File Systems and Management

Prof P.C.P. Bhatt

Introduction

- Much of the work within the computer system can be seen as *management of processes and files.*
- Files are the *primary means of accessing* the information.
- The Storage of files may be in the *main memory* or in *secondary memory (disk memory).*
- *OS* allows users to manage such files with a *file system*.

What are files?

- Irrespective of the content, any *organized information* is a file, e.g. - a telephone numbers list, web images or data logged from an instrument, all are files.
- In *UNIX files* are arbitrary *bit (or byte) streams.*
- A *file system* is that software which allows users and applications to organize their files

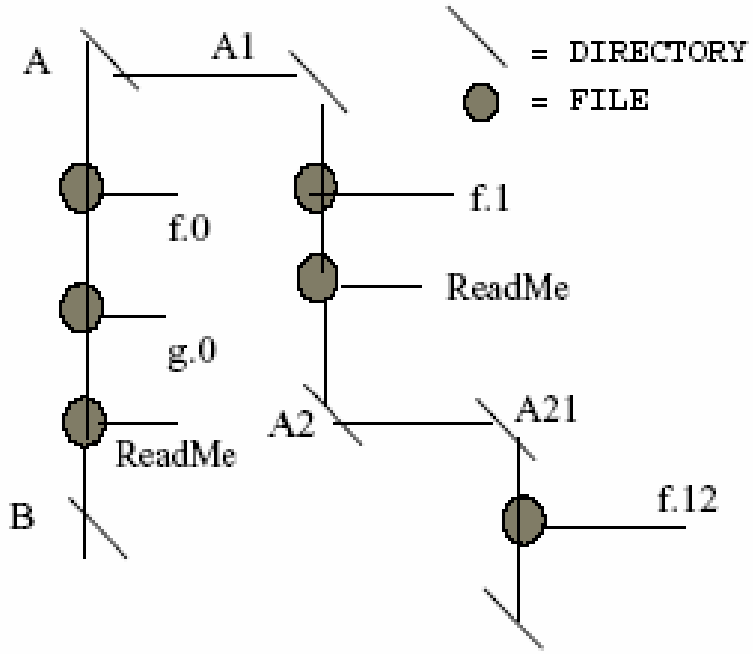
User's view of files

At this stage it is important to have a perspective which is relevant to the user, so we will give user's view of files

- First need (of a user) is to be able to *access* a file.
- The file system must be able to *locate* the file sought … *identify* the file by its *name.*
- *File names* have *extensions* (e.g. .c, .obj) which define the file type.
- Related files are organized into *directories.*
- Within a directory, each file must have a *unique name.*

Directory and File Organization

We have a tree structure amongst directories. Files form leaves in the tree structure of directories.



A and B are top level directories.

A1 and A2 are subdirectories under A

f.0 and g.0 are files under directory A.

f.1 is a file under subdirectory A1.

File f.12 has a full path name A/A2/A21/f.12.

Also, note the two ReadMe files. Each is in a different directory. There can be only one file with the name ReadMe in a directory.

File Types

- Most OS use *file types* in their *id.*
- *File descriptor* in the file structure used by the file management software to help OS provide file management services.
- MAC OS typically stores this information in its *resource fork* – done to let OS display the icons of the application environment in which the file was created.
- PDP-11 used an octal 0407 as a *magic number* to identify executable files.
- File system stores other information such as *location* of the file etc.

File Operations - 1

- *User* performs *various operations* – read, write, save, retrieve, display, append, copy, delete etc. with files.
- *User* changes file *attributes* such as access permissions of files.
- Unix provides a visual editor called *vi* to view text files as well as edit them.
- Unix provides the *ls* command to get a file listing. The *ls* command has several options for file listing.

File Operations - 2

Choose Option	To get this information
none	Lists files in a single column
-l	Lists long revealing file type, access permission, last access time etc.
-d	For each named directory, list directory information.
-a	Lists all files, including those with a “.”
-s	Sizes of files in blocks occupied

Ls command options

File Extensions

- Most OS *generate or use a file extension.*
- For Example

Windows System

.exe	Executable file
.doc	Word file
.txt	Text file
.bmp	image

UNIX System

.c	C Program
.o	Executable
.tex	LaTex

Using Regular Expressions

- Most *OSs allow* use of *regular expression* operations with commands – gives command flexibility, e.g. *, ?, +
- Various *options* with each command also *give flexibility*, e.g. *rm –r, ls –la*
- Files may be used in multiple contexts, changes made in one must appear in the other copy also. Creating symbolic links alleviates this problem – *ln* command.

  ln filea linkfilea

File Access Permissions

- A file system manages access by checking a file's *permissions*.
- A file may be accessed to perform *read, write* or *execute* operations.
- The usage is determined by the *context in which* the file was created.

An Example

- Consider a *file* supporting an application of *bus schedules*. It shall contain a bus *time-table* with the following restrictions :
  - ✓ *read-only* permission for the *general public*,
  - ✓ *read, write* permission for the supervisor,
  - ✓ *read, write and execute* permissions for the management.

Who all can access a File?

- Unix recognizes three category of users – *user/owner*, *group* and *others*.
- *Owner* may be a *person* or a *program* or an application or a system based utility. The *notion of a group* comes from the *software engineering* team operations.
- Others has the *connotation of public usage.*
- Organization of this information is as 9 bits *r w x r w x r w x* for each owner, group and others where each *r w x* is an octal number, e.g 100 100 100 gives read permission for owner, group and others.

File Access and Security

In Unix, owner of a file can change its permissions using *chmod* command which has a syntax as follows:

chmod pattern fileName as in *chmod* 644 myFile

e.g : 644 corresponds to the pattern rw-r--r--

664 corresponds to the pattern *rw-rw—r--*

ls –l command displays the access permissions of a file.

Security Concerns

- *File permissions* are the most elementary and effective form of *security* measure *in a stand alone single user system.*
- Some systems provide *security* by having *passwords* for files.
- Enhanced security would be to *encrypt* a file with some *key.*
- Unix provides crypt command to encrypt files. Syntax is : *crypt EncryptKey <InputFile> OutputFile*

Information Required for File Management - 1

Nature of Information	Its significance	Its use in management
File Name	Chosen by its creator user or program.	To check its uniqueness within a directory
File Type	Text, binary, program etc.	To check its correct usage
Date of creation and last usage	Time and date	Useful for recording identity of user(s) also
Current usage	Time and date	Identity of all current users
Back-up information	Time and date	Useful for recovery following a crash

Information required for File Management - 2

Permission	rwx information	Controls read, write, execute, useful for network access also
Starting address	Physical mapping	Useful for access
Size	User must allocate within allocated space.	Internal allocation of disc blocks
File Structure	Useful in data manipulation	To check its usage

File Storage Management - 1

An OS needs to maintain several pieces of information for file management, e.g. access and modification times of a file.

- *Audit Trail* gives who accessed which file and did what.
- In Unix, these trails are maintained in the *syslog* file – useful to recover files after a system crash and to detect unauthorized accesses to a system.
- Unix internally recognizes 4 different file types *ordinary, directory, special and named.*
- Ordinary files are those created by the users programs or utilities.

File Storage Management - 2

- Directory files organize the files hierarchically, they are different from ordinary files.
- *Inode* structure is used by Unix to maintain information about named files.

Inode Structure in Unix - 1

File Type	16 bit information Bits 14-12 : File type (ordinary, directory, character etc.) Bits 11-9 : Execution flags. Bits 8-6 : Owners r w x information Bits 5-3 : Groups r w x information Bits 2-0 : Others r w x information
Link Count	Number of symbolic references to this file
Owner's ID	Log in ID of the person who owns this file.
Group's ID	Group ID of the user

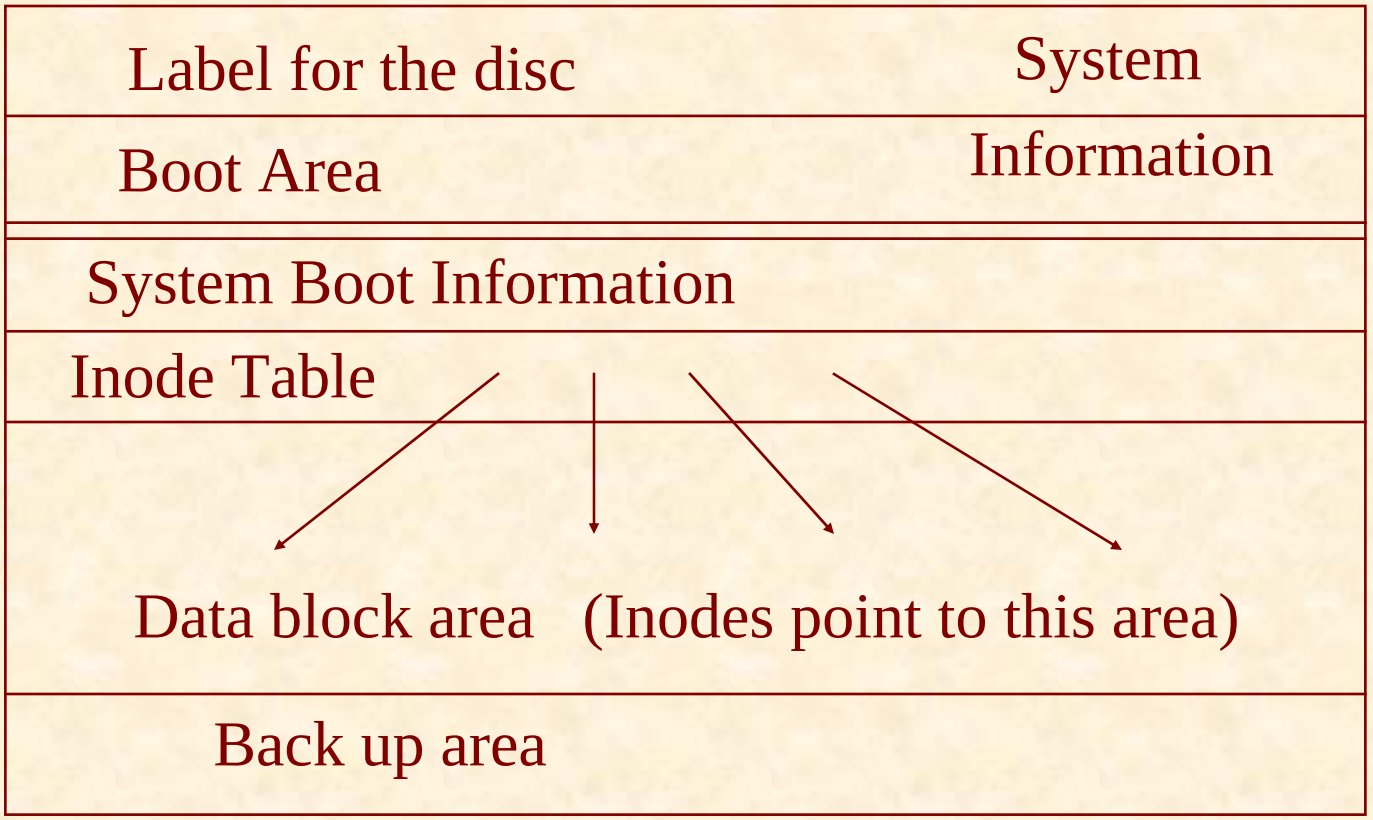
Inode Structure in Unix - 2

File Size	Expressed in number of bytes.
File Address	39 bytes of addressing information.
Last access to file	Date and time of last access.
Last modified	Date and time of last modification.
Last inode modification	Date and time of last inode modification.

File Control Blocks

- The Microsoft counterpart of an inode is a *File Control Block (FCB).*
- The FCBs store *file name, location of secondary storage, length of a file in bytes, date and time of its creation* etc.

Organization of Inodes
(Similarly for FCBs)



The Root File System - 1

- The root file system is created when *Unix is installed.*
- The root file system has a *directory tree structure.*
- The root of the root file system in Unix flavored OSs is identified by *'/'*.
- The OS specifies how the system and user files should be distributed for space allocation on the disc storage.

The Root File System - 2

- System files are programs executable with the .bin and .exe extensions in Unix and Microsoft OS respectively.
- The main advantage of the root file system is that the system knows where to look for some specific routines.

Conventions followed in Unix - 1

- Subdirectory *usr* contains binaries sharable by system and user - *read-only* usage mode.
- Executable programs are found in subdirectory *bin* found at any level.
- Subdirectory *sbin* contains some system executable files – used during boot time and power on.

Conventions followed in Unix - 2

- Subdirectory *lib* contains libraries – found at several places in the file system. Subdirectory *etc* contains host related files – has many subdirectories to store configuration (*config*), internet (*hosts*) and device (*dev*) information.
- Subdirectory *mnt* contains device mount information (in Linux).
- Subdirectory *tmp* contains temporary files created during file operation.
- Subdirectory *var* contains files that have variable data E.g. *mail* and *system log* contain variable data. It may also have subdirectories for spools.

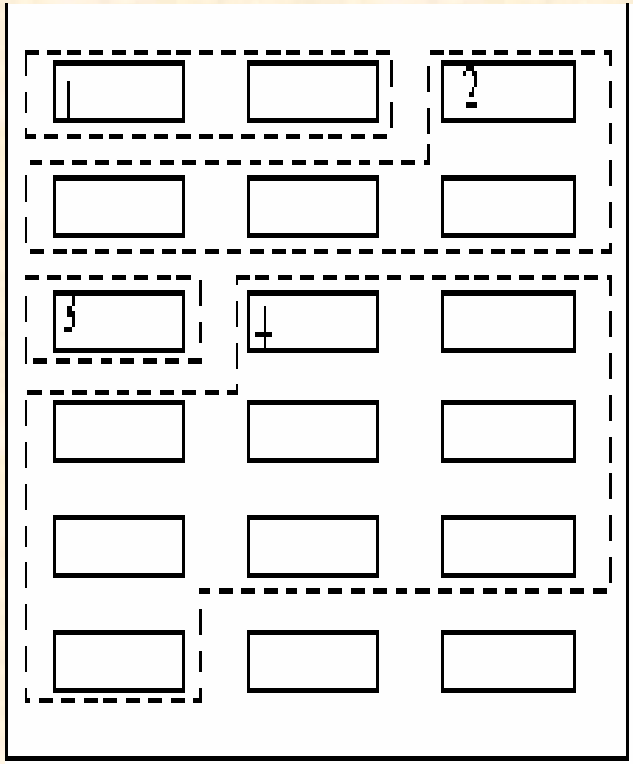
Conventions followed in Unix - 3

- All of X related support are in the subdirectory *X11*.
- Each user *name* will find his home directory as */home/name*
- Subdirectory *include* contains all C header files.
- Subdirectory marked *yp* (yellow pages) has network information – provides data-base support for network operations.

Block Based File Organization - 1

- We *next* discuss organization of *files as blocks* of information.
- In this section we shall discuss *various techniques* used in the organization of *blocks*.

Contiguous Allocation - 1



Files : File_1 (size 1145 bytes)

File_2 (size 4060 bytes)

File_3 (size 700 bytes)

File_4 (size 9000 bytes)

If we know *apriori* the size of the file to be created, this information can be given to OS for it to follow a *pre-allocation* policy and find a suitable memory block that can fit the entire file as a contiguous block.

Contiguous Allocation - 2

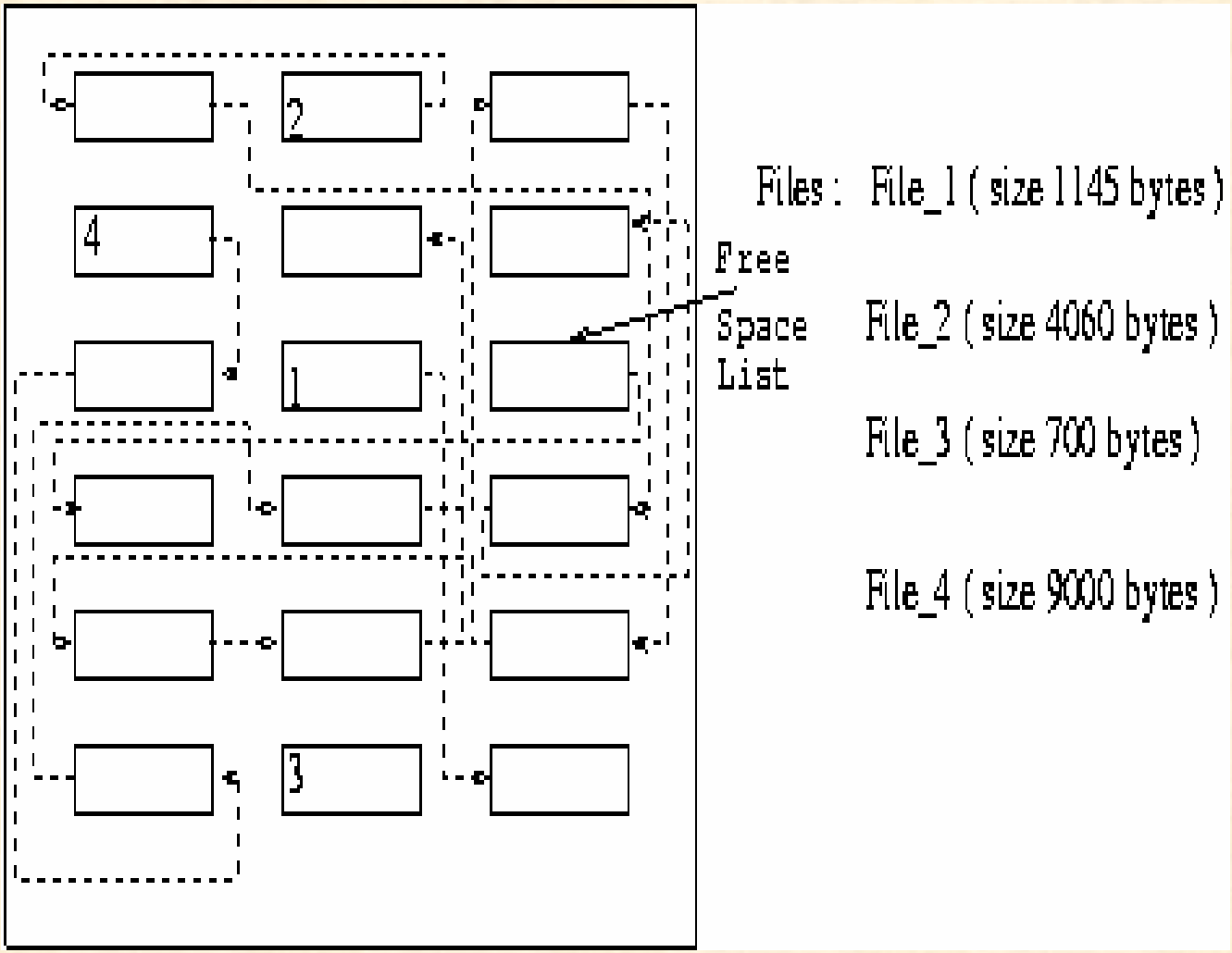
- The *numbers 1, 2, 3 and 4* in the previous figure *identify the starting blocks* of the four files.
- One *advantage* of the pre-allocation policy is that the *retrieval of information* is *very fast.*
- One disadvantage of this policy is that it *requires apriori information of the size* of the file.
- Other disadvantage is that it might *not be possible* to find a *contiguous memory* block always.

  Also, note this is a *static allocation*.

Chained List Allocation - 1

- This is a *dynamic block allocation* policy that overcomes the disadvantages of pre-allocation policy.
- The *disadvantage* of *dynamic allocation* is that *random access* to blocks is not possible.

Chained List Allocation - 2

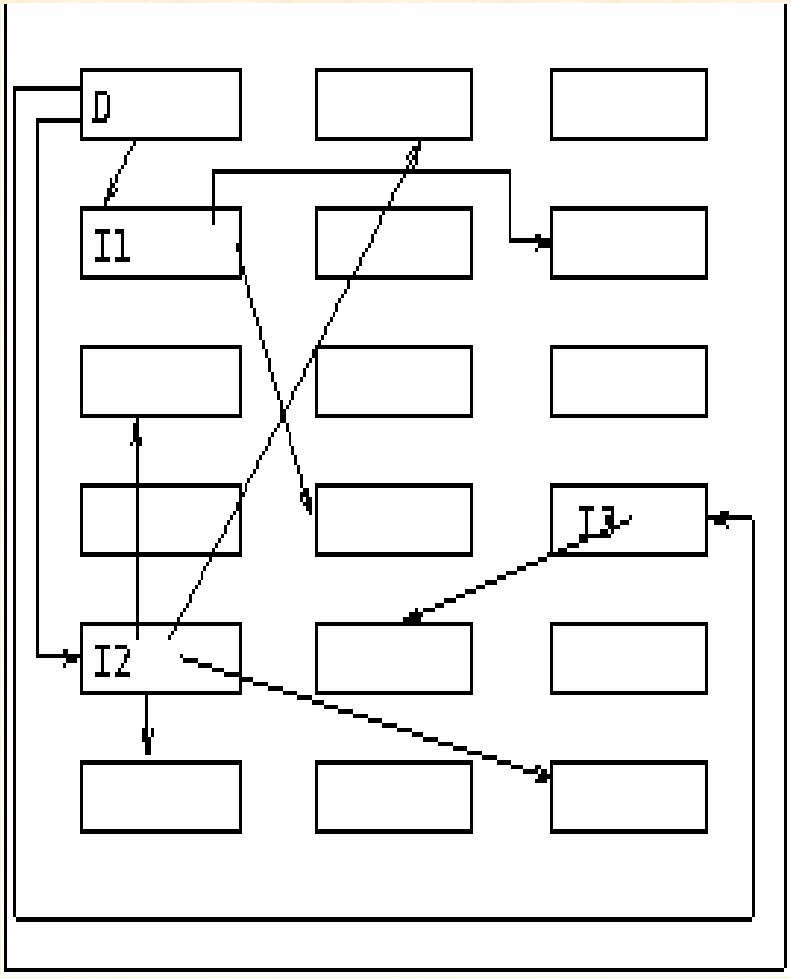


Chained Allocation

Indexed Allocation - 1

- An *index file* for each file in its *first block* is maintained.
- Thus *address information* for each block can be *obtained* with *one level of indirection*.
- The *advantage* of this method is that there is a *direct access to any part of the file.*

Indexed Allocation - 2



Files : File_1 (size 1145 bytes)

File_2 (size 4060 bytes)

File_3 (size 700 bytes)

Indexed Allocation

Internal and External Fragmentation

- In mapping byte streams to blocks, block size was assumed to be 1024 bytes.
- In the previous example, for a file size of 1145 bytes, 2 blocks were allocated – 1024 + 121 bytes in the second block. Such *non-utilization of space caused internally* is called *Internal Fragmentation*.

Internal and External Fragmentation

- Most *OSs maintain a free space list* to allocate blocks as needed.
- Suppose a file was initially 7 blocks after which it was reduced to 4 blocks, a *hole* of 3 blocks is produced. Due to such holes, shortage of memory occurs due to non-utilization of holes called *External Fragmentation*.

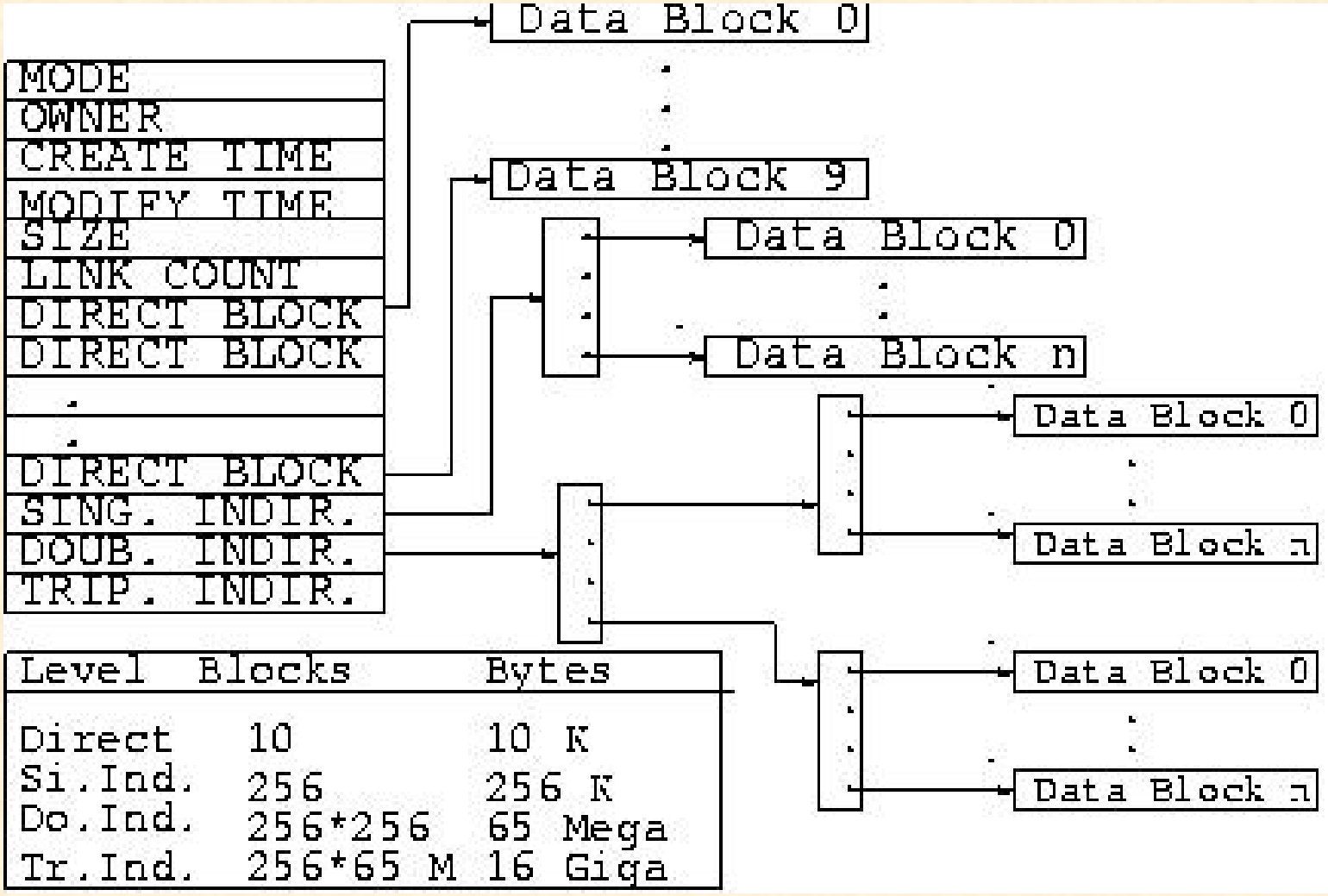
Policies in Practice - 1

- *MSDOS* and *OS2* use a *FAT* (File Allocation Table) strategy where stores entries for files for each directories (similar to *index node* in *Unix*).
- *File name* is used to *get* the *starting address of the first block* of *a file.*
- Each file block is *chained linked to* the *next block till* an *EOF* is stored in some block
- *FAT maintains* a *list* of *free block chains*
- *FAT* was stored in the *first few blocks* of disc space.

Policies in Practice - 2

- *Extension of FAT – FAT32* is *supported* on *Windows98* and higher. FAT32 in addition supports longer file-names and file compression.
- Other version of FAT on *Windows NT* is *NTFS*.
- Unlike FAT, *NTFS* spreads the file *tables* throughout the discs for their *efficient* management.
- Like *FAT32*, supports *long file-names* and *file compression*.
- File access permissions are supported by *NTFS*. *Windows2000* uses *NTFS*.

Storage Allocation in Unix - 1



Storage Allocation in Unix - 2

Unix was designed for *large scale program development* with team effort and hence supports group access to very large files at very high speeds.

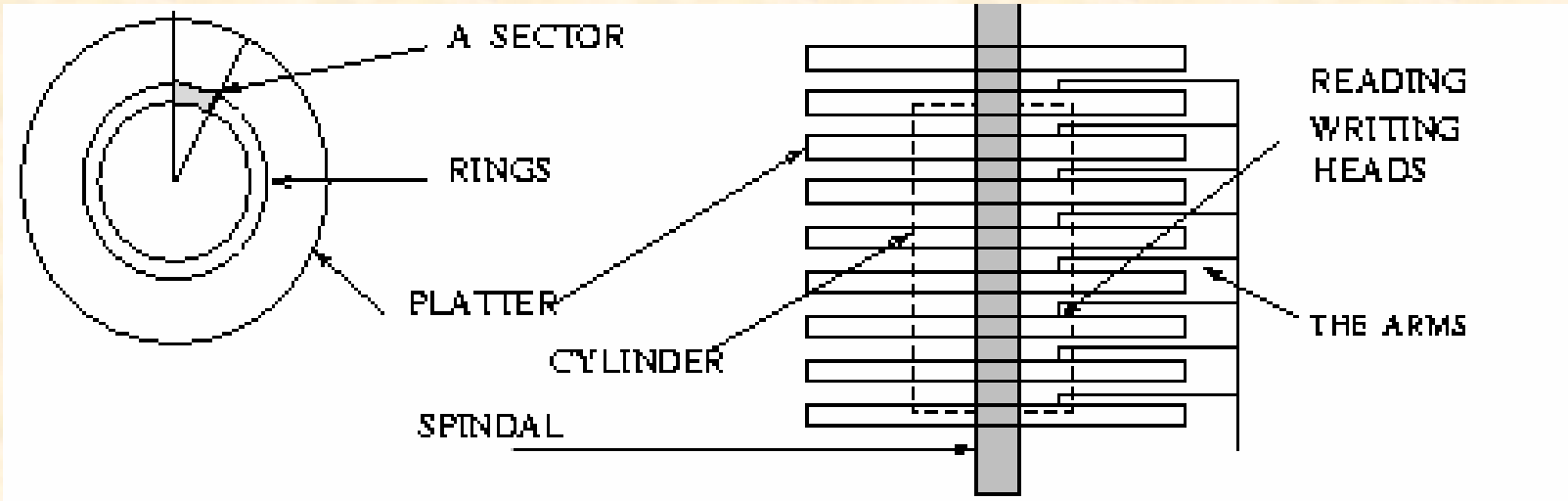
File *information in Unix* is *stored in two parts:*

- *Part 1:* has information about the mode of access, symbolic links, owner and times of creation and last modification.
- *Part 2:* is a 39 byte area within inode structure. These 39 bytes are 13, 3 byte pointers. First 10 of these point to the first 10 blocks of the file.

Storage Allocation in Unix - 3

Note: If the files are longer then the other 3, 3 byte addresses are used for indirect indexing. So the 11^{th} 3 byte points to the real data. If the file is still longer, the 12^{th} byte points and so on.

Physical Layout of Information on Media - 1



NOTE THAT THE RINGS ON THE DISC PLATTERS ON THE SPINDLE FORM A CYLINDER SINCE ALL HEADS ARE ON A PARTICULAR RING AT THE SAME TIME SO IT IS EASY TO ORGANISE INFORMATION ON A CYLINDER. THE INFORMATION IS STORED IN THE SECTORS THAT CAN BE IDENTIFIED ON THE RINGS. SECTORS ARE SEPARATED FROM EACH OTHER. ALL SECTORS CAN HOLD EQUAL AMOUNT OF INFORMATION

Physical Layout of Information on Media - 2

- In *previous figure*, tracks may be *envisaged as rings on a platter*.
- *Each ring* is *capable* of *storing 1 bit along* its *width*.
- These *rings* are *broken into* sectors…necessary

because of the *physical nature* of the *control required*

to let the system recognize where the blocks begin in a disc.

Physical Layout of Information on Media - 3

- *Individual characters cannot be identified* because the *discs move* at *very high speeds*.
- With *multiple discs mounted* on a *spindle*, they *form a cylinder* with *tracks equidistant from* the *center*.
- These *cylinders* can be *given continuous block sequence numbers* to *store information.*

Disc Partitions

- Allows *better management of disc* space.
- *Unix* maintains *disc partitions* to *house system, kernel and user files. Windows* too *partitions the hard disc.*
- *Disc partitions* are *mounted* on a *file system.*
- A *disc partition* is organized into a *directory structure* - tree.
- This tree gets connected to some node in the overall file system tree – *mounting a file system*
- This basic concept is carried on in case of file servers on a network also.

Portable Storage

- External media types are tapes, media, floppy diskettes.
- They can be physically ported.
- Unix treats these as special files.
- PCs and MAC OS show an icon when these are mounted.

Why do we mount a device - 1

In Unix,

- All non-removable discs are mounted automatically on booting.
- Removable discs are mounted explicitly.
- A *mount point* is specified which signifies the point in the Unix directory tree where the removable disc files get attached.

Why do we mount a device - 2

- Since Unix has a *multi-user environment*, removable discs are secured – a CDROM once mounted becomes a part of the tree and cannot be accidentally removed.
- In Windows, removable discs are *automatically mounted* when they are accessed.

Why do we mount a device - 3

- During LINUX installation, the *setup asks* for *mount points* for the logical drives/partitions.

For *example*,

"*/*" is the mount point for the *root*

"/…/…/" i.e. conceptually an arbitrary node can be given as a *mount point for another drive etc.*