to a file user.

7.0 REFERENCES/FURTHER READING

Davis, W. S. & Rajkumar, T. M. (2001). *Operating Systems: A Systematic View,* (5th edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems,* (3rd edition). New Jersey: Prentice Hall.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles,* (5th edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System,* (3rd edition). New Jersey: Prentice Hall.

Williams, B. K. (1999). *Using Information Technology: A Practical Information to Computers and Communications,* (3rd edition). Boston: McGraw Hill.

MODULE 2 FILE STORAGE MANAGEMENT

UNIT 1 FILE ALLOCATION

CONTENTS

1.0 INTRODUCTION

The main purpose of a computer system is to execute programs. Those programs together with the data they access must be in main memory (at least partially) during execution. Since main memory is usually too small to accommodate all the data and programs permanently, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disk as the primary on-line storage medium for information (both programs and data).

We explore ways to allocate disk space.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain the issues involved in file allocation
- differentiates between the different types of file allocation methods
- list the advantages and disadvantages of each of file allocation methods
- clearly distinguish difference between dynamic allocation and preallocation methods.

3.0 MAIN CONTENT

3.1 File Allocation

In allocating disk space, several issues are involved:

- When a new file is created, is the maximum space required for the file allocated at once?
- Is space allocated to a file as one or more contiguous units? We shall refer to these units as portions. That is, a **portion** is a contiguous set of allocated blocks. The size of a portion can range from a single block to the entire file. What size of portion should be used for file allocation?
- What sort of data structure or table is used to keep track of the portions assigned to a file? An example of such a structure is a **file allocation table (FAT)**, found on DOS and some other systems.

Let us examine these issues in turn.

3.1.1 Preallocation versus Dynamic Allocation

A pre-allocation policy requires that the maximum size of a file be declared at the time of the file creation request. In a number of cases, such as program compilations, the production of summary data files, or the transfer of a file from another system over a communications network, this value can be reliably estimated. However, for many applications, it is difficult if not impossible to estimate reliably the maximum potential size of a file. In those cases, users and application programmers would tend to overestimate file size so as not to run out of space. This clearly is wasteful from the point of view of secondary storage allocation. Thus, the dynamic allocation which allocates space to a file in portions as needed.

3.2 Portion Size

The second issue listed is that of the size of the portion allocated to a file. At one extreme, a portion large enough to hold the entire file is allocated. At the other extreme, space on the disk is allocated one block at a time. In choosing a portion size, there is a tradeoff between efficiency from the point of view of a single file versus overall system efficiency.

Below is a list of four items to be considered in the tradeoff:

- Contiguity of space increases performance, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented operating system.
- Having a large number of small portions increases the size of tables needed to manage the allocation information.
- Having fixed-size portions (for example, blocks) simplifies the reallocation of space.
- Having variable-size or small fixed-size portions minimizes waste of unused storage due to over-allocation.

Of course, these listed items interact and must be considered together. The result is that there are two major alternatives: variable, large contiguous portions and blocks.

a. Variable, Large Contiguous Portions

This will provide better performance. The variable size avoids waste, and the file allocation tables are small. However, space is hard to reuse.

b. Blocks

Small fixed portions provide greater flexibility. They may require large tables or complex structures for their allocation. Contiguity has been abandoned as a primary goal; blocks are allocated as needed.

Both options are compatible with pre-allocation and dynamic allocation. In the case of variable, large contiguous portions, a file is pre-allocated one contiguous group of blocks. This eliminates the need for a file allocation table; all that is required is a pointer to the first block and the number of blocks allocated. In the case of blocks, all of the portions required are allocated at one time. This means that the file allocation table for the file will remain of fixed size, because the number of blocks allocated is fixed.

With variable-size portions, we need to be concerned with the fragmentation of free space. The following are possible alternative strategies:

- **First fit:** Choose the first unused contiguous group of blocks of sufficient size from a free block list.
- **Best fit:** Choose the smallest unused group that is of sufficient size.
- **Nearest fit:** Choose the unused group of sufficient size that is closest to the previous allocation for the file to increase locality.

It is not clear which strategy is best. The difficulty in modeling alternative strategies is that so many factors interact, including types of files, pattern of file access, degree of multiprogramming, other performance factors in the system, disk caching, disk scheduling, and so on.

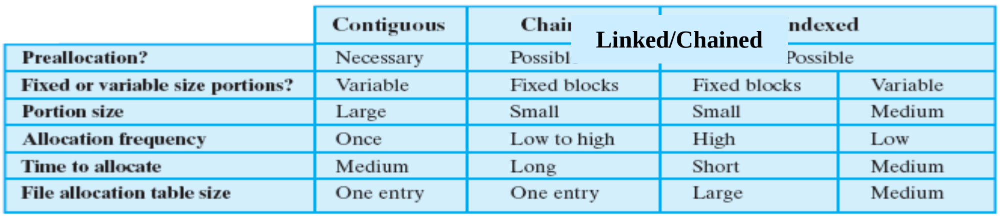
SELF-ASSESSMENT EXERCISE 1

1. What are some of the issues involved in file allocation?
2. What do you understand by pre-allocation policy?
3. List some of the strategies that can be used in free space fragmentation.

3.3 File Allocation Methods

The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilised effectively and file can be accessed quickly. Three major methods of allocating disk space are in wide use: contiguous, linked and indexed. Table 9 summarises some of the characteristics of each method.

Table 9 File Allocation Methods



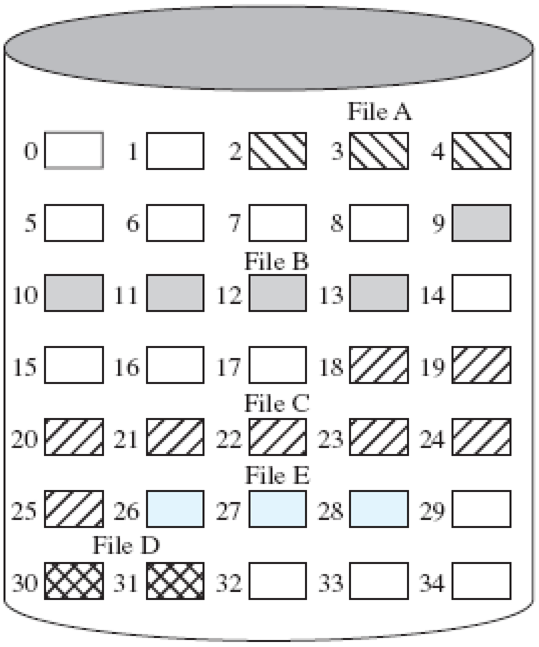
	Contiguous	Chai Linked/Chained	ndexed	
Preallocation?	Necessary	Possibl	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

***Source**: Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

3.3.1 Contiguous Allocation

With **contiguous allocation**, a single contiguous set of blocks is allocated to a file at the time of file creation (Figure 11). Thus, this is a pre-allocation strategy, using variable-size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. Contiguous allocation is the best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block.

For example, if a file starts at block *b*, and the *i*th block of the file is wanted, its location on secondary storage is simply $b + i - 1$. Contiguous allocation presents some problems. External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. From time to time, it will be necessary to perform a compaction algorithm to free up additional space on the disk (Figure 12). Also, with preallocation, it is necessary to declare the size of the file at the time of creation, with the problems mentioned earlier.

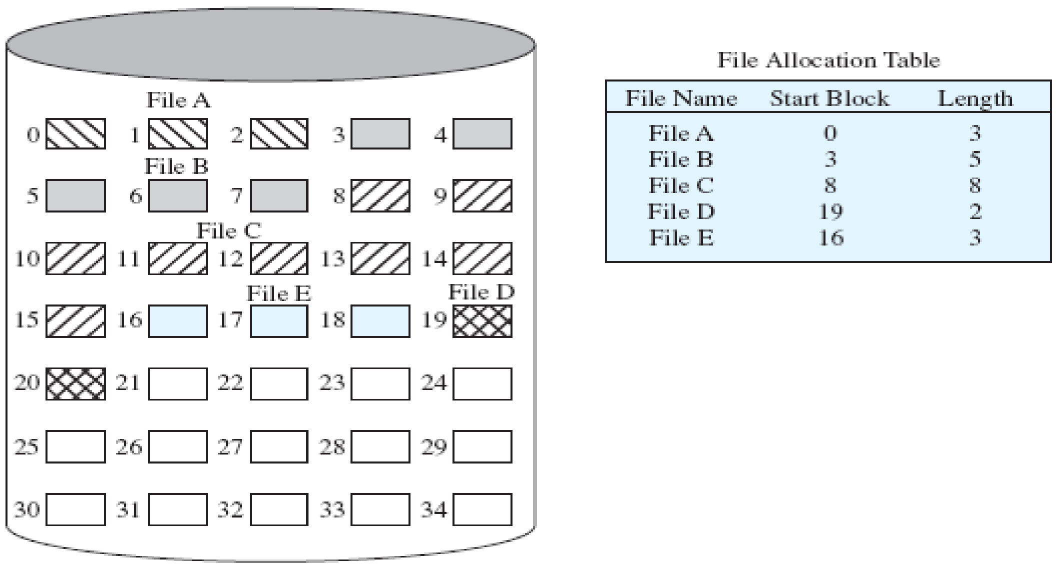


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Fig 11: Contiguous File Allocation

***Source**:* *Operating Systems; Internal and Design Principles*, 5th ed. By William Stallings (2004).



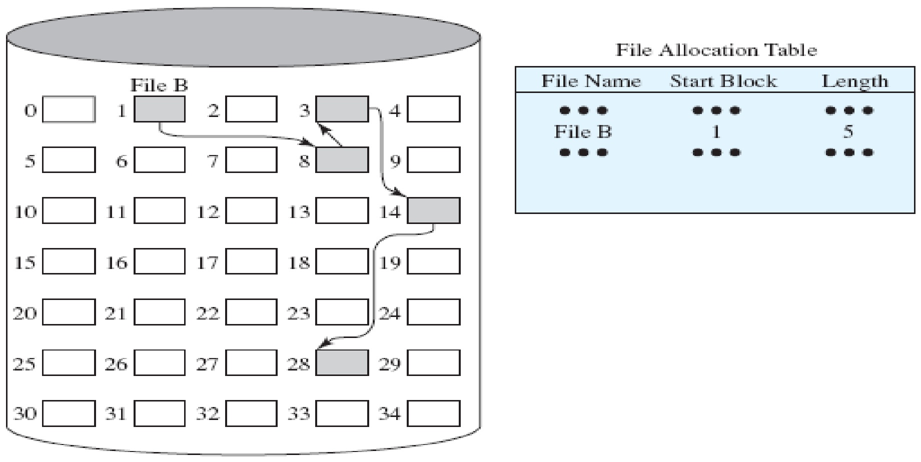
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Fig 12: Contiguous File Allocation (After Compaction)

***Source**: Operating Systems; Internal and Design Principles, 5th ed. By William Stallings (2004).*

3.3.2 Linked/Chained Allocation

At the opposite extreme from contiguous allocation is **chained allocation** (Figure 13). Typically, allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again, the file allocation table needs just a single entry for each file, showing the starting block and the length of the file.



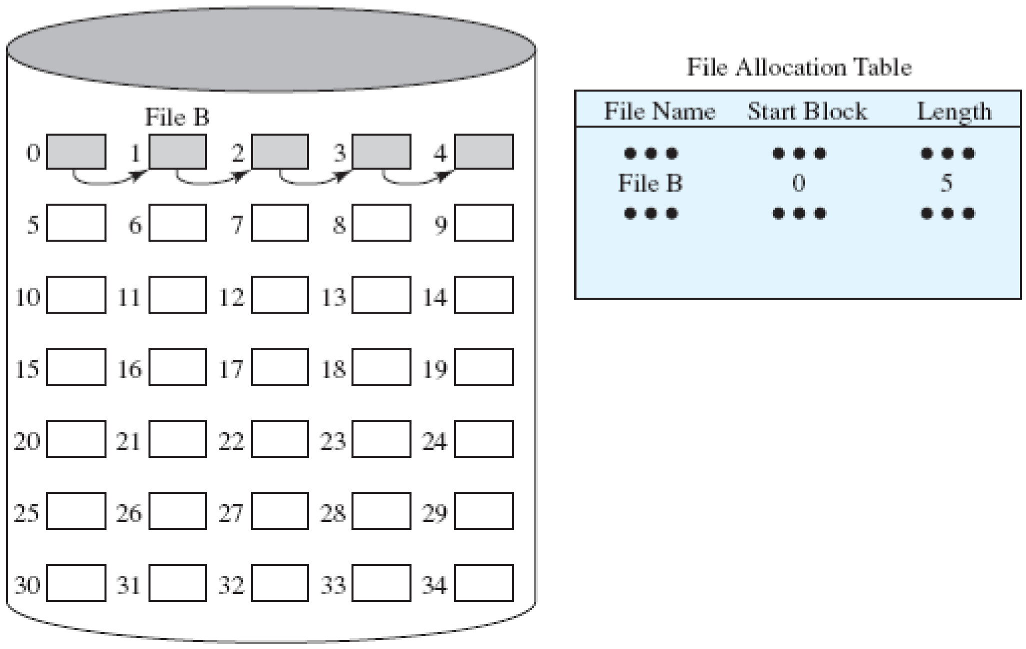
File Name	Start Block	Length
● ● ●	● ● ●	● ● ●
File B	1	5
● ● ●	● ● ●	● ● ●

Fig 13: Linked/Chained File Allocation

***Source**: Operating Systems; Internal and Design Principles, 5th ed. By William Stallings (2004).*

Although pre-allocation is possible, it is more common simply to allocate blocks as needed. The selection of blocks is now a simple matter: any free block can be added to a chain. There is no external fragmentation to worry about because only one block at a time is needed. This type of physical organisation is best suited to sequential files that are to be processed sequentially. To select an individual block of a file requires tracing through the chain to the desired block.

One consequence of linking/chaining, as described so far, is that there is no accommodation of the principle of locality. Thus, if it is necessary to bring in several blocks of a file at a time, as in sequential processing, then a series of accesses to different parts of the disk are required. This is perhaps a more significant effect on a single-user system but may also be of concern on a shared system. To overcome this problem, some systems periodically consolidate files.



File Name	Start Block	Length
•••	•••	•••
File B	0	5
•••	•••	•••

Fig 14: Chained File Allocation (After Consolidation)

***Source**: Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

3.3.3 Indexed Allocation

Indexed allocation addresses many of the problems of contiguous and linked/chained allocation. In this case, the file allocation table contains a separate one-level index for each file; the index has one entry for each portion allocated to the file. Typically, the file indexes are not physically stored as part of the file allocation table. Rather, the file index for a file is kept in a separate block and the entry for the file in the file allocation table points to that block. Allocation may be on the basis of either fixed-size blocks (Figure 15) or variable-size portions (Figure 16). Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size portions improves locality. In either case, file consolidation may be done from time to time. File consolidation reduces the size of the index in the case of variable-size portions, but not in the case of block allocation. Indexed allocation supports both sequential and direct access to the file and thus is the most popular form of file allocation.

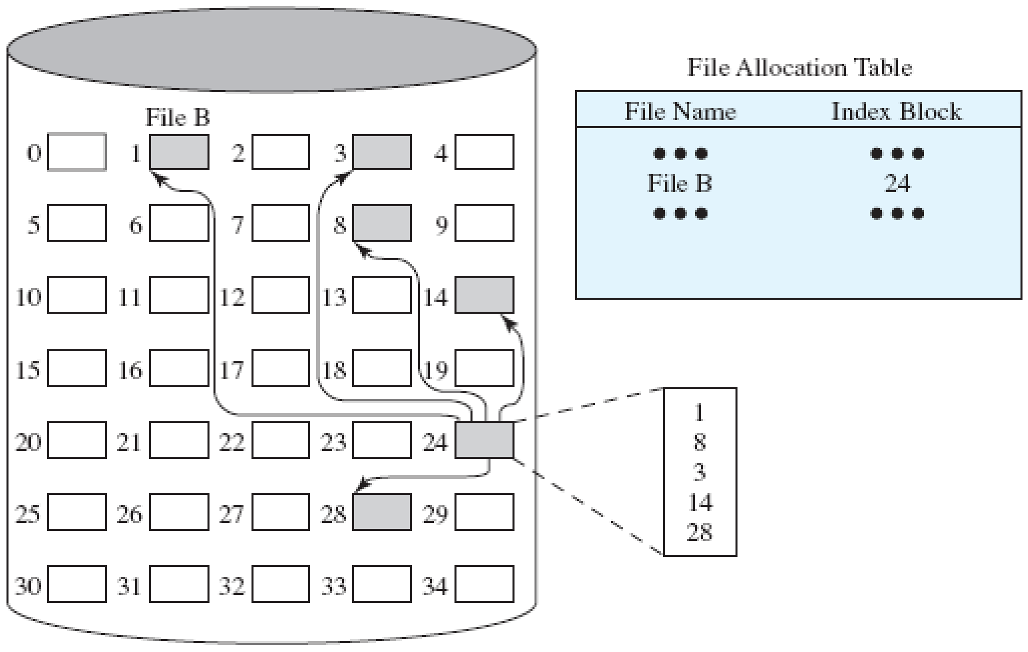


Fig 15: Indexed Allocation with Block Portions

***Source**: Operating Systems; Internal and Design Principles, 5th ed. by William Stallings (2004).*

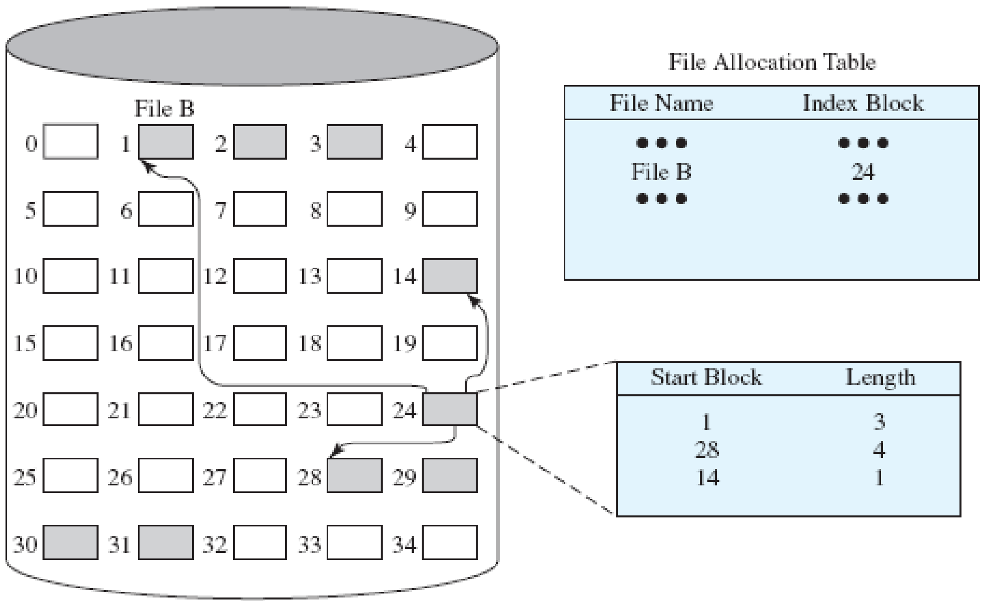


Fig 16: **Indexed Allocation with Variable-Length Portions**

Source*:* *Operating Systems; Internal and Design Principles,* 5th ed. By William Stallings (2004).

SELF-ASSESSMENT EXERCISE 2

1. Relate size of file to be allocated to disk space.
2. List the different methods of file allocation you know.
3. What is FAT?

4.0 CONCLUSION

There are many key issues of interest in the study of allocation of disk blocks to files. A proper understanding of the different methods of file allocation available will go a long way in user's appreciation of the file handling capabilities of any type of operating system.

5.0 SUMMARY

In this unit, you have learnt that:

- A pre-allocation policy requires that the maximum size of a file be declared at the time of the file creation request.
- In choosing a portion size, there are tradeoffs between efficiency from the point of view of a single file versus overall system efficiency.
- The methods of file allocation that are in common use are contiguous, linked/chained, and indexed.
- Each file allocation method has its peculiar merits and demerits.

6.0 TUTOR-MARKED ASSIGNMENT

1. List and briefly discuss any three file allocation methods.
2. What are the different strategies possible in free space fragmentation and why is it that none of the strategies can be considered as the best?
3. What are the tradeoffs to consider in allocating memory blocks to file?

7.0 REFERENCES/FURTHER READING

Davis, W S. & Rajkumar, T. M. (2001). *Operating Systems: A Systematic View,* (5th edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems,* (3rd Edition).New Jersey: Prentice Hall.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles,* (5th Edition). New Jersey: Prentice Hall,

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd Edition). New Jersey: Prentice Hall.

Williams, B. K. (1999). *Using Information Technology: A Practical Information to Computers and Communications*, (3rd Edition). Boston: McGraw Hill.

UNIT 2 RECORD BLOCKING

CONTENTS

1.0 INTRODUCTION

It has been said that file is a body of stored data or information in an electronic format. Almost all information stored on computers is in the form of files. Files reside on mass storage devices such as hard disks, optical disks, magnetic tapes, and floppy disks. When the Central Processing Unit (CPU) of a computer needs data from a file, or needs to write data to a file, it temporarily stores the file in its main memory, or Random Access Memory (RAM), while it works on the data.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define record blocking
- list the different types of record blocking techniques
- discuss the merits and demerits of each record blocking style.

3.0 MAIN CONTENT

3.1 Record Blocking

As indicated in Figure 03, records are the logical unit of access of a structured file, whereas blocks are the unit of I/O with secondary storage. For I/O to be performed, records must be organized as blocks.

There are several issues to consider.

First, should blocks be of fixed or variable length? On most systems, blocks are of fixed length. This simplifies I/O, buffer allocation in main memory, and the organisation of blocks on secondary storage.

Next, what should the relative size of a block be compared to the average record size? The tradeoff is this: The larger the block, the more records that are passed in one I/O operation. If a file is being processed or searched sequentially, this is an advantage, because the number of I/O operations is reduced by using larger blocks, thus speeding up processing. On the other hand, if records are being accessed randomly and no particular locality of reference is observed, then larger blocks result in the unnecessary transfer of unused records.

However, combining the frequency of sequential operations with the potential for locality of reference, we can say that the I/O transfer time is reduced by using larger blocks. The competing concern is that larger blocks require larger I/O buffers, making buffer management more difficult.

3.1.1 Methods of Blocking

Given the size of a block, there are three methods of blocking that can be used, namely; fixed, variable-length and variable-length unspanned spanning.

a. Fixed Blocking

Fixed-length records are used, and an integral number of records are stored in a block. There may be unused space at the end of each block. This is referred to as internal fragmentation. File fragmentation is defined as a condition in which files are broken apart on disk into small, physically separated segments.

b. Variable-Length Spanned Blocking

Variable-length records are used and are packed into blocks with no unused space. Thus, some records must span two blocks, with the continuation indicated by a pointer to the successor block.

c. Variable-Length Unspanned Blocking

Variable-length records are used, but spanning is not employed. There is wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.

SELF-ASSESSMENT EXERCISE 1

1. Give examples of storage devices on which data might be stored
2. How can buffer management be made difficult?

The following figure 17 illustrates these methods assuming that a file is stored in sequential blocks on a disk.

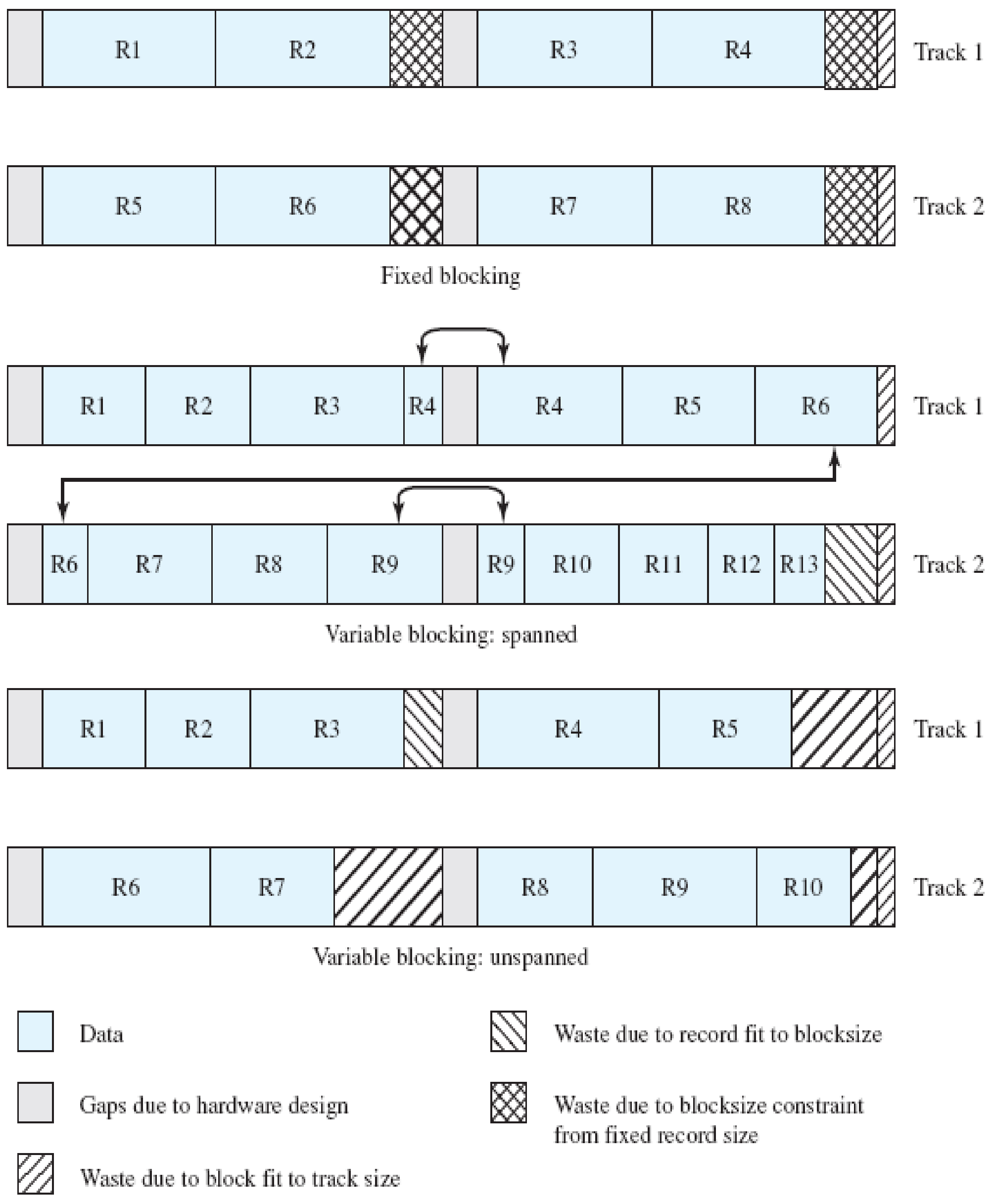

R1
R2
R3
R4
Track 1
R5
R6
R7
R8
Track 2
Fixed blocking
R1
R2
R3
R4
R4
R5
R6
Track 1
R6
R7
R8
R9
R9
R10
R11
R12
R13
Track 2
Variable blocking: spanned
R1
R2
R3
R4
R5
Track 1
R6
R7
R8
R9
R10
Track 2
Variable blocking: unspanned
Data
Gaps due to hardware design
Waste due to block fit to track size
Waste due to record fit to blocksize
Waste due to blocksize constraint from fixed record size

Fig 17: **Record Blocking Methods**

***Source**:* *Operating Systems; Internal and Design Principles, 5th ed. By William Stallings (2004).*

The figure assumes that the file is large enough to span two tracks. The effect would not be changed if some other file allocation schemes were used as discussed under secondary storage management in the next unit.

Fixed blocking is the common mode for sequential files with fixed-length records. Variable-length spanned blocking is efficient of storage and does not limit the size of records. However, this technique is difficult to implement. Records that span two blocks require two I/O operations, and files are difficult to update, regardless of the organisation. Variable-length unspanned blocking results in wasted space and limits record size to the size of a block.

The record-blocking technique may interact with the virtual memory hardware, if such is employed. In a virtual memory environment, it is desirable to make the page as the basic unit of transfer. Pages are generally quite small, so that it is impractical to treat a page as a block for unspanned blocking. Accordingly, some systems combine multiple pages to create a larger block for file I/O purposes.

SELF-ASSESSMENT EXERCISE 2

1. What do you understand by blocking of records?
2. What is the advantage of fixed blocking over variable-length blocking?

4.0 CONCLUSION

A file consists of a collection of blocks and the operating system is responsible for allocating blocks to files.

7.0 SUMMARY

In this unit, you have learnt that:

- Records are the logical unit of access of a structured file, whereas blocks are the unit of I/O with secondary storage
- There are three methods of blocking that can be used. They are fixed, variable-length and variable-length unspanned spanning.

8.0 TUTOR-MARKED ASSIGNMENT

1. Compare and contrast the different techniques of blocking records.
2. Draw a well labeled diagram to illustrate the types of records blocking.

7.0 REFERENCES/FURTHER READING

Claybrook, Billy G. (1991). *File Management Techniques.* New York: John Wiley.

Davis, W. S. & Rajkumar, T.M. (2001). *Operating Systems; A Systematic View*, (5th Edition). New Jersey: Addison-Wesley.

Grosshans, D. (1986). *File Systems; Design and Implementation*. New Jersey: Prentice Hall.

Livadas, P. (1990). File Structures: Theory and Practice. New Jersey: Prentice Hall.

Silberschatz, Abraham. (2004). *Operating System Concepts with Java,* (6th Edition). New Jersey: John Wiley.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles*, (5th Edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System*, (3rd Edition). New Jersey: Prentice Hall.

UNIT 3 FREE SPACE MANAGEMENT

CONTENTS

1.0 INTRODUCTION

Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files, if possible.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- state reasons why management of space is important in file system management
- list the various techniques of managing space
- explain the concepts of volume and disk quotas with respect to file management.

3.0 MAIN CONTENT

3.1 Free Space Management

Files are normally stored on disk, so management of disk space is a major concern to the file system designers. To keep track of free disk space, the system maintains a free space list. The free space list records all disk blocks that are free – those that are not allocated to some file or directory.

To create a file, we search the free space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list. The free-space list, despite its name, might not be implemented as a list, as we shall discuss.

3.1.1 Techniques Used in Space Management

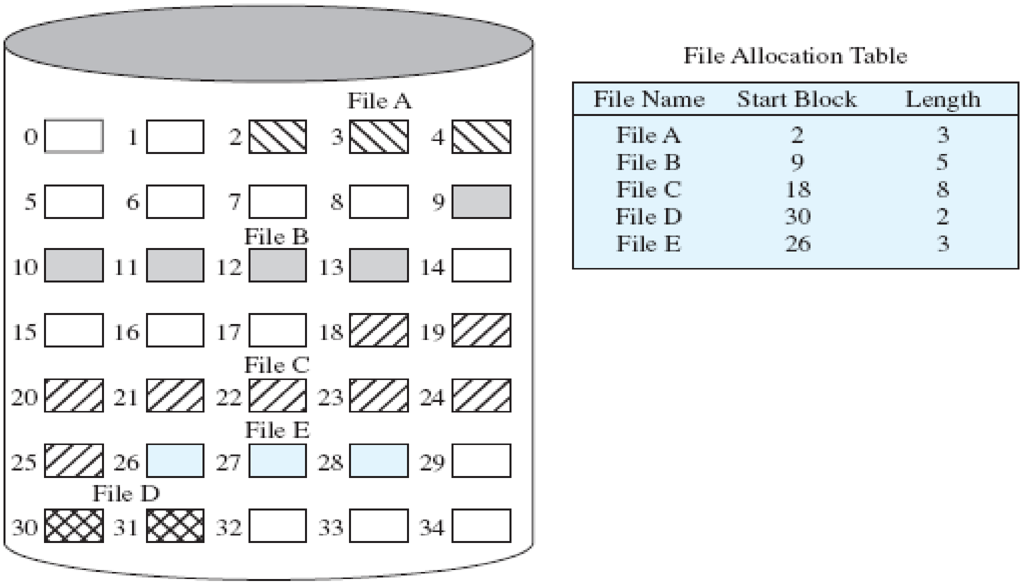
Four different techniques of free space management are discussed in this unit. The methods are:

- Bit tables
- Chained free portion
- Indexing, and
- Free block list.

3.1.2 Bit Tables

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. For example, for the disk layout of Figure 11 (reproduced as Figure 17), a vector of length 35 is needed and would have the following value:

00111000011111000011111111111011000



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Fig 18: Contiguous File Allocation

***Source**: Operating Systems; Internal and Design Principles, 5th ed. By William Stallings (2004).*

A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods outlined.

Another advantage is that it is as small as possible. However, it can still be sizeable. The amount of memory (in bytes) required for a block bitmap is

$$\frac{\text{disk size in bytes}}{8 \text{ X file system block size}}$$

Thus, for a 16-Gbyte disk with 512-byte blocks, the bit table occupies about 4 Mbytes. Can we spare 4 Mbytes of main memory for the bit table? If so, then the bit table can be searched without the need for disk access. But even with today's memory sizes, 4 Mbytes is a hefty chunk of main memory to devote to a single function. The alternative is to put the bit table on disk. But a 4-Mbyte bit table would require about 8000 disk blocks. We can't afford to search that amount of disk space every time a block is needed.

Even when the bit table is in main memory, an exhaustive search of the table can slow file system performance to an unacceptable degree. This is especially true when the disk is nearly full and there are few free blocks remaining. Accordingly, most file systems that use bit tables maintain auxiliary data structures that summarise the contents of sub-ranges of the bit table. For example, the table could be divided logically into a number of equal-size sub-ranges. A summary table could include, for each sub-range, the number of free blocks and the maximum-sized contiguous number of free blocks. When the file system needs a number of contiguous blocks, it can scan the summary table to find an appropriate sub-range and then search that sub-range.

3.1.3 Linked List/Chained Free Portions

The free portions may be chained together by using a pointer and length value in each free portion. This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first portion. This method is suited to all of the file allocation methods. If allocation is a block at a time, simply choose the free block at the head of the chain and adjust the first pointer or length value. If allocation is by variable-length portion, a first-fit algorithm may be used: The headers from the portions are fetched one at a time to determine the next suitable free portion in the chain. Again, pointer and length values are adjusted.

This method has its own problems. After some use, the disk will become quite fragmented and many portions will be a single block long. Also, note that every time you allocate a block, you need to read the block first to recover the pointer to the new first free block before writing data to that block. If many individual blocks need to be allocated, at one time, for a file operation, this greatly slows file creation. Similarly, deleting highly fragmented files is very time consuming.

3.1.4 Indexing

The indexing approach treats free space as a file and uses an index table as described under file allocation. For efficiency, the index should be on the basis of variable-size portions rather than blocks. Thus, there is one entry in the table for every free portion on the disk. This approach provides efficient support for all of the file allocation methods.

3.1.5 Free Block List

In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved portion of the disk. Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number, so the size of the free block list is 24 or 32 times the size of the corresponding bit table and thus must be stored on disk rather than in main memory. However, this is a satisfactory method. Consider the following points:

- The space on disk devoted to the free block list is less than 1% of the total disk space. If a 32-bit block number is used, then the space penalty is 4 bytes for every 512-byte block.
- Although the free block list is too large to store in main memory, there are two effective techniques for storing a small part of the list in main memory.

a. The list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory. When a new block is allocated, it is popped from the top of the stack, which is in main memory. Similarly, when a block is deallocated, it is pushed onto the stack. There has to be a transfer between disk and main memory when the in-memory portion of the stack becomes either full or empty. Thus, this technique gives almost zero-time access most of the time.
b. The list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory. A block is allocated by taking the first entry from the head of the queue and deallocated by adding it to the end of the tail of the queue. There only has to be a transfer between disk and

main memory when either the in-memory portion of the head of the queue becomes empty or the in-memory portion of the tail of the queue becomes full.

In either of the strategies listed in the preceding point (stack or FIFO queue), a background thread can slowly sort the in-memory list or lists to facilitate contiguous allocation.

SELF-ASSESSMENT EXERCISE 1

1. Why is managing space so important?
2. List the different methods operating systems can employ in space management.

3.2 Volume

The term *volume* is used somewhat differently by different operating systems and file management systems, but in essence a volume is a logical disk. Volume is defined as follows:

- "a collection of addressable sectors in secondary memory that an OS or application can use for data storage. The sectors in a volume need not be consecutive on a physical storage device; instead they need only appear that way to the OS or application. A volume may be the result of assembling and merging smaller volumes."

In the simplest case, a single disk equals one volume. Frequently, a disk is divided into partitions, with each partition functioning as a separate volume. It is also common to treat multiple disks as a single volume of partitions on multiple disks as a single volume.

3.3 Disk Quotas

To prevent people from hogging too much disk space, multiuser operating systems, such as UNIX, often provide a mechanism for enforcing disk quotas. The idea is that the system administrator assigns each user a maximum allotment of files and blocks, and the operating system makes sure that the users do not exceed their quotas. A typical mechanism is described below.

When a user opens a file, the attributes and disk addresses are located and put into an open file table in main memory. Among the attributes is an entry telling who the owner is. Any increases in file's size will be charged to the owner's quota.

A second table contains the quota record for every user with a currently open file, even if the file was open by someone else. This table is shown in the figure 19. It is an extract from a quota file on disk for the users whose files are currently open. When all the files are closed, the records written back to the quota file.

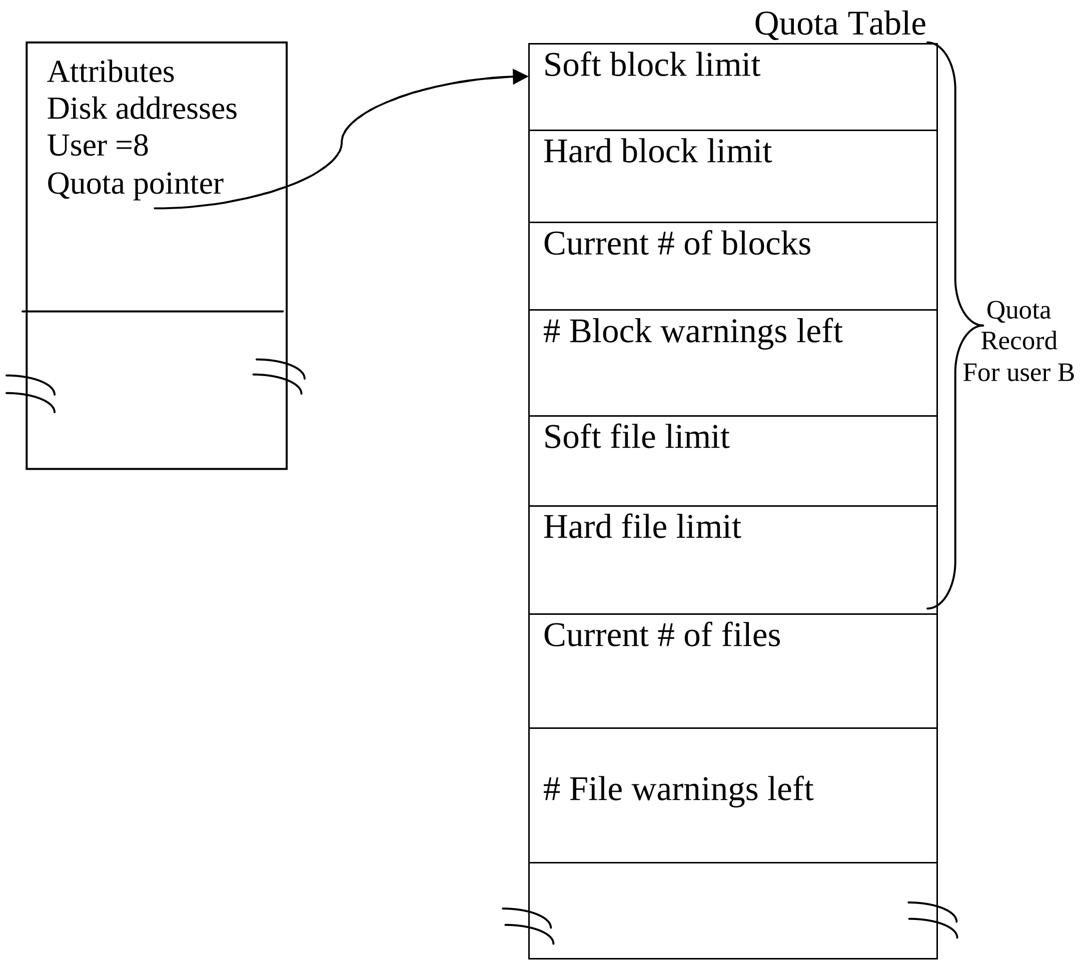


Fig 19: **Quotas are kept track of on a per-user basis in a quota table**

Source: *Modern Operating Systems by Andrew S. Tanenbaum (2006)*

When a new entry is made in the open file table, a pointer to the owner's quota record is entered into it, to make easy to find the various limits. Every time a block is added to the file, the total number of blocks charged to the owner is incremented, and a check is made against both the hard and soft limits. The soft limit may be exceeded, but the hard limit may not. An attempt to append to a file when the hard block limit has been reached will result in an error. Analogous checks also exist for the number of files.

When a user attempts to log in, the system checks the quota file to see if the user has exceeded the soft limit for either number of files or number of disk blocks. If either limit has been violated, a warning is displayed, and the count of warnings remaining is reduced by one. If the count ever gets to zero, the user has ignored the warning one time too many, and is not permitted to log in. Getting permission to log in again will require some discussion with the system administrator.

This method has the property that the users may go above their soft limits during a terminal session, provided they remove the excess before logging out. The hard limits may never be exceeded.

SELF-ASSESSMENT EXERCISE 2

What do you understand by the following:

a. Disk quotas
b. Volume
c. Bit tables

4.0 CONCLUSION

Disk space management is a very important service which any operating system must provide so as to efficiently manage disk memory and optimize service.

7.0 SUMMARY

In this unit, you have learnt that:

- Space management is an essential feature provided by operating system
- Techniques of free space management are bit tables, linked/chained free portion, indexing, and free block list.
- In bit table, each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use
- In indexing, free space is treated as a file and uses an index table as described under file allocation
- In free block list, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved portion of the disk.
- In linked/chained free portion, free portions are chained together by using a pointer and length value in each free portion

- Multiuser operating systems prevent people from hogging too much disk space by providing a mechanism for enforcing disk quotas.
- When a disk is divided into partitions, each partition functions as a separate volume.

8.0 TUTOR-MARKED ASSIGNMENT

1. Write briefly on the following in the context of free space management.

a. Bit table
b. Indexing
c. Free block list

2. Explain how a multiuser operating system ensures that user do not use too much unnecessary space.
3. What do you understand by volume with respect to storage management?

7.0 REFERENCES/FURTHER READING

Claybrook, Billy G. (1991). *File Management Techniques.* New York: John Wiley.

Davis, W.S. & Rajkumar, T.M. (2001). *Operating Systems; A Systematic View,* (5th Edition). New Jersey: Addison-Wesley.

Grosshans, D. (1986). *File Systems; Design and Implementation*. New Jersey: Prentice Hall.

Livadas, P. (1990). File Structures: Theory and Practice. New Jersey: Prentice Hall.

Silberschatz, Abraham. (2004). *Operating System Concepts with Java,* (6th Edition). New Jersey: John Wiley.

Stallings, Williams. (2004). *Operating Systems; Internal and Design Principles, (*5th edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System,* (3rd Edition). New Jersey: Prentice Hall.

filesystems.palconit.com/filesystems-free-space-management.html

www.cs.man.ac.uk/~rizos/CS2051/2001-02/lect15.pdf

UNIT 4 FILE SYSTEM PERFORMANCE AND RELIABILITY

CONTENTS

1.0 INTRODUCTION

It is very essential that a file system is reliable, long-lasting and perform optimally. The loss of valuable data due to a file system failure could lead to catastrophic consequences. In this unit, we will look at some of the issues involved in safeguarding the data and improve system performance.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain why a system may not perform optimally
- discuss the methods employed by file system designer in improving system performance
- explain why a system has to be reliable
- discuss various techniques through which a system can be made more reliable.

3.0 MAIN CONTENT

3.1 File System Performance

Access to disk is much slower than access to memory. Reading a memory word typically takes a few hundred nanoseconds at most. Reading a disk block takes tens of milliseconds, a factor of 100,000 slower. As a result of this difference in access time, many file systems have been designed to reduce the number of disk accesses needed.

3.1.1 Methods of Improving Performance

a. Block Caching

The most common technique used to reduce disk accesses is the **block cache** or **buffer cache**. (Cache is pronounced “cash,” and is derived from the French *cacher,* meaning to hide.) In this context, a cache is a collection of blocks that logically belong on the disk, but are being kept in memory for performance reasons.

Various algorithms can be used to manage the cache, but a common one is to check all read requests to see if the needed block is in the cache. If it is, the read request can be satisfied without a disk access. If the disk is not in the cache, it is first read into the cache, and then copied to wherever it is needed. Subsequent requests for the same block can be satisfied from the cache.

b. Reduction in Disk Motion

Caching is not the only way to increase the performance of a file system. Another important technique is to reduce the amount of disk arm motion by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder. When output file is written, the file system has to allocate the blocks one at a time, as they are needed. If the free blocks are recorded in a bit map, and the whole bit map is in the main memory, it is easy enough to choose a free block as close as possible to the previous block. With a free list, part of which is on disk, it is much harder to allocate blocks close together. However, even with a free list, some block clustering can be done. The trick is to keep track of disk storage not in blocks, but in groups of consecutive blocks. If a track consists of 64 sectors of 512 bytes, the system could use 1K blocks (2 sectors), but allocate disk storage in units of 2 blocks (4 sectors). This is not the same as having a 2K disk block, since the cache would still use 1K but reading a file sequentially on an otherwise idle system would reduce the number of seeks by a factor of two, considerably improving performance.

A variation on the same scheme is to take account of rotational positioning. When allocating blocks, the system attempts to place consecutive blocks in a file in the same cylinder, but interleaved for maximum throughput. Thus, if a disk has a rotation time of 16.67 msec and it takes about 4 msec for a user process to request and get a disk block, each block should be placed at least a quarter of the way around from its predecessor.

Another performance bottleneck in systems that use i-node or the equivalent is that reading even a short file requires two disk accesses: one for the i-node and one for the block. The usual i-node placement is shown in Figure 20(a) below.

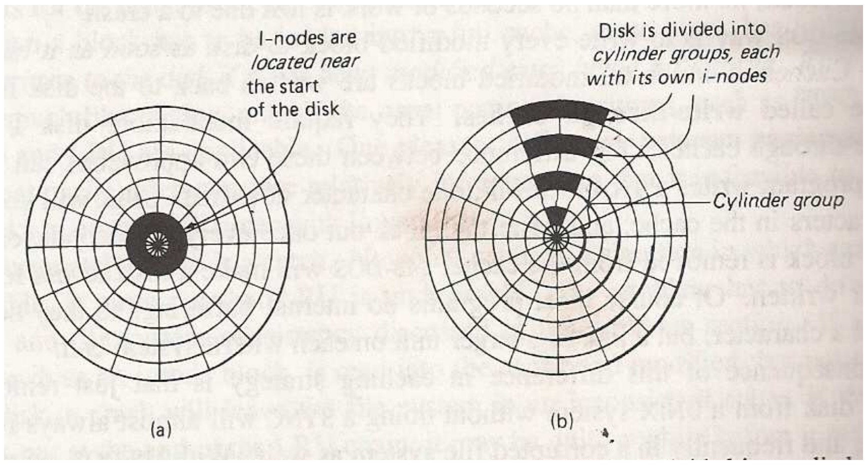

I-nodes are located near the start of the disk
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(a)
(b)

Fig 20: (a) I-nodes placed at the start of the disk
(b) Disk divided into cylinder groups, each with its own blocks and i-nodes

Source: *Modern Operating Systems by Andrew S. Tanenbaum (2006)*

Here all the i-nodes are near the beginning of the disk, so the average distance between i-node and its blocks will be about half the number of cylinders, requiring long seeks.

One easy performance improvement is to put the i-nodes in the middle of the disk, rather than at the start, thus reducing the average seek between the i-node and the first block by a factor of two. Another idea, shown in Figure 20(b) is to divide the disk into cylinder groups, each with its own i-nodes, blocks, and free list. When creating a new file, any i-node can be chosen, but having done this, an attempt is made to find a block in the same cylinder group as the i-node. If none is available, then a block in a cylinder group close by is used.

3.2 File System Reliability

Destruction of a file system is often a far greater disaster than destruction of a computer. If computer is destroyed by fire, lightning surges, or a cup of coffee poured onto the keyboard, it is annoying and will cost money, but generally a replacement can be purchased with a minimum of fuss. Inexpensive personal computers can even be replaced within a few hours.

If a computer file system is irrevocably lost, whether due to hardware, software or any other means, restoring all the information will be difficult, time consuming, and in many cases, impossible. For the people whose programs, documents, customer files, tax records, databases, marketing plans, or other data are gone forever, the consequences can be catastrophic. While the file system cannot offer any protection against any physical destruction of the equipment and media, it can help protect the information. Let's look at some of the issues involved in safeguarding the file system.

SELF-ASSESSMENT EXERCISE 1

1. What is system performance?
2. List different techniques that can be used in improving system performance

3.2.1 Techniques to Improve Reliability

a. Bad Block Management

Disks often have bad blocks. Floppy disks are generally perfect when they leave the factory, but they can develop bad blocks during use. Hard disks frequently have bad blocks from the start: it is just too expensive to manufacture them completely free of all defects. In fact, most hard disk manufacturers supply with each drive a list of the bad blocks their tests have discovered.

Two solutions to the bad block problems are used, one hardware and the other software.

- The **hardware solution** is to dedicate a sector on the disk to the bad block list. When the controller is first initialized, it reads the bad block list and picks a space block (or track) to replace the defective ones, recording the mapping in the bad block list. Henceforth, all requests for the bad block will use the spare.

- **Software solution** requires the user or file system to carefully construct a file containing all the bad blocks. This technique removes them from the free list, so they will never occur in data files. As long as the bad block file is never read or written, no problem will arise. Care has to be taken during disk backups to avoid reading this file.

b. Backups

Even with a clever strategy for dealing with bad blocks, it is important to back up files frequently. After all, automatically switching to a spare track after a crucial data block has been ruined is somewhat akin to locking the barn door after the prized race horse has escaped.

Backup technique is as simple as it sounds. It involves keeping another copy of the data on some other machine or device so that the copy could be used in case of a system failure. There are two types of backup techniques, namely full dump and incremental dump.

- **Full dump** simply refers to making a backup copy of the whole disk on another disk or machine. It is pretty obvious that the process of full dumping is time consuming as well as memory consuming.

- **Incremental dump** has some advantages over full dump. The simplest form of incremental dumping is to make a full dump periodically (say monthly or weekly) and to make a daily dump of only those files that have been modified since the last full dump. A better scheme could be to change only those files that have been changed since the last full dump. Such a scheme of data backup is time efficient as well as memory efficient. To implement this method, a list of dump times for each file must be kept on disk. The dump program should then check for each file on the hard disk and if it finds that the file has been modified since it was last dumped, it dumps the file yet again and changes the file’s time of last dump to the current time.

MS-DOS provides some assistance in making backups. Associated with each file is an attribute bit called the a**rchive bit**. When the file system is backed up, the archive bits of all the files are cleared. Subsequently, whenever a file is modified, the operating system automatically sets its archive bit. When it is time for the next backup, the backup program checks all the archive bits and only backs up those files whose bit is set. It also clears all these bits to monitor further usage of the files.

3.2.2 File System Consistency

Another way of safeguarding data is to check for file consistency. The directory information in the main memory is more updated than that residing on the disk. In case of a system malfunction, if the table of open files is lost, all the changes made in the directories of open files are lost too. Hence, there arises an inconsistency between the state of some files and the directory structure. Many systems employ a "consistency checker" program that checks and corrects disk inconsistencies. Therefore, a file management system should have some consistency checking mechanism associated with it.

SELF-ASSESSMENT EXERCISE 2

1. What are the types of backups available?
2. What is an archive bit?

4.0 CONCLUSION

File system designers have various measures in place in operating system to make sure that a system performs optimally and that it is reliable as much as technically possible.

5.0 SUMMARY

In this unit, you have learnt that:

- The loss of valuable data due to a file system failure could lead to catastrophic consequences
- System performance could be improved by block caching and reduction in disk motion
- Systems can be made more reliable by efficient bad block management, timely backups and checking for file system consistency

6.0 TUTOR-MARKED ASSIGNMENT

1. In your own words define the following terms:

a. System reliability
b. System performance
c. Disk bad block

2. Discuss the various backup techniques.
3. What are the measures available in improving system performance?

7.0 REFERENCES/FURTHER READING

Claybrook, B. G. (1991). *File Management Techniques.* New York: John Wiley.

Davis, W.S. & Rajkumar, T.M. (2001). *Operating Systems: A Systematic View,* (5th Edition). New Jersey: Addison-Wesley.

Grosshans, D. (1986). *File Systems; Design and Implementation*. New Jersey: Prentice Hall.

Livadas, P. (1990). File Structures: Theory and Practice. New Jersey: Prentice Hall.

Silberschatz, Abraham. (2004). *Operating System Concepts with Java,* (6th Edition). New Jersey: John Wiley.

Stallings, Williams (2004). *Operating Systems; Internal and Design Principles,* (5th edition). New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System,* (3rd Edition). New Jersey: Prentice Hall.

UNIT 5 FILE SYSTEM SECURITY AND INTEGRITY

CONTENTS

1.0 INTRODUCTION

File systems often contain information that is highly valuable to their users. Therefore, prevention of or protection against (a) access to information or (b) intentional but unauthorised destruction or alteration of that information is a major concern. In this unit we will look at a variety of issues concerned with security and integrity.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- state and explain two reasons why computing environment needs to be secured
- identify different categories of intrusion and mechanism to detect them
- list and discuss some design principles for security

- explain different methods that can be used to authenticate a user such as: password, biometrics
- list and explain different types of threats in computing environment.

3.0 MAIN CONTENT

3.1 The Security Environment

Security has many facets. Two of the more important ones relate to *data loss* and *intrusion*. Some of the common causes of data loss are:

- Natural phenomenon such as: fire, flood, earthquakes, wars, riots or attacks from rodents
- Hardware or software errors: CPU malfunctions, unreadable disks or tapes, telecommunication errors, program bugs
- Human errors: incorrect data entry, wrong tape or disk mounted, wrong program run, lost disk or tape.

Most of these can be dealt with by maintaining adequate backups, preferably far away from the original data.

3.1.1 System Protection

To protect the system, we must take security measures at four levels:

- **Physical:** The site or sites containing the computer systems must be physically secured against armed or surreptitious entry by intruders.
- **Human:** Users must be screened carefully to reduce the chances of authorising a user who then gives access to an intruder (in exchange for a bribe, for example).
- **Network:** Much computer data in modern systems travels over private leased lines, shared lines such as: the Internet, or dial-up lines. Intercepting these data could be just as harmful as breaking into a computer; and interruption of communications could constitute a remote **denial-of-service attack**, diminishing users' use of and trust in the system.
- **Operating system:** The system must protect itself from accidental or purposeful security breaches.

Security at the first two levels must be maintained if operating-system security is to be ensured. A weakness at a high level of security (physical or human) allows circumvention of strict low-level (operating-system) security measures.

Furthermore, the system hardware must provide protection to allow the implementation of security features. Most contemporary operating systems are now designed to provide security features.

3.2 Intrusion/Categories of Intruders

Intrusion is a set of actions that attempt to compromise the integrity, confidentiality, or availability of any resource on a computing platform.

Categories of Intruders

- **Casual prying by non technical users**. Many people have terminals to timesharing systems on their desks, and human nature being what it is, some of them will read other people's electronic mails and other files if no barriers are placed in the way.
- **Snooping by insiders**. Students, system programmers, operators, and other technical personnel often consider it to be a personal challenge to break the security of a local computer system. They are often highly skilled and are willing to devote a substantial amount of time to the effort.
- **Determined attempt to make money.** Some bank programmers have attempted banking system to steal from the bank. Schemes vary from changing software to truncating rather than rounding off interest, keeping the fraction of money for themselves, siphoning off accounts not used for years, to blackmail ("pay me or I will destroy all the bank's records.")
- **Commercial or military espionage.** Espionage refers to a serious and well funded by a competitor or a foreign country to steal programs, trade secrets, patents, technology, circuit designs, marketing plans, and so forth. Often this attempt will involve wiretapping or even erecting antennas at the computer to pick up its electromagnetic radiation.

The amount of effort that one puts into security and protection clearly depends on who the enemy is thought to be. Absolute protection of the system from malicious abuse is not possible, but the cost to the perpetrator can be made sufficiently high to deter most, if not all, unauthorised attempts to access the information residing in the system.

3.2.1 Intrusion Detection

Intrusion detection strives to detect attempted or successful intrusions into computer systems and to initiate appropriate responses to the intrusions. Intrusion detection encompasses a wide array of techniques that vary on a number of *axes*. These axes include:

- The time that detection occurs. Detection can occur in real time (while the intrusion is occurring) or after.
- The types of inputs examined to detect intrusive activity.
- The range of response capabilities. Simple forms of response include alerting an administrator to the potential intrusion or somehow halting the potentially intrusive activity—for example, killing a process engaged in apparently intrusive activity.

Intrusion can be detected through:

- **Auditing and Logging.** A common method of intrusion detection is **audit-trail processing**, in which security-relevant events are logged to an audit trail and then matched against attack signatures (in signature-based detection) or analyzed for anomalous behavior (in anomaly detection).
- **Tripwire** operates on the premise that many intrusions result in anomalous modification of system directories and files.
- **System-Call Monitoring** is a more recent and speculative form of anomaly detection. This approach monitors process system calls to detect in real time when a process is deviating from its expected system-call behavior.

3.3 Design Principles for Security

Some researchers have identified several general principles that can be used as guide to designing secure systems. A brief summary of their ideas is given below:

- System design should be public.
- The default should be no access. Errors in which legitimate access is refused will be reported much faster than errors in which unauthorised access is allowed.
- Check for current authority. The system should not check permission, determine that access is permitted. Many systems check for permission when a file is opened and not afterwards. This means that a user who opens a file and keeps it open for weeks will continue to have access, even if the owner has long changed the file protection.
- Give each process the least privilege possible.
- The protection mechanism should be simple, uniform, and build to the lowest layers of the system.
- The scheme chosen must be psychologically acceptable. If users feel that protecting their files is too much work, they just will not do it.

3.4 User Authentication

A major security problem for operating systems is authentication. The protection system depends on the ability to identify the programs and processes currently executing, which in turn depends on the ability to identify each user of the system. The process of identifying users when they log on is called user authentication. How do we determine whether a user's identity is authentic? Generally, authentication is based on one or more of three items:

- User possession (a key or card)
- User knowledge (a user identifier and password)
- User attributes (fingerprint, retina pattern, or signature).

SELF-ASSESSMENT EXERCISE 1

1. What is intrusion and how can it be detected?
2. State some of the design principles for securing a computing environment
3. What is user authentication and how can it be implemented?

3.4.1 Passwords

The most common approach to authenticating a user identity is the use of **passwords**. When a user identifies herself by user ID or account name, she is asked for a password. If the user-supplied password matches the password stored in the system, the system assumes that the user is legitimate. Passwords are often used to protect objects in the computer system, in the absence of more complete protection schemes. Different passwords may be associated with different access rights. For example, different passwords may be used for reading files, appending files, and updating files.

Password Vulnerabilities

Passwords are extremely common because they are easy to understand and use. Unfortunately, passwords can often be guessed, accidentally exposed, sniffed, or illegally transferred from an authorized user to an unauthorised one.

There are two common ways to guess a password.

- One way is for the intruder (either human or program) to know the user or to have information about the user.

- The use of brute force, trying enumeration, or all possible combinations of letters, numbers, and punctuation, until the password is found.

Short passwords are especially vulnerable to the last method. For example, a four-digit password provides only 10,000 variations. On average, guessing 5,000 times would produce a correct hit. A program that could try a password every 1 millisecond would take only about 5 seconds to guess a four-digit password. Enumeration is not as successful at finding passwords in systems that allow longer passwords, that differentiate between uppercase and lowercase letters, and that allow use of numbers and all punctuation characters in passwords.

In addition to being guessed, passwords can be exposed as a result of visual or electronic monitoring. An intruder can look over the shoulder of a user (**shoulder surfing**) when the user is logging in and can learn the password easily by watching the keystrokes. Alternatively, anyone with access to the network on which a computer resides could seamlessly add a network monitor, allowing her to watch all data being transferred on the network (**sniffing**), including user IDs and passwords. Encrypting the data stream containing the password solves this problem. Exposure is a particularly severe problem if the password is written down where it can be read or lost. Some systems force users to select hard-to-remember or long passwords, which may cause a user to record the password. As a result, such systems provide much less security than systems that allow easy passwords!

The final method of password compromise/illegal transfer is the result of human nature. Most computer installations have a rule that forbids users to share accounts. This rule is sometimes implemented for accounting reasons but is often aimed at improving security.

3.4.2 Biometrics

There are many other variations to the use of passwords for authentication. Palmor hand-readers are commonly used to secure physical access—for example, access to a data center. These readers match stored parameters against what is being read from hand-reader pads. The parameters can include a temperature map, as well as finger length, finger width, and line patterns. These devices are currently too large and expensive to be used for normal computer authentication.

Fingerprint readers have become accurate and cost-effective and should become more common in the future. These devices read your finger's ridge patterns and convert them into a sequence of numbers. Over time, they can store a set of sequences to adjust for the location of the finger on the reading pad and other factors. Software can then scan a finger on the pad and compare its features with these stored sequences to determine if the finger on the pad is the same as the stored one. Of course, multiple users can have profiles stored, and the scanner can differentiate among them. A very accurate two-factor authentication scheme can result from requiring a password as well as a user name and fingerprint scan. If this information is encrypted in transit, the system can be very resistant to spoofing or replay attack.

3.5 Programme Threats

When a program written by one user may be used by another, misuse and unexpected behavior may result. Some common methods by which users gain access to the programs of others are:

- Trojan horses
- Trap doors
- Stack and buffer overflow.

3.5.1 Trojan Horse

Many systems have mechanisms for allowing programs written by users to be executed by other users. If these programs are executed in a domain that provides the access rights of the executing user, the other users may misuse these rights. A text-editor program, for example, may include code to search the file to be edited for certain keywords. If any are found, the entire file may be copied to a special area accessible to the creator of the text editor. A code segment that misuses its environment is called a **Trojan horse**.

A variation of the Trojan horse is a program that emulates a login program. An unsuspecting user starts to log in at a terminal and notices that he has apparently mistyped his password. He tries again and is successful. What has happened is that his authentication key and password have been stolen by the login emulator, which was left running on the terminal by the thief. The emulator stored away the password, print out a login error message, and exit; the user was then provided with a genuine login prompt.

3.5.2 Trap Door

The designer of a program or system might leave a hole in the software that only he/she is capable of using. This type of security breach is called **trap door**. Programmers have been arrested for embezzling from banks by including rounding errors in their code and having the occasional half-cent credited to their accounts. This account crediting can add up to a large amount of money, considering the number of transactions that a large bank executes. Trap doors pose a difficult problem because, to detect them, we have to analyze all the source code for all components of a system. Given that software systems may consist of millions of lines of code, this analysis is not done frequently, and frequently it is not done at all!

3.5.3 Stack and Buffer Overflow

This is the most common way for an attacker outside of the system, on a network or dial-up connection, to gain unauthorized access to the target system. An authorized user of the system may also use this exploit for **privilege escalation**, to gain privileges beyond those allowed for that user. Essentially, the attack exploits a bug in a program. The bug can be a simple case of poor programming, in which the programmer neglected to code bounds checking on an input field. The buffer-overflow attack is especially pernicious, as it can be run within a system and can travel over allowed communications channels. Such attacks can occur within protocols that are expected to be used to communicate with the machine, and they can therefore be hard to detect and prevent. They can even bypass the security added by firewalls. One solution to this problem is for the CPU to have a feature that disallows execution of code in a stack section of memory.

3.6 System Threats

Most operating systems provide a means by which processes can spawn other processes. In such an environment, it is possible to create a situation where operating system resources and user files are misused. The two most common methods for achieving this misuse are worms and viruses. We discuss each below, along with a somewhat different form of system threat: *denial of service*.

3.6.1 Worms

A **worm** is a process that uses the *spawn* mechanism to ravage system performance. The worm spawns copies of itself, using up system resources and perhaps locking out all other processes. On computer networks, worms are particularly potent, since they may reproduce themselves among systems and thus shut down the entire network.

3.6.2 Viruses

Like worms, viruses are designed to spread into other programmes and can wreck havoc in a system by modifying or destroying files and causing system crashes and programme malfunctions. Whereas a worm is structured as a complete, standalone programme, a virus is a fragment of code embedded in a legitimate programme. Viruses are a major problem for computer users, especially users of microcomputer systems. Multiuser computers generally are not susceptible to viruses because the executable programs are protected from writing by the operating system. Even if a virus does infect a programme, its powers are limited because other aspects of the system are protected. Single-user systems have no such protections and, as a result, a virus has free run. Viruses are usually spread when users download viral programmes from public bulletin boards or exchange disks containing an infection.

In recent years, a common form of virus transmission has been via the exchange of Microsoft Office files, such as Microsoft Word documents. These documents can contain so-called *macros* (or Visual Basic Programmes) that programmes in the Office suite (Word, PowerPoint, or Excel) will execute automatically.

Most commercial antivirus packages are effective against only particular known viruses. They work by searching all the programmes on a system for the specific pattern of instructions known to make up the virus. When they find a known pattern, they remove the instructions, *disinfecting* the programme. These commercial packages have catalogs of thousands of viruses for which they search. Viruses and the antivirus software continue to become more sophisticated. Some viruses modify themselves as they infect other software to avoid the basic pattern-match approach of antivirus software. The antivirus software in turn now looks for families of patterns rather than a single pattern to identify a virus.

The best protection against computer viruses is prevention, or the practice of **safe computing**. Purchasing unopened software from vendors and avoiding free or pirated copies from public sources or disk exchange is the safest route to preventing infection. For macro viruses, one defense is to exchange Word documents in an alternative file format called **rich text format (RTF)**. Unlike the native Word format, RTF does not include the capability to attach macros. Another defense is to avoid opening any e-mail attachments from unknown users.

3.6.3 Denial of Service

The last attack category, **denial of service**, is aimed not at gaining information or stealing resources but rather at disrupting legitimate use of a system or facility. An intruder could delete all the files on a system, for example. Most denial-of-service attacks involve systems that the attacker has not penetrated. Indeed, launching an attack that prevents legitimate use is frequently easier than breaking into a machine or facility.

3.7 File Protection

There are three most popular implementations of file protection:

- **File Naming**

It depends upon the inability of a user to access a file he cannot name. This can be implemented by allowing only users to see the files they have created. But since most file systems allow only a limited number of characters for filenames, there is no guarantee that two users will not use the same filenames.

- **Password Protection**

This scheme associates a password to each file. If a user does not know the password associated to a file then he cannot access it. This is a very effective way of protecting files but for a user who owns many files, and constantly changes the password to make sure that nobody accesses these files will require that users have some systematic way of keeping track of their passwords.

- **Access Control**

An access list is associated with each file or directory. The access list contains information on the type of users and accesses that they can do on a directory or file. An example is the following access list associated to a UNIX file or directory:

drwxrwxrwx

The **d** indicates that this is an access list for a directory, the first **rwx** indicates that it can be read, written, and executed by the owner of the file, the second **rwx** is an access information for users belonging to the same group as the owner (somewhere on the system is a list of users belonging to same group as the owner), and the last **rwx** for all other users. The **rwx** can be changed to just **r - -** indicating that it can only be read, or – w - for write-only, **- - x** for execute only.

SELF-ASSESSMENT EXERCISE 2

1. Write short notes on the following:

a. Viruses
b. Worms
c. Denial of service

d. Trojan horse
e. Trap door

2. What are the different ways through which a file could be protected?

4.0 CONCLUSION

The data stored in the computer system must be protected from unauthorised access, malicious destruction or alteration, and accidental introduction of inconsistency. It is easier to protect against accidental loss of data consistency than to protect against malicious access to the data. Absolute protection of the information stored in a computer system from malicious abuse is not possible; but the cost to the perpetrator can be made sufficiently high to deter most, if not all, attempts to access that information without proper authority.

Methods of preventing or detecting security incidents include intrusion-detection systems, auditing and logging of system events, monitoring of system software changes, and system-call monitoring.

5.0 SUMMARY

In this unit, you have learnt that:

- Data loss could be caused by natural tragedy (such as earthquake, flood), hardware or software failure and human errors
- Users can be authenticated by the use of password, biometrics, etc
- Computer files can be attacked through System threats (Trojan horse, trap doors, stack and overflow buffer) and program threats (Worms and Virus)
- Files can be protected through password, file naming and access control
- Passwords are vulnerable to attack and may give the best type of file/ system protection.

6.0 TUTOR-MARKED ASSIGNMENT

1. Compare and contrast viruses and worms.
2. Write all that you know about intrusion.
3. What do you understand by System and Program threats?
4. What do you understand by “Safe Computing”?

7.0 REFERENCES/FURTHER READING

Claybrook, B. G. (1991). *File Management Techniques*. New York: John Wiley.

Davis, W.S. & Rajkumar, T.M. (2001). *Operating System: A Systematic View,* (5th Edition). New Jersey: Addison-Wesley.

Deitel, H. M. (2004). *Operating Systems, (3rd Edition)*. New Jersey: Prentice Hall.

Stallings, Williams. (2004). *Operating Systems; Internal and Design Principles,* (5th Edition*)*. New Jersey: Prentice Hall.

Tanenbaum, A. S. (2006). *Modern Operating System,* (3rd Edition). New Jersey: Prentice Hall.

MODULE 3 FILE PROCESSING AND APPLICATIONS

UNIT 1 DATA VALIDATION

CONTENTS

1.0 INTRODUCTION

It is very essential that data being input to a data processing task are accurate and meet all conditions for its usage. Data could be in form of numbers, text, images, or sounds. A computer can not notice errors in the data being processed in the way that humans do. Validation checks are an attempt to build into the computer programme power of judgment so that incorrect items of data are detected and reported.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain data validation concepts
- explain different data validation procedures
- discuss validation rule and validation criteria
- define what data dictionary is
- explain data corruption.

3.0 MAIN CONTENT

3.1 Data Validation

In computing, **data validation** is the process of ensuring that a programme operates on clean, correct and appropriate data. It uses routines, often called "*validation rules*" or "*check routines*", that check for correctness, meaningfulness, and security of data that are input to the system. The rules may be implemented through the automated facilities of a data dictionary, or by the inclusion of explicit application programme validation logic.

The checks can be made at two steps:

a. Input – when data is first input to the computer, different checks can be applied to prevent errors going forward for processing.
b. Update – further checking is possible when data input are matched with master files.

3.2 Validation Methods

- **Format or picture check**

Checks that the data is in a specified format (template), e.g., dates are in the format DD/MM/YYYY.

- **Data type checks**

Checks the data type of the input and give an error message if the input data does not match with the chosen data type, e.g. In an input box accepting numeric data, if the letter 'O' was typed instead of the number zero, or letter 'I' is entered instead of figure '1', an error message would appear.

- **Range check**

Checks that the data lie within a specified range of values, e.g., the month of a person's date of birth should lie between 1 and 12.

- **Limit check**

Unlike range checks, data is checked for one limit only, upper OR lower, e.g., data should not be greater than 2 (>2).

- **Presence check**

Checks that important data are actually present and have not been missed out, e.g., customers may be required to have their telephone numbers listed.

- **Check digits**

Used for numerical data. An extra digit is added to a number which is calculated from the digits. The computer checks this calculation when data are entered, e.g., The ISBN (International Standard Book Number) for a book. The last digit is a check digit calculated using a modulus 11 method.

- **Batch totals**

Check for missing records. Numerical fields may be added together for all records in a batch. The batch total is entered and the computer checks that the total is correct, e.g., add the 'Total Cost' field of a number of transactions together.

- **Hash totals**

This is just a batch total done on one or more numeric fields which appears in every record, *e.g.*, add the Telephone Numbers together for a number of Customers.

- **Spelling and grammar check**

Look for spelling and grammatical errors.

- **Consistency Checks**

Checks fields to ensure data in these fields corresponds, e.g., If Title = “Mr.”, then Gender = "M".

- **Cross-system Consistency Checks**

Compares data in different systems to ensure it is consistent, e.g., the address for the customer with the same id is the same in both systems. The data may be represented differently in different systems and may need to be transformed to a common format to be compared, e.g., one system may store customer name in a single name field as 'Yaradua, Shehu M', while another in three different fields: First_Name (Shehu), Last_Name (Yaradua) and Middle_Name (Musa); to compare the two, the validation engine would have to transform data from the second system to match the data from the first, for example, using SQL: Last_Name || ', ' || First_Name || substr(Middle_Name, 1, 1) would convert the data from the second system to look like the data from the first 'Yaradua, Shehu M'.

- **File existence check**

Checks that a file with a specified name exists. This check is essential for programs that use file handling.

- **Logic check**

Checks that an input does not yield a logic error, e.g., an input value should not be 0 when there will be a number that divides it somewhere in a programme.

3.3 Validation Rule

A validation rule is a criterion used in the process of data validation, carried out after the data has been encoded onto an input medium and involves a data vet or validation programme. This is distinct from *formal verification*, where the operation of a programme is determined to be that which was intended, and that meets the purpose.

The method is to check that data falls into the appropriate parameters defined by the systems analyst. A judgment as to whether data is valid is made possible by the validation program, but it cannot ensure complete accuracy. This can only be achieved through the use of all the clerical and computer controls built into the system at the design stage.

The difference between data validity and accuracy can be illustrated with a trivial example. A company has established a personnel file and each record contains a field for the Job Grade. The permitted values are A, B, C, or D. An entry in a record may be valid and accepted by the system if it is one of these characters, but it may not be the correct grade for the individual worker concerned. Whether a grade is correct can only be established by clerical checks or by reference to other files. During systems design, therefore, data definitions are established which place limits on what constitutes valid data. Using these data definitions, a range of software validation checks can be carried out.

3.4 Validation Criteria

An example of a validation check is the procedure used to verify an ISBN.

- **Size**

The number of characters in a data item value is checked; for example, an ISBN must consist of 10 characters only (in the previous version--the standard for 1997 and later has been changed to 13 characters.)

- **Format checks**

Data must conform to a specified format. Thus, the first 9 characters must be the digits 0 through 9' the 10th must be either those digits or an *X*.

- **Consistency**

Codes in the data items which are related in some way can thus be checked for the consistency of their relationship. The first number of the ISBN designates the language of publication. for example, books published in French-speaking countries carry the digit "2". This must match the address of the publisher, as given elsewhere in the record.

- **Range**

Does not apply to ISBN, but typically data must lie within maximum and minimum preset values. For example, customer account numbers may be restricted within the values 10000 to 20000, if this is the arbitrary range of the numbers used for the system.

- **Check digit**

An extra digit calculated on, for example, an account number, can be used as a self-checking device. When the number is input to the computer, the validation program carries out a calculation similar to that used to generate the check digit originally and thus checks its validity. This kind of check will highlight transcription errors where two or more digits have been transposed or put in the wrong order. The 10th character of the 10- character ISBN is the check digit.

SELF-ASSESSMENT EXERCISE 1

1. Briefly explain what you understand by data validation?
2. State at least seven data validation methods you know and explain five of them.
3. What is the difference between data validity and data accuracy?

3.5 Data Dictionary

Data Dictionary, in computer science, is the description of the data stored in a database. The content of the data dictionary may best be thought of as “data about the data”—that is, description of all of the other objects (files, programs, and so on) in the system. In particular, a data dictionary stores all the various schemas and file specifications and their locations. A complete data dictionary also includes information about which programs use which data and which users are interested in which reports. The data dictionary is frequently integrated into the system it describes.

3.6 Data Corruption

Data corruption refers to errors in computer data that occur during transmission or retrieval, introducing unintended changes to the original data. In general, when there is a data corruption, the file containing that data would be inaccessible, and the system or the related application will give an error. For example, if a Microsoft Word file is corrupted, when you try to open that file with MS Word, you will get an error message, and the file would not be opened. Some programs can give a suggestion to repair the file automatically. It depends on the level of corruption, and the in-built functionality of the application to handle the error. There are various causes of the corruption.

Data corruption during transmission has a variety of causes. Interruption of data transmission causes *information loss*. Environmental conditions can interfere with data transmission, especially when dealing with wireless transmission methods. Heavy clouds can block satellite transmissions. Wireless networks are susceptible to interference from devices such as microwave ovens.

Data loss during storage has two broad causes: hardware and software failure. Head crashes, and general wear and tear of media fall into the former category, while software failure typically occurs due to bugs in the code.

When data corruption behaves as a poisson process, where each bit of data has an independently low probability of being changed, data corruption can generally be detected by the use of checksums, and can often be corrected by the use of error correcting codes.

If an uncorrectable data corruption is detected, procedures such as automatic retransmission or restoration from backups can be applied. Certain levels of RAID disk arrays have the ability to store and evaluate parity bits for data across a set of hard disks and can reconstruct corrupted data upon the failure of a single or multiple disks, depending on the level of RAID implemented.

If appropriate mechanisms are employed to detect and remedy data corruption, data integrity can be maintained. This is particularly important in banking, where an undetected error can drastically affect an account balance, and in the use of encrypted or compressed data, where a small error can make an extensive dataset unusable.

SELF-ASSESSMENT EXERCISE 2

1. What is data dictionary?
2. What are the various factors that can be responsible for data corruption?

4.0 CONCLUSION

Various methods by which data can be validated have been discussed. It is always essential that data to be used in data processing are accurate and falls within the allowed limit. Accuracy and appropriateness of data ensure integrity of the system and all processes that go around it.

5.0 SUMMARY

In this unit, you have learnt that:

- Data validation is the process of ensuring that a program operates on clean, correct and appropriate data.
- Various methods are available to make sure that data are accurate
- Validation rule is a criterion used in the process of data validation
- Data dictionary is a description of the data stored in a database
- Data corruption refers to errors in computer data that occur during transmission or retrieval.

6.0 TUTOR-MARKED ASSIGNMENT

1. Write exhaustively on the various methods of data validation
2. Discuss validation rules and validation criteria
3. What is the importance of data dictionary?
4. What do you understand by data corruption?

7.0 REFERENCES/FURTHER READING

www.http://en.wikipedia.org/wiki/Data_corruption

www.http://en.wikipedia.org/wiki/Validation_rule"

www.http://en.wikipedia.org/wiki/Data_validation"

Microsoft ® Encarta ® 2007. © 1993-2006 Microsoft Corporation.

UNIT 2 FILE MANAGERS

CONTENTS

1.0 INTRODUCTION

Various implementations of file management routines are available. The utility software, as they are called is either integrated with operating system or available as off-the-shelf software. This unit illustrates file management software with various examples.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- list the different types of file managers
- discuss peculiar features of each of the file managers
- identify which file is in use on any computer system.

3.0 MAIN CONTENT

3.1 File Manager

A *file manager* or *file browser* is a computer program that provides a user interface to work with file systems. The most common operations are *create, open, edit, view, print, play, rename, move, copy, delete,* *attributes*, *properties*, *search/find*, and *permissions*. Files are typically displayed in a hierarchy. Some file managers contain features inspired by web browsers, including forward and back navigational buttons. Some file managers provide network connectivity such as FTP, NFS, SMB or WebDAV. This is achieved either by allowing the user to browse for a file server, connect to it and access the server's file system like a local file system, or by providing its own full client implementations for file server protocols.

3.1.1 Orthodox File Managers

Orthodox file managers or “Commander-like” file managers have three windows (two panels and one command line window). Orthodox file managers are one of the older families of file managers. They develop and further extend the interface introduced by John Socha’s famous Norton Commander for DOS. The concept is more than twenty years old as Norton Commander version 1.0 was released in 1986. Despite their age they are actively developed and dozens of implementations exist for DOS, UNIX and Microsoft Windows.

3.1.2 Features of Orthodox File Managers

The following features define the class of orthodox file managers.

- They present the user with a two-panel directory view consisting of one active and one passive panel. The latter always serves as a target for file operations. Panels are shrinkable and if shrunk they expose the terminal window hidden behind them. Normally only the last line of the terminal window (the command line) is visible.
- They provide close integration with an underlying Operating System shell via command line and associated terminal window that permits viewing the results of executing the shell command entered on the command line (e.g., via Ctrl-O shortcut in Norton Commander).
- They provide the user with extensive keyboard shortcuts.
- The file manager can be used without or with minimal use of the mouse.
- Users can create their own file associations and scripts that are invoked for certain file types and organize these scripts into a hierarchical tree (e.g., as a user script library or user menu).
- Users can extend the functionality of the manager via the so called *User menu* or *Start menu* and extensions menu. Norton Commander introduced the concept of user-defined file associations that is now used in all modern file managers.

Other common features include:

- Information on the “active” and “passive” panels may be used for constructing commands on the command line. Examples include current file, path to left panel, path to right panel, etc.
- They provide an in-built viewer for (at least) the most basic file types.
- They have a built-in editor. In many cases, the editor can extract certain elements of the panels into the text being edited.
- Many support virtual file systems (VFS) such as viewing compressed archives, or via an FTP connection.
- They often have the word *commander* in the name.

An orthodox file manager typically has three windows. Two of the windows are called panels and are symmetrically positioned at the top of the screen. The third is the command line which is essentially a minimised command (shell) window that can be expanded to full screen. Only one of the panels is active at a given time. The active panel contains the “file cursor”. Panels are resizable. Each panel can be hidden. Files in the active panel serve as the source of file operations performed by the manager. For example, files can be copied or moved to the passive panel. This gives the user the ability to use only the keyboard with the convenience of the mouse interface. The active panel shows information about the current working directory and the files that it contains. The passive (inactive) panel shows the contents of the same or other directory (the default target for file operations). Users may customize the display of columns that show relevant file information. The active panel and passive panel can be switched (often by pressing the tab key). Other user interface elements include:

1. Path: shows the source/destination location of the directory in use
2. Information about directory size, disk usage and disk name (usually at the bottom of the panels)
3. Panel with information about file name, extension, date and time of creation, last modification, permissions (attributes) and others
4. Info panel with number of files in directory, sum of size of selected files
5. Tabbed interface (usually GUI file managers)
6. Function keys: F1 to F10 has all the same functions under all orthodox file managers. Examples: F5 always copies file(s) from the active to the inactive panel, while F6 moves the file.

The introduction of tabbed panels in some file managers (for example Total Commander) made it possible to manipulate more than one active and passive directory at a time.

Orthodox file managers are among the most portable file managers. Examples are available on almost any platform both with command-line interface and graphical user interface. This is the only type of command line managers that have a published standard of the interface (and actively supported by developers). This makes it possible to do the same work on different platforms without much relearning of the interface.

Sometimes they are called dual-pane managers, a term that is typically used for programs such as the Windows File Explorer. It is technically incorrect since they have three windows including a command line window below (or hidden behind) two symmetric panels. Command line windows play a very prominent role in the functionality of this type of file manager. Furthermore, most of these programs allow using just one pane with the second one hidden. Focusing on 'dual panes' may be misleading; it is the combination of all of these features which is important.

In summary, a chief distinguishing feature is the presence of the command line window and direct access to shell via this window - not the presence of two symmetric panes which is relatively superficial.

Notable examples include:

- Altap Salamander
- Demos Commander
- Directory Opus
- Dos Navigator
- Double Commander
- FAR Manager
- File Commander
- FreeCommander
- Krusader
- Midnight Commander
- muCommander
- Norton Commander
- PathMinder
- SE-Explorer
- Total Commander
- Volkov Commander
- WinSCP
- ZTreeWin

3.2 File-List File Manager

Less well-known, but older are the so-called *file-list* file managers. Examples include **flist** which was in use since 1981 on the Conversational Monitor System. This is a variant of **fulist** which originated before late 1978 according to comments by its author Theo Alkema.

The **flist** program provides a list of files in the user's "minidisk" and allows sorting by any of the file attributes. The file attributes could be passed to scripts or function-key definitions, making it simple to use flist as part of CMS EXEC, EXEC 2 or xedit scripts.

This program ran only on IBM VM/SP CMS, but was the inspiration for other programs, for example **filelist** (a script run via the Xedit editor), and programs running on other operating systems. These include a program also called **flist** running on OpenVMS and **fulist** (from the name of the corresponding internal IBM program) on Unix.[11]

3.3 Directory Editors

While this category is known as file managers, an older term is *directory editor*, which dates back at least to 1978. There was a directory editor written for EXEC 8 at the University of Maryland, available to other users at that time. The term was used by other developers, e.g., the **dired** program written by Jay Lepreau in 1980, which ran on BSD. This was in turn inspired by an older program with the same name running on TOPS-20. **Dired** inspired other programs, e.g., dired the editor script (for emacs and similar editors) as well as **ded**.

SELF-ASSESSMENT EXERCISE 1

1. What is a file browser?
2. What are some of the features of orthodox file managers?
3. Write briefly on file-list file manager.

3.4 Navigational File Manager

A *navigational file manager*, also called an ***Explorer type manager***, is a newer type of file manager which became prominent because of its integration in Microsoft Windows. The Windows Explorer is a classic representative of the type, using a "navigational" metaphor to represent file system locations. Since the advent of GUIs it has become the dominant type of file manager for desktop computers, being used, for example, in all Microsoft Windows products.

Typically it has two panes, with the file system tree in the left pane and the current directory in the right one. For Mac OS X, the Finder is an example of a navigational file manager.

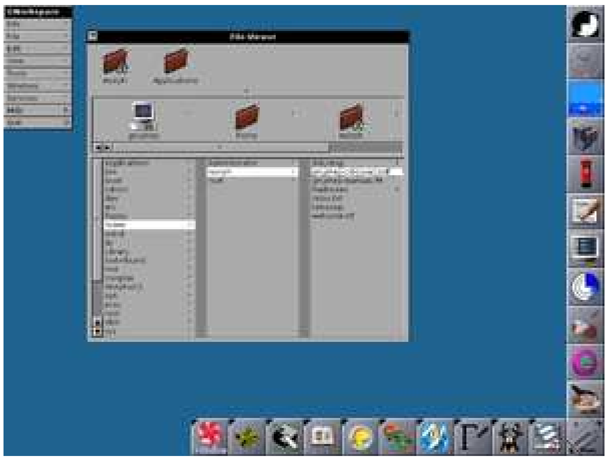


Fig 21: The Miller Column Browser from GNUstep is a type of Navigational File Manager.

Source: *www.en.wikipedia.com*

3.4.1 Features of Navigational File Manager

- The window displays the location currently being viewed.
- The location being viewed (the current directory) can be changed by the user, by opening folders, pressing a *back button*, typing a location, or using additional pane with the navigation tree representing part or all the file system.
- Icons represent files, programs, and directories.

The interface in a navigational file manager often resembles a web browser, complete with *back*, *forward* buttons that work with history, and maybe even *reload* buttons. Sometimes there is also an address bar where the file or directory path (or URL) can be typed.

Moving from one location to another need not open a new window. At the same time several file manager instances can be opened, and they can communicate with each other via drag-and-drop and clipboard operations, so it is possible to view several directories simultaneously and perform cut-and paste operations between instances.

Most navigational managers have two panes with the left pane a tree view of the file system. The latter serves as the most common instrument for file system navigation. This means that unlike orthodox managers, the two panes are asymmetrical: the first (usually left) provides the tree view of file system and the second (usually right) gives file view of the current directory.

When a directory of the tree is selected it becomes current and the content of the second (right) pane changes' to the files in the current directory. File operations are based on drag-and-drop and editor metaphors: users can select and copy files or directories into the clipboard and then paste them in a different place in the file system or even in a different instance of file manager.

Example of Navigational File Manager

Notable examples include:

- Windows Explorer
- Mac OS X Finder
- XTree / ZTreeWin
- XYplorer

3.5 Spatial File Manager

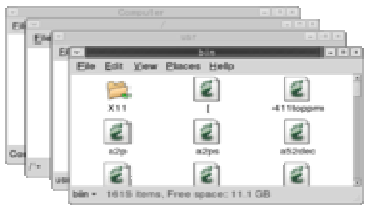


Fig 22: The Nautilus File Manager

Source: *www.en.wikipedia.com*

Spatial file managers use a spatial metaphor to represent files and folders as if they were real physical objects. A spatial file manager imitates the way people interact with physical objects.

3.5.1 Features of Spatial File Manager

Some ideas behind the concept of a spatial file manager are:

- A single window represents each opened folder.
- Each window is unambiguously and irrevocably tied to a particular folder.
- Stability: files, folders, and windows go where the user moves them, stay where the user puts them ("preserve their spatial state"), and retain all their other "physical" characteristics (such as size, shape, color and location).
- The same item can only be viewed in one window at a time.

As in navigational managers, when a folder is opened, the icon representing the folder changes—perhaps from an image showing a closed drawer to an opened one, perhaps the folder's icon turns into a silhouette filled with a pattern—and a new window is opened.

Examples of Spatial File Manager

Examples of file managers that to some extent use a spatial metaphor include:

- Apple’s Finder 5 to 9 (versions up to Mac OS X)
- RISC OS Filer
- Amiga’s Workbench
- GNOME’s Nautilus from version 2.6 onwards
- BeOS's Tracker
- OS/2’s Workplace Shell
- Digital Research's GEM (implemented in Atari TOS and as a somewhat reduced version for PCs)
- ZDESKTOP and FILEMAGE Zoomable File-System Viewers (spatial view of hierarchical data)

3.5.2 Dysfunctional Spatial File Managers

- Windows Explorer in Windows 95 was set as a spatial file manager model by default; but because it also worked as a navigational file manager, folders could be opened in multiple windows, which made it fail all the above criteria. Later versions gradually abandoned the spatial model.
- Apple's Finder in Mac OS X — much like in Explorer, the integration of spatial and navigational mode means that the spatial mode does not actually work.

3.6 3D File Managers

Some projects have attempted to implement a three-dimensional method of displaying files and directory structures. The exact implementation tends to differ between projects, as three-dimensional file browsing has not yet become popular and thus there are no common standards to follow.

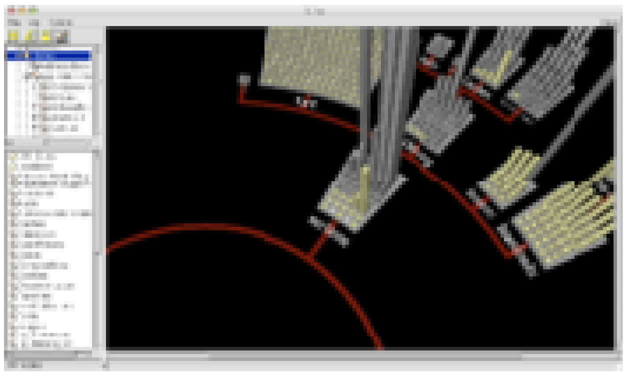


Fig 23: File System Visualizer, One example of a 3D File Manager.

Source: www.en.wikipedia.com

Examples of 3D File Managers

Examples of three-dimensional file managers include:

- fsn, for Silicon Graphics' IRIX systems, notably featured prominently in one scene from the film Jurassic Park, as a representation of Unix systems.
- File System Visualizer, or fsv, an open source clone of fsn for modern Unix-like systems.
- BumpTop, a file manager using a three dimensional representation of a desktop with realistic physics, intended for use with a stylus and touchscreen.
- Real Desktop, a desktop replacement with similarities to BumpTop.
- Knexus, a real-time 3D virtual library interface. 'Books' in the library act as symbolic links to files. The organisational structure is free-form similar to a three-dimensional Mind Map.

3.7 Web-Based File Manager

Web-based file managers are typically scripts written in PHP, Perl, Asp or any other server side languages. When installed on a local server or on a remotely hosted server they allow files and folders located there to be managed and edited without the need for FTP Access.

More advanced, and usually commercially distributed, web-based file management scripts allow the administrator of the file manager to configure secure, individual user accounts, each with individual account permissions. Authorized users have access to documents stored on the server or in their individual user folders anytime from anywhere via a web browser.

A web-based file manager can serve as an organization's digital repository. For example, documents, digital media, publishing layouts, and presentations can be stored, managed, and shared between customers, suppliers, remote workers or just internally

SELF-ASSESSMENT EXERCISE 2

1. Why is Navigational file manager very prominent?
2. Why is it that some Spatial File managers are termed “Dysfunctional”?

4.0 CONCLUSION

Various file management software are available each with its peculiar advantages and shortfalls. Adequate knowledge of the different groups will aid the decision to acquire which one will be best suited for users’ file management routines.

9.0 SUMMARY

In this unit, you have learnt that:

- A file manager or file browser is a computer program that provides a user interface to work with file systems.
- The most common operations that a file manager can perform are create, open, edit, view, print, play, rename, move, copy, delete, attributes, properties, search/find, and permissions.
- There are different types of file managers, each with its peculiar features, advantages and disadvantages.

10.0 TUTOR-MARKED ASSIGNMENT

1. Make a table and list, at least, five examples of five file managers.
2. Why is it that orthodox file managers are endearing?
3. Compare and contrast 3-D file managers and Navigational File managers.

7.0 REFERENCES/FURTHER READING

www.http://en.wikipedia.org/wiki/File_managers

www.en.wikipedia.org/wiki/Comparison_of_file_managers

www.lifehacker.com/399155/five-best-alternative-file-managers

UNIT 3 MANAGING FILES IN WINDOWS

CONTENTS

1.0 INTRODUCTION

As discussed in unit 2 of this module, there are as many file managers in the software market. In this unit, we learn how to manage files in Microsoft Windows Operating System. Also discussed are some useful tips for a successful management of electronic files.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- illustrate the different ways of managing files in windows environment
- organise and manage files using Windows Explorer, My Computer and even within an application
- explain clearly why it is better to store files properly than using the search function.

3.0 MAIN CONTENT

3.1 Management Files in Windows

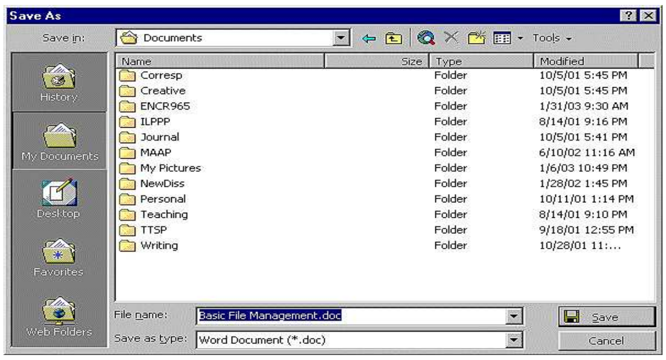
There are three ways of managing files in Windows operating systems:

- From within a Programme
- By using My Computer and
- By using Windows Explorer.

a. Managing Files from within a Programme

When you choose “File” → “Save As” from within a programme such as Microsoft Word, a dialogue box appears with three important features:

- “Save in” (near the top of the box)
- ➢ A dropdown box that brings up your computer’s directory structure, to allow you to choose where to save your file.
- “File name” (near the bottom of the box)
- ➢ Allows you to type in a name for your file.
- “Save as type” (at the bottom of the box)
- ➢ A dropdown box that allows you to choose a format (type) for your file. The default file format will appear with the default file extension.



Save in: Documents

Name	Size	Type	Modified
Corresp		Folder	10/5/01 5:45 PM
Creative		Folder	10/5/01 5:45 PM
ENCR965		Folder	1/31/03 9:30 AM
ILPPP		Folder	8/14/01 9:16 PM
Journal		Folder	10/5/01 5:41 PM
MAAP		Folder	6/10/02 11:16 AM
My Pictures		Folder	1/6/03 10:49 PM
NewDiss		Folder	1/28/02 1:45 PM
Personal		Folder	10/11/01 1:14 PM
Teaching		Folder	8/14/01 9:10 PM
TTSP		Folder	9/18/01 12:55 PM
Writing		Folder	10/28/01 11:...

File name: Basic File Management.doc

Save as type: Word Document (*.doc)

Fig 24: **Save As Dialog Box**

Source: *Captured from MS Windows Screen*

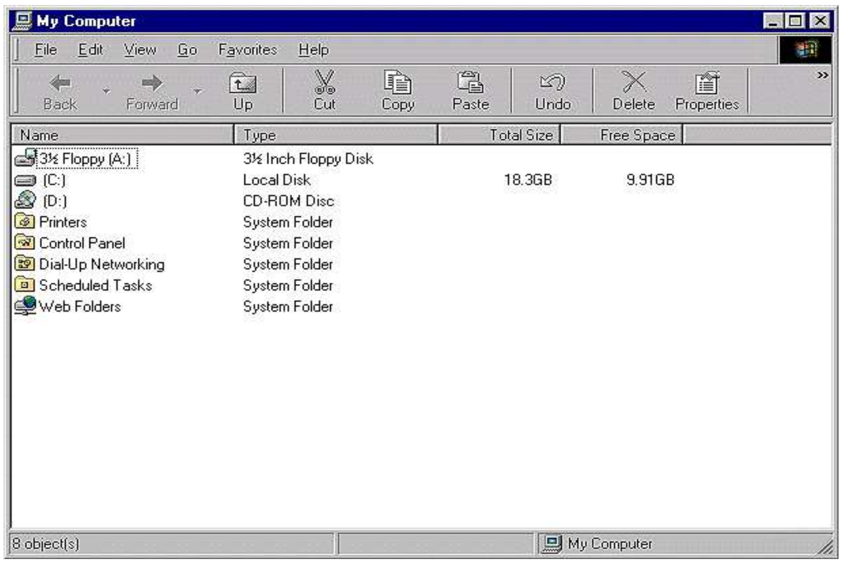
These three options also appear when you choose “File” → “Open” from within Microsoft Word, but they have slightly different names. Other programs will have the same three options, which again might have slightly different names.

b. Using My Computer

Fig 25: **My Computer Icon**

Source: *Captured from MS Windows Screen*

Double-clicking on the My Computer icon, which is located in the upper left-hand corner of your desktop, will open a window labelled My Computer. From within this window, you can open, move, copy, and delete files; you can also create, move, copy, and delete folders. Double-clicking on any folder icon also opens My Computer, but you will see the contents of that directory rather than the contents of your computer.



Name	Type	Total Size	Free Space
3½ Floppy (A:)	3½ Inch Floppy Disk		
(C:)	Local Disk	18.3GB	9.91GB
(D:)	CD-ROM Disc		
Printers	System Folder		
Control Panel	System Folder		
Dial-Up Networking	System Folder		
Scheduled Tasks	System Folder		
Web Folders	System Folder		

8 object(s) My Computer

Fig 26: My Computer Dialog Box

Source: *Captured from MS Windows Screen*

- At the “top” level of the directory structure are the drives, differentiated by letters:
- A:\ is your floppy disk drive
- C:\ is your hard disk
- D:\ is your Zip, CD, or DVD drive
- F:\ is probably your flash
- Go to “View” at the top of the window to change the way files and folders are displayed within the window. There are four ways to view files and folders:

➢ Large icons
➢ Small icons
➢ List – Choose this when you want to work with several files or folders at a time.
➢ Details – This is a good mode to work in when you want to see when the file was created, its size, and other important information.

- The toolbar has several buttons that enable you to work with files and folders:

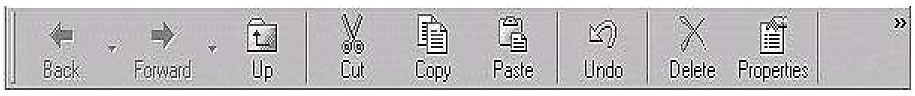


Fig 27: **Tool Bar Panel**

Source: *Captured from MS Windows Screen*

- **Up** – Choosing “Up” enables you to navigate through the computer’s directory structure quickly. Clicking on this button will change the contents of the current window, taking you “up” in the directory structure until you get to the highest level, at which only the drives are shown.

- **Cut** – When you single-click on a file or folder to select it, it will be highlighted in the window. Choosing “Cut” will delete the file or folder from its current location and copy it to the clipboard so that it can be pasted elsewhere.

- **Copy** – Choosing “Copy” will copy a selected file or folder into the clipboard so that it can be pasted elsewhere, but will not remove the file or folder from its current location.

- **Paste** – Choosing “Paste” will paste a file or folder that is stored in the clipboard into the current location.

- **Undo** – Choosing “Undo” allows you to undo an action that you have just performed. This is particularly useful when you have just deleted something you didn’t mean to delete.

- **Delete** – Choosing “Delete” will delete a selected file or folder without copying it to the clipboard.

- **Properties** – Choosing “Properties” will bring up a box that gives you information about a particular file or folder.

To create a new folder in the current window, you can do one of two things:

- Go to “File” “New” “Folder.” $\longrightarrow$ $\longrightarrow$

A new folder appears in the current window, and the folder name is highlighted that will allow you to name it.

- Right-click anywhere in the current window (not on an icon or filename) and choose “New” → “Folder.”
- Right-clicking on a selected file or folder will allow you to do several useful things, among which are the following:
- Rename a file or folder by choosing “Rename.” A blinking cursor will appear in the file or folder name.
- Create a desktop shortcut by choosing “Send To” → “Desktop as Shortcut.”
- Copy the file or folder to a floppy disk by choosing “Send To” → “3 ½ Floppy (A:).”
- Cut, copy, paste, or print a file.

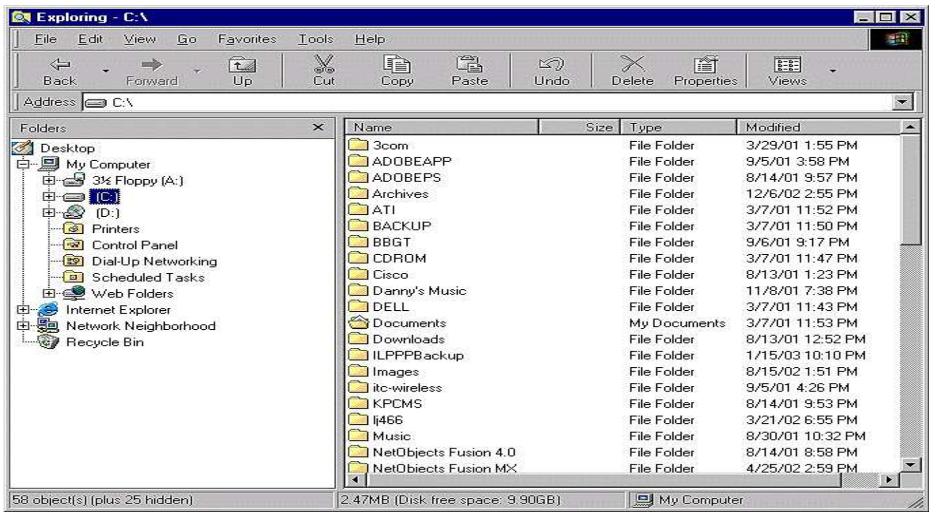
c. Using Windows Explore

Fig 28: Windows Explorer Icon

Source: *Captured from MS Windows Screen*

- In Windows Explorer, the entire directory structure is always available at all times in the left-hand pane. In this respect it differs from My Computer.
- Another difference between Windows Explorer and My Computer is that Windows
- Explorer allows you to drag-and-drop files and folders with the mouse.
- In the left-hand pane, drives, directories, and subdirectories are visible. To **expand** your view of the contents of a drive or directory, click on the **+** sign next to the directory name. To **collapse** your view of the contents of a drive or directory, click on the **–** sign next to the directory name.
- To see the contents of a drive or directory, click once on it (i.e., select it). In the right hand pane, the contents of the selected drive or directory are then displayed. The right hand pane functions just like the windows in My Computer.

In the example below, the drive C:\ is selected, and its contents are shown in the right-hand pane:


Exploring - C:\
File Edit View Go Favorites Tools Help
Back Forward Up Cut Copy Paste Undo Delete Properties Views
Address C:\
Folders

Desktop
My Computer
3½ Floppy (A:)
(C:)
(D:)
Printers
Control Panel
Dial-Up Networking
Scheduled Tasks
Web Folders
Internet Explorer
Network Neighborhood
Recycle Bin

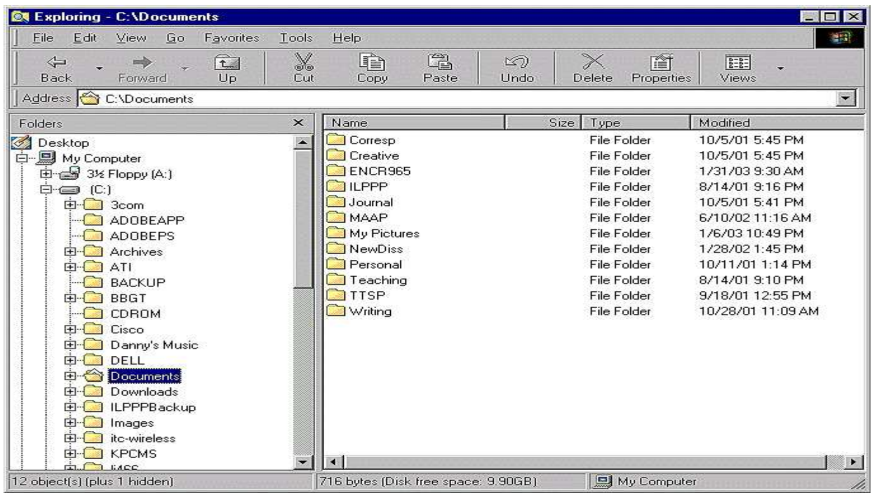
Name	Size	Type	Modified
3com		File Folder	3/29/01 1:55 PM
ADOBEAPP		File Folder	9/5/01 3:58 PM
ADOBEPS		File Folder	8/14/01 9:57 PM
Archives		File Folder	12/6/02 2:55 PM
ATI		File Folder	3/7/01 11:52 PM
BACKUP		File Folder	3/7/01 11:50 PM
BBGT		File Folder	9/6/01 9:17 PM
CDROM		File Folder	3/7/01 11:47 PM
Cisco		File Folder	8/13/01 1:23 PM
Danny's Music		File Folder	11/8/01 7:38 PM
DELL		File Folder	3/7/01 11:43 PM
Documents		My Documents	3/7/01 11:53 PM
Downloads		File Folder	8/13/01 12:52 PM
ILPPPBackup		File Folder	1/15/03 10:10 PM
Images		File Folder	8/15/02 1:51 PM
itc-wireless		File Folder	9/5/01 4:26 PM
KPCMS		File Folder	8/14/01 9:53 PM
lj466		File Folder	3/21/02 6:55 PM
Music		File Folder	8/30/01 10:32 PM
NetObjects Fusion 4.0		File Folder	8/14/01 8:58 PM
NetObjects Fusion MX		File Folder	4/25/02 2:59 PM

58 object(s) (plus 25 hidden) | 2.47MB (Disk free space: 9.90GB) | My Computer

Fig 29: Save As Dialog Box

Source: *Captured from MS Windows Screen*

In the next example, the drive C:\ has been expanded, and the directory "Documents" has been selected. Its contents are displayed in the right-hand pane:


Exploring - C:\Documents
File Edit View Go Favorites Tools Help
Back Forward Up Cut Copy Paste Undo Delete Properties Views
Address C:\Documents
Folders

Desktop
My Computer
3½ Floppy (A:)
(C:)
3com
ADOBEAPP
ADOBEPS
Archives
ATI
BACKUP
BBGT
CDROM
Cisco
Danny's Music
DELL
Documents
Downloads
ILPPPBackup
Images
itc-wireless
KPCMS

Name	Size	Type	Modified
Corresp		File Folder	10/5/01 5:45 PM
Creative		File Folder	10/5/01 5:45 PM
ENCR965		File Folder	1/31/03 9:30 AM
ILPPP		File Folder	8/14/01 9:16 PM
Journal		File Folder	10/5/01 5:41 PM
MAAP		File Folder	6/10/02 11:16 AM
My Pictures		File Folder	1/6/03 10:49 PM
NewDiss		File Folder	1/28/02 1:45 PM
Personal		File Folder	10/11/01 1:14 PM
Teaching		File Folder	8/14/01 9:10 PM
TTSP		File Folder	9/18/01 12:55 PM
Writing		File Folder	10/28/01 11:09 AM

12 object(s) (plus 1 hidden) | 716 bytes (Disk free space: 9.90GB) | My Computer

Fig 30: My Documents Dialog Box

Source: *Captured from MS Windows Screen*

3.2 Sorting Files

You can sort files in My Computer and Windows Explorer by clicking once on the Name, Size, Type, or Modified header buttons. To sort the files with the most recent listed first, for instance, click once on "Modified." To re-sort them with the earliest listed first, click again on "Modified."

3.3 Working with More than One File

To select two or more separate files, hold down the "Ctrl" key and click on each filename. To select a contiguous group of files in a list, click on the first filename, then hold down the "Shift" key and click on the last filename. All files in between will also be selected. You can then perform cut, copy, and delete functions on all the selected files.

3.4 Locating Lost File

Use the "Find File" facility of your operating system by going to "Start" $\longrightarrow$ "Find" $\longrightarrow$ "Files or Folders." A box will appear that will allow you to search for a file by name, by part of its name (use * as a wildcard character), by location, by date, by type, or by other criteria.

SELF-ASSESSMENT EXERCISE 1

1. Explain the steps on how you can files in My Computer and Windows Explorer?
2. List the different types of operations you can perform with right-clicking?

3.7 Tips for Management of Electronic Files

It is very important to keep the files on your computer organized and up-to-date. Just as with paper files, the goal of computer file management is to ensure that you can find what you're looking for, even if you're looking for it years after its creation. The following file management tips will be of help in keeping your files accessible:

- **Organise by file types.** Make applications easier to find by creating a folder called Program Files on your drive and keeping all your applications there. For instance, the executables for Word, PowerPoint, Simply Accounting and WinZip would all reside in the Program Files folder.
- **One place for all.** Place all documents in the My Documents folder and nowhere else. So whether it's a spreadsheet, a letter or a PowerPoint presentation, it goes here. This will make it easier to find things and to run backups.

- **Create folders in My Documents.** These are the drawers of your computer's filing cabinet, so to speak. Use plain language to name your folders; you don't want to be looking at this list of folders in the future and wondering what "TFK" or whatever other interesting abbreviation you invented means.
- **Nest folders within folders.** Create other folders within these main folders as need arises. For instance, a folder called "Invoices" might contain folders called "2004", "2005" and "2006". A folder named for a client might include the folders "customer data" and "correspondence". The goal is to have every file in a folder rather than having a bunch of orphan files listed.
- **Follow the file naming conventions.** Do not use spaces in file names, keep file names under 27 characters, and use all lower case. So a file named for a place should be nounabuja rather than Noun Abuja. If you break any of these rules, be consistent about it.
- **Be specific.** Give files logical, specific names and include dates in file names if possible. The goal when naming files is to be able to tell what the file is about without having to open it and look. So if the document is a letter to a customer reminding him that payment is overdue, call it something like "overdue081206" rather than something like "letter". How will you know who the letter is to without opening it? See the next point.
- **File as you go.** The best time to file a document is when you first create it. So get in the habit of using the "Save As" dialogue box to file your document as well as name it, putting it in the right place in the first place.
- **Order your files for your convenience.** If there are folders or files that you use a lot, force them to the top of the file list by renaming them with a ! or an AA at the beginning of the file name.
- **Cull your files regularly.** Sometimes what's old is obvious as in the example of the folder named "Invoices" above. If it's not, keep your folders uncluttered by clearing out the old files. Do NOT delete business related files unless you are absolutely certain that you will never need the file again. Instead, in your main collection of folders in My Documents, create a folder called "Old" or "Inactive" and move old files into it when you come across them.
- **Back up your files regularly.** Whether you're copying your files onto another drive or onto tape, it's important to set up and follow a regular back up regime.

The search function is a wonderful thing but it will never match the ease of being able to go directly to a folder or file. If you follow these file management tips consistently, even if you don't know where something is, you know where it should be – a huge advantage when it comes to finding what you are looking for.

SELF-ASSESSMENT EXERCISE 2

Outline five of the activities that can be performed with files and folders through Toolbar buttons

4.0 CONCLUSION

Working with electronic files is more efficient than working with paper files but it is far more rewarding if the various techniques and tips of organising and, managing of such files are known. It helps in easier and efficient information retrieval.

9.0 SUMMARY

In this unit, you have learnt that:

- We can organise and manage files in Windows through My Computer, Windows Explorer and even within an Application programme.
- Various activities can be performed with right-clicking a file
- Files can be sorted and be searched for on a computer system
- It is good to make use of some helpful tips in storing of electronic files.

10.0 TUTOR-MARKED ASSIGNMENT

1. What are the useful tips that can assist in efficient information storage and retrieval?
2. How do you manage files within an application programme?
3. What is the difference between the use of Control and Shift Keys when copying files?

7.0 REFERENCES/FURTHER READING

www.cti.itc..virginia.edu/2ttspeng/ basic file_management.pdf

www.westernu.edu/bin/computing/explorer_manging_files

UNIT 4 FILE SORTING, SEARCHING, AND MERGING

CONTENTS

1.0 INTRODUCTION

Computer or electronic files are stored in main memory or secondary storage devices. We may need to retrieve, arrange or manipulate these files for some purposes. There are, therefore, various techniques that are available for accomplishing all these tasks. This unit considers sorting and search algorithms.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain clearly concepts of sorting and searching
- describe different Sorting algorithms
- list different types of searching techniques
- state the merits and shortfalls of sorting and search algorithms
- appreciate importance of memory utilisation and involvement in search and sorting operations
- discuss file merging and why it is less popular than sort and search.

3.0 MAIN CONTENT

3.1 Sorting and Search Algorithms

Different types of algorithms abound in the literature with regards to file sorting, searching, processing and so on. Below is a discussion on two of the popular operations on file handling.

3.1.1 Sorting Algorithm

In computer science and mathematics, a **sorting algorithm** is a prescribed set of well-defined rules or instructions that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important to optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly. It is also often useful for canonicalising data and for producing human-readable output. More formally, the output must satisfy two conditions:

1. The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order);
2. The output is a permutation, or reordering, of the input.

Since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. For example, bubble sort was analyzed as early as 1956. Although many consider it a solved problem, useful new sorting algorithms are still being invented (for example, library sort was first published in 2004). Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures, randomized algorithms, best, worst and average case analysis, time-space tradeoffs, and lower bounds.

Summaries of Popular Sorting Algorithms are as follows:

- **Bubble Sort**

This is a sorting algorithm that continuously steps through a list, swapping items until they appear in the correct order. *Bubble sort* is a straightforward and simplistic method of sorting data that is used in computer science education. The algorithm starts at the beginning of the data set. It compares the first two elements, and if the first is greater than the second, it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass. While simple, this algorithm is highly inefficient and is rarely used except in education. For example, if we have 100 elements then the total number of comparisons will be 10000. A slightly better variant, cocktail sort, works by inverting the ordering criteria and the pass direction on alternating passes.

- **Insertion Sort**

This is a simple sorting algorithm that is relatively efficient for small lists and mostly-sorted lists, and often used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list. In arrays, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by one. Shell sort is a variant of insertion sort that is more efficient.

- **Shell Sort**

Shell sort was invented by Donald Shell in 1959. It improves upon bubble sort and insertion sort by moving out of order elements more than one position at a time. One implementation can be described as arranging the data sequence in a two-dimensional array and then sorting the columns of the array using insertion sort. Although this method is inefficient for large data sets, it is one of the fastest algorithms for sorting small numbers of elements.

- **Merge Sort**

Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements (i.e., 1 with 2, then 3 with 4...) and swapping them if the first should come after the second. It then merges each of the resulting lists of two into lists of four, then merges those lists of four, and so on; until at last two lists are merged into the final sorted list. Of the algorithms described here, this is the first that scales well to very large list.

- **Heap Sort**

Heap sort is a much more efficient version of selection sort. It also works by determining the largest (or smallest) element of the list, placing that at the end (or beginning) of the list, then continuing with the rest of the list, but accomplishes this task efficiently by using a data structure called a *heap*, a special type of *binary tree*. Once the data list has been made into a heap, the root node is guaranteed to be the largest (or smallest) element. When it is removed and placed at the end of the list, the heap is rearranged so the largest element remaining moves to the root.

- **Quick Sort**

Quick sort is a divide and conquer algorithm which relies on a *partition* operation: to partition an array, we choose an element, called a *pivot*, move all smaller elements before the pivot, and move all greater elements after it. This can be done efficiently in linear time and in-place. We then, recursively sort the lesser and greater sub-lists. Efficient implementations of quick sort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice. Because of its modest space usage, quick sort is one of the most popular sorting algorithms, available in many standard libraries. The most complex issue in quick sort is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower performance, but if at each step we choose the *median* as the pivot then it works with better performance.

- **Bucket Sort**

Bucket sort is a sorting algorithm that works by partitioning an array into a finite number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. Thus this is most effective on data whose values are limited (e.g. a sort of a million integers ranging from 1 to 1000). A variation of this method called the single buffered count sort is faster than quick sort and takes about the same time to run on any set of data.

- **Radix Sort**

Radix sort is an algorithm that sorts a list of fixed-size numbers of length k by treating them as bit strings. We first sort the list by the least significant bit while preserving their relative order using a stable sort. Then we sort them by the next bit, and so on from right to left, and the list will end up sorted. Most often, the counting sort algorithm is used to accomplish the bitwise sorting, since the number of values a bit can have is minimal - only ‘1’ or ‘0’.

- **Distribution Sort**

Distribution sort refers to any sorting algorithm where data is distributed from its input to multiple intermediate structures which are then gathered and placed on the output. It is typically not considered to be very efficient because the intermediate structures need to be created, but sorting in smaller groups is more efficient than sorting one larger group.

- **Shuffle Sort**

Shuffle sort is a type of distribution sort algorithm that begins by removing the first 1/8 of the *n* items to be sorted, sorts them recursively, and puts them in an array. This creates n/8 “buckets” to which the remaining 7/8 of the items are distributed. Each “bucket” is then sorted, and the “buckets” are concatenated into a sorted array.

3.2 Memory Usage Patterns and Index Sorting

When the size of the array to be sorted approaches or exceeds the available primary memory, so that (much slower) disk or swap space must be employed, the memory usage pattern of a sorting algorithm becomes important, and an algorithm that might have been fairly efficient when the array fit easily in RAM may become impractical. In this scenario, the total number of comparisons becomes (relatively) less important, and the number of times sections of memory must be copied or swapped to and from the disk can dominate the performance characteristics of an algorithm. Thus, the number of passes and the localisation of comparisons can be more important than the raw number of comparisons, since comparisons of nearby elements to one another happen at *system bus* speed (or, with caching, even at CPU speed), which, compared to disk speed, is virtually instantaneous.

For example, the popular recursive quick sort algorithm provides quite reasonable performance with adequate RAM, but due to the recursive way that it copies portions of the array it becomes much less practical when the array does not fit in RAM, because it may cause a number of slow copy or move operations to and from disk. In that scenario, another algorithm may be preferable even if it requires more total comparisons.

One way to work around this problem, which works well when complex records (such as in a relational database) are being sorted by a relatively small key field, is to create an index into the array and then sort the index, rather than the entire array. (A sorted version of the entire array can then be produced with one pass, reading from the index, but often even that is unnecessary, as having the sorted index is adequate.) Because the index is much smaller than the entire array, it may fit easily in memory where the entire array would not, effectively eliminate the disk-swapping problem. This procedure is sometimes called “*tag sort*”.

Another technique for overcoming the memory-size problem is to combine two algorithms in a way that takes advantages of the strength of each to improve overall performance. For instance, the array might be subdivided into chunks of a size that will fit easily in RAM (say, a few thousand elements), the chunks sorted using an efficient algorithm (such as quick sort or heap sort), and the results merged as per *merge sort*. This is less efficient than just doing merge sort in the first place, but it requires less physical RAM (to be practical) than a full quick sort on the whole array.

Techniques can also be combined. For sorting very large sets of data that vastly exceed system memory, even the index may need to be sorted using an algorithm or combination of algorithms designed to perform reasonably with virtual memory, i.e., to reduce the amount of swapping required.

SELF-ASSESSMENT EXERCISE 1

1. List five different sorting techniques.
2. Why is radix sorting technique different from others?

3.3 Search Algorithm

In computer science, a *search algorithm*, broadly speaking, is an algorithm that takes a problem as input and returns a solution to the problem, usually after evaluating a number of possible solutions. Most of the algorithms studied by computer scientists that solve problems are kinds of search algorithms. The set of all possible solutions to a problem is called the *search space*. *Brute-force search*, otherwise known as naïve or uninformed, algorithms use the simplest method of the searching through the search space, whereas informed search algorithms use *heuristic functions* to apply knowledge about the structure of the search space to try to reduce the amount of time spent searching.

3.3.1 Uninformed Search

An uninformed search algorithm is one that does not take into account the specific nature of the problem. As such, they can be implemented in general, and then the same implementation can be used in a wide range of problems due to abstraction. The drawback is that, in actual practice, many search spaces are extremely large, and an uninformed search (especially of tree or graph storage structures) will take a reasonable amount of time for even relatively small examples. As such, practical needs usually demand something better.

a. List Search

Lists and sequences generally, are perhaps the most commonly encountered data structures; search algorithms adapted for them are common as well. The goal is to find one element of a set (i.e., from the list) by some criterion (typically called a key and perhaps containing other information related to it). As this is a common problem in computer science, the computational complexity of these algorithms has been intensively studied.

The simplest such algorithm is *linear search*, which examines each element of the list as they are encountered. It is expensive in running time compared to many other algorithms. It can be used directly on any unprocessed list, regardless of history, which is sometimes useful. A more sophisticated list search algorithm is binary search. This is significantly better than linear search for large lists, but is sometimes useful for surprisingly small ones given its increase in speed. But it requires the list be sorted before searching and generally, that the list be randomly accessible. This may force lengthy reads of mass storage before a binary search can be started. Interpolation search is better than binary search for large sorted lists with ‘fairly even’ key.

Grover’s algorithm is a quantum algorithm which offers quadratic speedup over classical linear search for unsorted lists. However, it runs only on currently non-existent quantum computers.

Hash tables can also be used for list searches. They run in constant time in the average case, but have terrible worst-case time. In addition, more space and time is required to set up such tables. Another search based on specialized data structures uses self-balancing binary search trees. Such tree searches can be seen as extending the ideas of binary search to allow fast insertion and removal in the data structures.

Most list search algorithms, such as linear search, binary search, and self-balancing binary search trees, can be extended with little additional cost to find all values less than or greater than a given key, an operation called *range search*. One of such exception is hash tables, which cannot perform such searches efficiently.

b. Tree Search

Tree search algorithms are the likely the most used searching techniques for structured data. They examine trees of nodes, whether the tree is explicit or implicit (i.e., generated the search algorithm's execution). The basic principle is that a node is taken from a data structure, its successors examined and added to the data structure. By manipulating the data structure, the tree can be explored in different orders. For instance, level by level (i.e., breadth-first search) or reaching a leaf node first and backtracking (i.e., depth-first search), etc. Other examples of tree-searches include iterative-deepening search, depth-limited search, bidirectional search, and uniform-cost search.

The efficiency of a tree search is highly dependent upon the number and structure of nodes in relation to the number of items on that node. If there are a large number of items on one or more nodes, there may well be a requirement to use a specific different search technique for locating items within that particular set to attain adequate performance. In other words, a tree search is not mutually exclusive with any other search technique that may be used for specific sets. It is a method of reducing the number of relevant items to be searched to those within certain branches of the tree, thus reducing running time. For example, the Greater London telephone directory may contain entries for 20,000+ people whose surname is 'Smith' belonging on a tree branch on which are found 'surnames beginning S'. The list of names may, or may not be, further ordered (i.e., structured) by subscribers first names or initials. A binary search may be appropriate to locate a particular person with given name 'Alice' and perhaps thereafter a linear search to locate a particular address for a specific Alice Smith.

c. Graph Search

Many of the problems in graph theory can be solved using graph traversal algorithms, such as *Dijkstra’s algorithm*, *Kruskal’s algorithm*, the *nearest neighbour algorithm*, and *Prim's algorithm*. These can be seen as extensions of the tree-search algorithms.

3.3.2 Informed Search

In an informed search, a heuristic that is specific to the problem is used as a guide. A good heuristic will make an informed search dramatically out-perform any uninformed search. There are few prominent informed list-search algorithms. A possible member of that category is a hash table with a hashing function that is a heuristic based on the problem at hand. Most informed search algorithms explore trees. These include best-first search, and A*. Like the uninformed algorithms, they can be extended to work for graphs as well.

a. Adversarial Search

In games such as chess, programs typically maintain and recalculate as needed, a game tree of all possible moves by both players for the current position, and the resulting board configurations and we can search this tree to find an effective playing strategy. This type of problem has the unique characteristic that we must account for any possible move our opponent might make. Game-playing computer programs, as well as other forms of artificial intelligence like machine planning, often use search algorithms like the minimax algorithm, search tree pruning, and alpha-beta pruning.

b. Constraint Satisfaction

This is a type of search which solves constraint satisfaction problems rather than looking for a data value. The solution sought is a set of values assigned to a set of variables. Because the variables can be processed in any order, the usual tree search algorithms are not suitable. Methods of solving constraint problems include combinatorial search and backtracking, both of which take advantage of the freedom associated with constraint problems. Common tricks or techniques involved in backtracking are 'constraint propagation’, which is a general form of ‘forward checking'. Other local search algorithms, such as generic algorithm, which minimise the conflicts, may also be practical.

In the *minmax algorithm*, one takes first all the minimum values, then -- from them -- takes the maximum value. Naturally, it's the reverse for the *maxmin algorithm*.

3.4 Merge Algorithm

Merge algorithms are a family of algorithms that run sequentially over multiple sorted lists, typically producing more sorted lists as output. This is well-suited for machines with tape drives. Use has declined due to large random access memories, and many applications of merge algorithms have faster alternatives when a random-access memory is available.

The general merge algorithm has a set of pointers $p_{0...n}$ that point to positions in a set of lists $L_{0...n}$. Initially they point to the first item in each list. The algorithm is as follows:

- While any of $p_{0...n}$ still point to data inside of $L_{0...n}$ instead of past the end:

1. do something with the data items $p_{0...n}$ point to in their respective lists.
2. find out which of those pointers points to the item with the lowest key; advance one of those pointers to the next item in its list.

3.4.1 Analysis of Merge Algorithm

Merge algorithms generally run in time proportional to the sum of the lengths of the lists; merge algorithms that operate on large numbers of lists at once will multiply the sum of the lengths of the lists by the time to figure out which of the pointers points to the lowest item, which can be accomplished with a heap-based priority queue in O(log n) time, for O(m log n) time, where n is the number of lists being merged and m is the sum of the lengths of the lists. When merging two lists each of length m, there is a lower bound of $2m - 1$ comparisons required in the worst case.

The classic merge (the one used in merge sort) outputs the data item with the lowest key at each step; given some sorted lists, it produces a sorted list containing all the elements in any of the input lists, and it does so in time proportional to the sum of the lengths of the input lists. In parallel computing, arrays of sorted values may be merged efficiently using an all nearest smaller values computation.

3.4.2 Language Support

The C++'s Standard Template Library has the function std::merge, which merges two sorted ranges of iterators, and std::inplace_merge, which merges two consecutive sorted ranges *in-place*. In addition, the std::list (linked list) class has its own merge method which merges another list into itself. The type of the elements merged must support the less-than (<) operator, or it must be provided with a custom comparator.

PHP has the array_merge() and array_merge_recursive() functions.

SELF-ASSESSMENT EXERCISE 2

1. Explain the basic difference between informed and uninformed search algorithm.
2. In your own words, define sorting and searching.
3. Why is it that merge algorithm is not as popular as sort and search algorithm?

4.0 CONCLUSION

There are other algorithms that are available in sorting and searching of electronic files, but those discussed in this unit should be sufficient for now. Mathematical components of each algorithm were intentionally removed as they are thought to be beyond the scope of the course. The advent of random access memories has narrow down the research on merge algorithm because there are alternatives to applications using this algorithm.

5.0 SUMMARY

In this unit, you have learnt that:

- Efficient sorting is important to optimizing the use of other algorithms.
- Examples of sorting algorithms include bubble, insertion, shell, merge, quick, bucket, radix and distribution sorts.
- Memory utilisation and efficiency of operation are factors to consider in the choice of any sorting method.
- In searching, uninformed algorithms use the simplest method of searching through the search space.
- Informed search algorithms use *heuristic functions* to apply knowledge about the structure of the search space to try to reduce the amount of time spent searching.
- Merge algorithms are a family of algorithms that run sequentially over multiple sorted lists, typically producing more sorted lists as output.
- The use of merge Algorithm has declined due to the development of large random access memories, and many applications of merge algorithms have faster alternatives.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain how memory problems can be averted during sorting operation.
2. Write short notes on any four sorting techniques.
3. Discuss briefly on the searching technique used in game programming.
4. Discuss merge algorithm. Does is it have any relationship with sorting?

7.0 REFERENCES/FURTHER READING

Knuth, D. E. (1998). *The Art of Computer Programming*, Vol.3. Reading, UK: Addison-Wesley.

http://en.wikipedia.org/wiki/Merge_algorithm

http://en.wikipedia.org/wiki/Sort_algorithm

http://en.wikipedia.org/wiki/Search_algorithm

www.citeseer.ist.psu.edu/178743.html

www.ieeexplore.ieee.org/xpls/abs_all.jsp

UNIT 5 FILE HANDLING IN HIGH LEVEL LANGUAGES

CONTENTS

1.0 INTRODUCTION

Programming languages have facilities embedded in them to read such data values from a file stored in the computer memory or any storage device. The syntax differs from one programming language to another. The operating system has efficient ways of managing various files generated by any programming language. In this unit you will learn how Fortran, C, and C# (C Sharp) handle and process files.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- give reasons why it is convenient to write programme in high level languages than machine language
- ultimate i/o capabilities of some high level programme languages
- explain how and why programming languages essentially do the same thing if not for the syntax
- write out the codes to open and close files in the programming language.

3.0 MAIN CONTENT

3.1 High Level Programming Language

When a computer executes a programme, it executes a string of very simple operations such as load, store, add, subtract, and multiply. The operations are done in machine language which is a string of binary numbers, 0s and 1s. Unfortunately, we humans find machine language very difficult to work with. We prefer to work with English-like statements and algebraic equations that are expressed in forms familiar to us. We write out our instructions in a high level language and then use special programmes called **compilers** and **linkers** to convert the instructions into the machine language the computer understands.

Programmers use many different high level languages, with different characteristics. Some of them are designed to work well for business problems, while others are designed for general scientific use. Still others are especially suited for applications like operating systems programming. It is important to pick a proper language to match the problem that one is trying to solve.

3.1.1 Fortran File Handling Capability

FORTRAN has a standard structure for holding data that are used in its programmes. To use files within a FORTRAN programme, we will need some way to select the desired file and to read from or write to it. Fortunately, FORTRAN has a wonderful flexible method to read from and write to files. This mechanism is called the **input/output** unit. The i/o unit corresponds to the first asterisk in the READ (*,*) and WRITE (*,*) statements. If the asterisk is replaced by an i/o number, then the corresponding read or write will be to the device assigned to that unit instead of to the standard input or output device. The i/o number must be of integer value.

Several FORTRAN statements may be used to control disk file input and output and are presented in the following table:

Table 10: Fortran input/output statements

I/O Statement	**Function**
OPEN	Associate a specific disk file with a specific i/o unit number
CLOSE	End the association of a specific disk file with a specific i/o unit number
READ	Read data from a specified i/o unit number
WRITE	Write data to specified i/o unit number
REWIND	Move to the beginning of a file
BACKSPACE	Move back one record in a file

The OPEN statement associates a file with a given i/o unit number. Its form is

```
OPEN (open_list)
```

Where open_list contains a series of clauses specifying the i/o unit number, the file name, and information about how to access the file. The clauses in the list are separated by commas. There are other items in the list but let us discuss the five most important ones.

1. A `UNIT =` clause indicating the i/o unit number to associate with this file. This has the form
   ```
   UNIT = int_expr
   ```
   where `int_expr` can be non-negative integer value.

2. A `FILE =` clause specifying the status of the file to be opened. This clause has the form
   ```
   FILE = char_expr
   ```
   Where `char_expr` is a character value containing the name of the file to be opened.

3. A `STATUS=` clause specifying the status of the file to be opened. This clause has the form
   ```
   STATUS=  char_expr
   ```
   where `char_expr` is one of the following: `‘OLD’`, `‘NEW’`, `‘REPLACE’`, `‘SCRATCH’`, and `‘UNKNOWN’`.

4. An `ACTION=` clause specifying whether a file is to be opened for reading only, for writing only, or for both reading and writing. This clause has the form
   ```
   ACTION= char_expr
   ```

where `char_expr` is one of the following: `‘READ’,’ WRITE’`, or `‘READWRITE’`. If no action is specified, the file is opened for both reading and writing.

5. An `IOSTAT=` clause specifying the name of an integer variable in which the status of the open operation can be returned. This clause has the form

   `IOSTAT= int_var`

   where `int_var` an integer variable. If the OPEN statement is successful, a zero will be returned in the integer variable. If it is not successful, a positive number corresponding to a system error message will be returned in the variable.

The above clauses may appear in any order in the OPEN statement.

Let us have some programming samples to illustrate the input/output facility in Fortran.

The programme below reads values of x, y, and z using list directed i/o from the file INPUT.DAT and will write the values of variables x, y, and z to the file OUTPUT.DAT in the specified format.

```
OPEN  (UNIT=8,  FILE=’INPUT.DAT’,  STATUS=’OLD’,
IOSTAT=IERROR)
READ(8,*)X,Y,Z
OPEN(UNIT=9,                  FILE=’OUTPUT.DAT’,
STATUS=’REPLACE’, IOSTAT=IERROR)
WRITE(9,100) X,Y,Z
100 FORMAT (‘X = ‘, F10.2, ‘Y = ‘, F10.2,’ Z =
‘, F10.2)
```

The programme will write the values of the variables X,Y, and Z to the output file OUTPUT.DAT in the specified format.

```
PROGRAMME reads
! This programme is written in Fortran 90/95
!Purpose
! To illustrate how to read an unknown number of
values from an ! input data file. Detecting both
any formatting errors and the ! end of file.
!
IMPLICIT NONE
! Declare variables
CHARACTER(len=20) :: filename ! Name of file to
open
```

```
INTEGER :: nvals = 0          ! Number of values
read in
INTEGER :: status            ! I/O status
REAL:: value                 !  Real  value  read
in
!
!  Get  the  file  name  and  echo  it  back  to  the
owner
WRITE(*,*) ‘Please enter input file name’
READ(*,*) filename
WRITE(*,1000) filename
1000 FORMAT (‘ ‘, ‘The input file name is : ‘,
A)
!
! Open the file and check for errors on open.
OPEN(UNIT=3,FILE=filename,     STATUS=     ‘OLD’,
ACION=‘READ’,IOSTAT= Status)
Openif: IF (status = 0) THEN
     ! OPEN was okay. Read values.
     Readloop: DO
          READ(3,*,IOSTAT=status) value      !Get
next value
          IF (status/=0) EXIT                !Exit
if not valid
          nvals = nvals + 1             !Valid:
Increase count
          WRITE(*,1010) nvals, value         !Echo
to the screen
          1010  FORMAT(‘  ‘,’Line  ‘,I6,’:Value  =
‘,F10.4)
     END DO readloop
!
!  The  WHILE  loop  has  terminated.  Was  is  it
because  of  a  READ  error  or  !  because  of  the
input file?
readif:  IF  (status  >  0)  THEN  !  a  READ  error
occurred. Tell user.
     WRITE (*,1020) nvals + 1
     1020  FORMAT(‘0’,’An  error  occurred  reading
line ‘,I6)
ELSE !  the  end  of  the  data  was  reached.  Tell
user.
     WRITE(*,1030) nvals
     1030 FORMAT(‘0’,’ End of file reached. There
were ‘, I6, &
                   ‘values in the file’)
END IF readif
```

```
ELSE openif
     WRITE(*,1040) status
     1040 (‘ ‘, ‘Error opening file: IOSTAT =
’,I6)
END IF openif

! Close file
CLOSE (UNIT =8)
END PROGRAMME
```

Note that the input file is opened with `STATUS= ‘OLD’,` since we are reading from the file and the input data must already exist before the programme is executed.

3.1.2 Testing the Fortran Programme

We will create two input files: one with valid data and one with an input data error. We will run the programme with both input files and verify that it works correctly both for valid data and data containing errors. Also we will run the programme with an invalid file name to show that it can properly handle missing input files.

The valid input file is called `READ1.DAT`. It contains the following lines:

```
-17.0
30.001
  1.0
12000.
-0.012
```

The invalid input file is called `READ2.DAT`. It contains the following lines:

```
-17.00
30.001
ABCDEF
 12000.
-0.012
```

Running these files through the programme yields the following results:

```
C> read
Please enter input file name:
‘read1.dat’
The input file name is: read1.dat
Line     1: Value =        -17.0000
Line     2: Value =         30.0000
Line          3: Value =           1.0000
Line     4: Value =    12000.0000
Line          5: Value =        -.0120
```

```
End of file reached. There were 5 values in the
file.

C> read
Please enter input file name:
'read2.dat'
The input file name is: read2.dat
Line     1: Value =    -17.0000
Line     2: Value =     30.0000

An error occurred reading line   3
```

Finally, let us test the programme with an invalid input file name:

```
C> read
Please enter input file name:
'JUNK.DAT'
The input file name is: JUNK.DAT
Error opening file: IOSTAT =6416
```

The number of `IOSTAT` error reported by this programme will vary from processor to processor, but it will always be positive.

3.2 C File Input/Output

The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header <stdio.h>. The I/O functionality of C is fairly low-level by modern standards; C abstracts all file operations into operations on streams of bytes, which may be "*input streams*" or "*output streams*". Unlike some earlier programming languages, C has no direct support for random-access data files; to read from a record in the middle of a file, the programmer must create a stream, seek to the middle of the file, and then read bytes in sequence from the stream.

The stream model of file I/O was popularized by the Unix operating system, which was developed concurrently with the C programming language itself. The vast majority of modern operating systems have inherited streams from Unix, and many languages in the C programming language family have inherited C's file I/O interface with few if any changes (for example, PHP). The C++ standard library reflects the "stream" concept in its syntax.

3.2.1 Opening a File Using `Fopen`

A file is opened using **`fopen`**, which returns an I/O stream attached to the specified file or other device from which reading and writing can be done. If the function fails, it returns a null pointer. The related C library function **`freopen`** performs the same operation after first closing any open stream associated with its parameters.

They are defined as:

```
FILE *fopen(const char *path, const char *mode);

FILE *freopen(const char *path, const char
*mode, FILE *fp);
```

3.2.2 Closing a Stream Using `fclose`

The `fclose` function takes one argument: a pointer to the `FILE` structure of the stream to close.

```
int fclose(FILE *fp);
```

The function returns zero on success, or EOF on failure.

The following programme opens a file named `sample.txt`, writes a string of characters to the file, then closes it.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(void)
{
    FILE *fp;
    size_t count;
    const char *str = "hello\n";

    fp = fopen("sample.txt", "w");
    if(fp == NULL) {
        perror("failed to open sample.txt");
        return EXIT_FAILURE;
    }
    count = fwrite(str, 1, strlen(str), fp);
    printf("Wrote %zu bytes. fclose(fp) %s.\n",
count, fclose(fp) == 0 ? "succeeded" :
"failed");
    fclose(fp);//close de file
    return EXIT_SUCCESS;
}
```

The following C program opens a binary file called *myfile*, reads five bytes from it, and then closes the file.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char  buffer[5]  =  {0};    /*  initialized  to
zeroes */
  int i, rc;
  FILE *fp = fopen("myfile", "rb");
  if (fp == NULL) {
    perror("Failed to open file \"myfile\"");
    return EXIT_FAILURE;
  }
  for  (i=  0;(rc  =  getc(fp))!=  EOF  &&  i  <  5;
buffer[i++]= rc) ;
  fclose(fp);
  if (i == 5) {
    puts("The bytes read were...");
    printf("%x   %x   %x   %x   %x\n",   buffer[0],
buffer[1], buffer[2],  buffer[3], buffer[4]);
  } else
    fputs("There   was   an   error   reading   the
file.\n", stderr);
  return EXIT_SUCCESS;
}
```

SELF-ASSESSMENT EXERCISE 1

1. Why do we write programmes in high level languages?
2. List the operations that can be performed on Fortran files
3. Write a line to open a file for both input and output in Fortran language

3.3 Text File Operations in C#

C-Sharp provides a File class which is used in manipulating text files. The File class is within the System namespace. Also we can use the StreamReader and StreamWriter classes, which are within the System.IO, namespace for reading from and writing to a text file. In this article we will see examples of creating a text file, reading contents of a text file and appending lines to a text file.

3.3.1 Creating a Text File

For creating text file we use the CreateText Method of the File Class. The CreateText Method takes in the path of the file to be created as an argument. It creates a file in the specified path and returns a StreamWriter object which can be used to write contents to the file.

Example:

```
public class FileClass
{
    public static void Main()
    {
    WriteToFile();
    }
    static void WriteToFile()
    {
    StreamWriter SW;
    SW=File.CreateText("c:\\MyTextFile.txt");
    SW.WriteLine("God is greatest of them all");
    SW.WriteLine("This is second line");
    SW.Close();
     Console.WriteLine(“File              Created
     SuccessFully”);
    }
}
```

3.3.2 Reading Contents of a Text File

For reading the contents of a text file we use the OpenText Method of the File class. The OpenText Method takes in the path of the file to be opened as an argument. It opens the specified file and returns a StreamReader object which can be used to read the contents of the file.

Example:

```
public class FileClass
{
    public static void Main()
    {
    ReadFromFile("c:\\MyTextFile.txt");
    }
    static void ReadFromFile(string filename)
    {
    StreamReader SR;
    string S;
    SR=File.OpenText(filename);
    S=SR.ReadLine();
    while(S!=null)
    {
    Console.WriteLine(S);
    S=SR.ReadLine();
    }
    SR.Close();
    }
}
```

3.3.3 Appending Content to a Text File

For appending content to a text file we use the AppendText Method of the File class. The AppendText method takes in the path of the file to which the contents to be appended as an argument. It opens the file in the specified path and returns a StreamWriter object which can be used to append contents to the file.

Example:

```
public class FileClass
{
    public static void Main()
    {
    AppendToFile();
    }
    static void AppendToFile()
    {
    StreamWriter SW;
    SW=File.AppendText("C:\\MyTextFile.txt");
    SW.WriteLine("This Line Is Appended");
    SW.Close();
    Console.WriteLine("Text              Appended
Successfully");
    }
}
```

SELF-ASSESSMENT EXERCISE 2

1. How can you open and close a file using C language?
2. List 10 high level programming languages you know.

4.0 CONCLUSION

It could be very difficult for anyone without a prior knowledge of any programming language to understand the codes in this unit. Understanding the codes line by line is not important but understanding the idea of what they are doing.

5.0 SUMMARY

In this unit, you have learnt that:

- It is not easy or feasible to write programme in machine language
- There are many programming languages, but it is important to choose a language that is best suited for the problem one is trying to solve
- Different languages have different file handling structures.
- In any language, to use a file it must be opened or created and must be disconnected from the program after execution.
- The files are external to the programme because they are stored in the computer memory.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is the importance of linkers and compilers to a programmer?
2. Write out the codes to open and close files in the three languages discussed in the unit
3. What are the various specifiers in Fortran OPEN syntax?

7.0 REFERENCES/FURTHER READING

CHAPMAN, S. J. (1998). Fortran *90/95 for Scientists and Engineers*. Boston: McGraw-Hill.

KING, Melvyn. (1995). *A First Course in Computer Programming Using C*. London: McGraw- Hill.

McGREGOR, Jim (1998). *Simple C*. England: Addison-Wesley

www.livephysics.com/.../fortran/fortran-file-handling.html

www.ruf.rice.edu/~statlab/fortran

www.cprogramming.com/tutorial/cfileio.html

www.cpp-home.com/archives/67.htm

www.devhood.com/Tutorials/tutorial_details.aspx

www.codeguru.com/csharp/csharp/cs_syntax