Nanyang Polytechnic

IT2164/IT2561 Operating Systems and Administration

Chapter 10

File Management

Objectives

- After this lesson, you will be able to :
  - Understand the structure of low-level files
  - Know the design of high-level file abstraction
  - Understand the three disk allocation methods
  - Know how a typical file system is structured

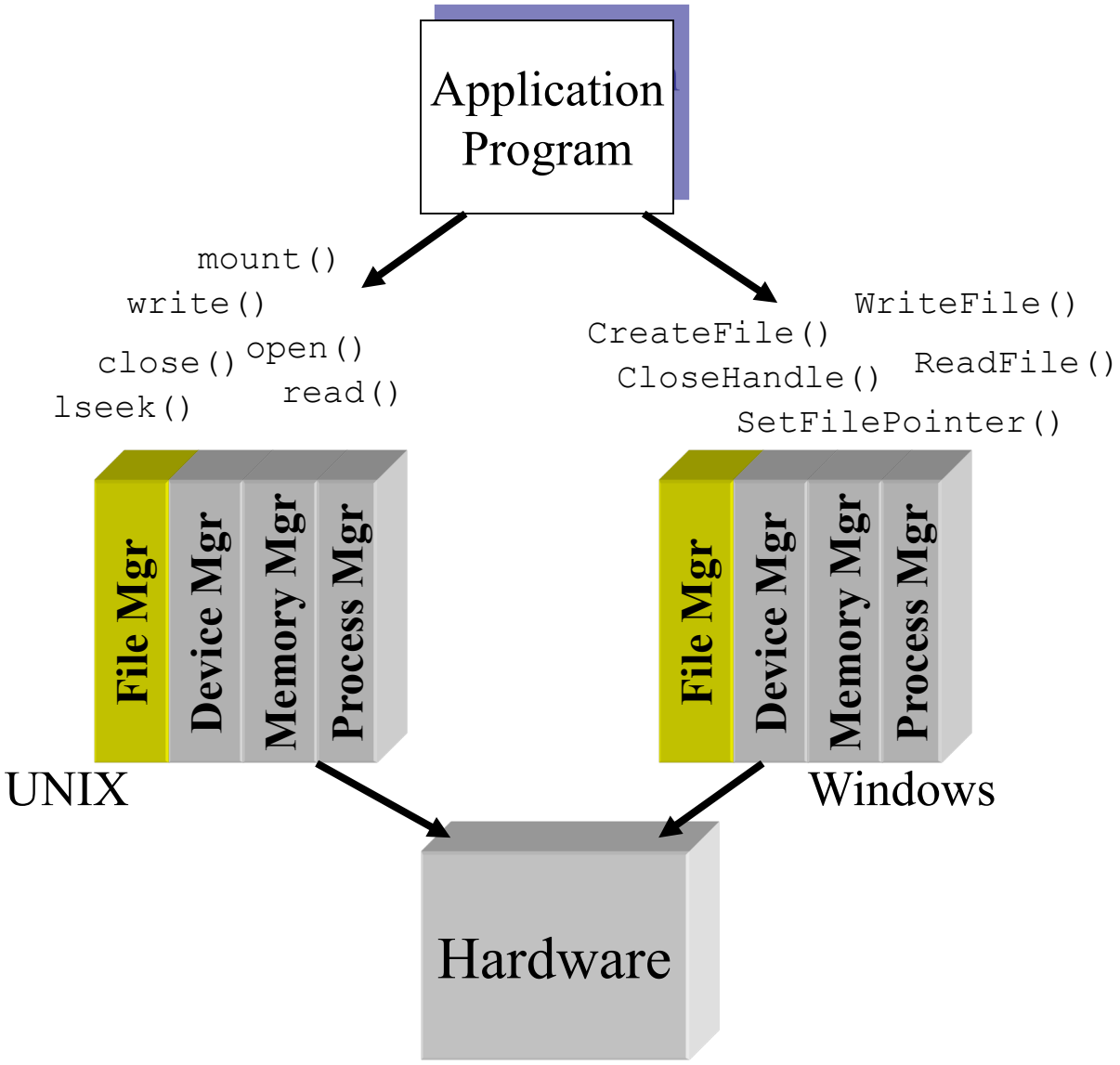
The Need for Files

- A computer system requires to store data in a non-volatile storage system.
- Although the computer system is good at computation, stable storage of data is required in order to perform meaningful tasks, e.g., storing company data, application programs, etc.
- Files are the OS mechanism for organizing and managing the data in a computer system for storage.
- Types of files include executable, data, text (including html), among many others.

Files

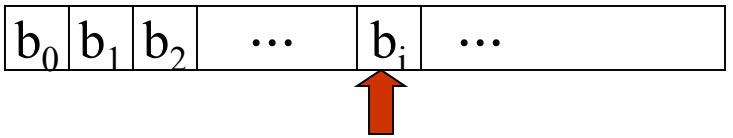
- Almost every application needs to read from or write to files.
- To support file handling, almost all programming languages provide file handling routines.
- These routines, when activated, execute the OS system calls to handle the files.
- The manager in the OS that handles files is the file manager.

View of the File Manager



Low Level Files

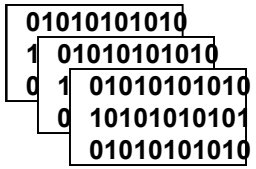
```
fid = open("fileName",…);
…
read(fid, buf, buflen);
…
close(fid);
```




```
int open(…) {…}
int close(…) {…}
int read(…) {…}
int write(…) {…}
int seek(…) {…}
```

Stream-Block Translation

Storage device response to commands



data blocks

Block Management

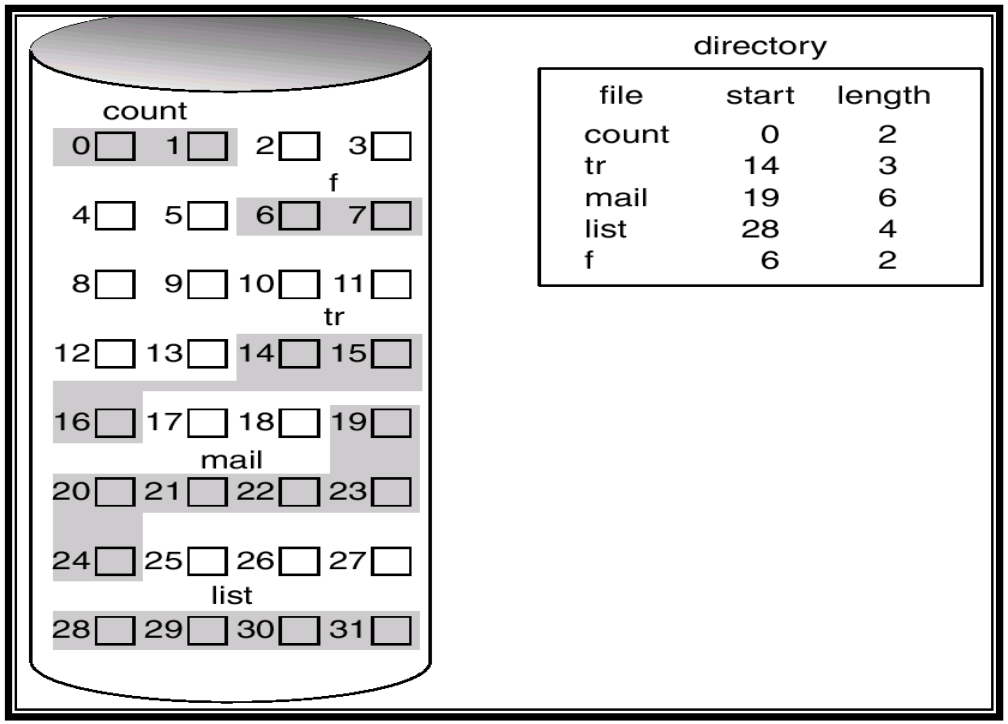
- In an actual storage system, data is stored in blocks, usually of equal size.
- Three well-known techniques of organizing these blocks :
  - As a contiguous set of blocks.
  - As a list of blocks interconnected with links.
  - As a collection of blocks interconnected by a **file index**.

Contiguous Allocation

- This method requires each file to occupy a set of contiguous blocks on the disk.
- Disk addresses define a linear ordering on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the file.
- Directory entry for each file indicates the address of starting block and length of the area allocated to this file.
- If the file is n blocks long and start at location b, then it occupies b, b+1, b+2, ..., b+n-1.

Contiguous Allocation

- In the example below, f is allocated 2 consecutive blocks while mail is allocated 6 consecutive blocks.



Contiguous Allocation Advantages

- Support both sequential and direct access
  - For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.
  - For direct access to block i of a file that starts at block b, we can immediately access block b + i
- Minimal disk seek time
  - As disk addresses are defined in a linear ordering on the disk, accessing block b+1 after block b normally requires no head movement. Moreover, when head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder) it is only one track. Thus the number of disk seeks required for accessing contiguously allocated files is minimal.

Contiguous Allocation Disadvantages

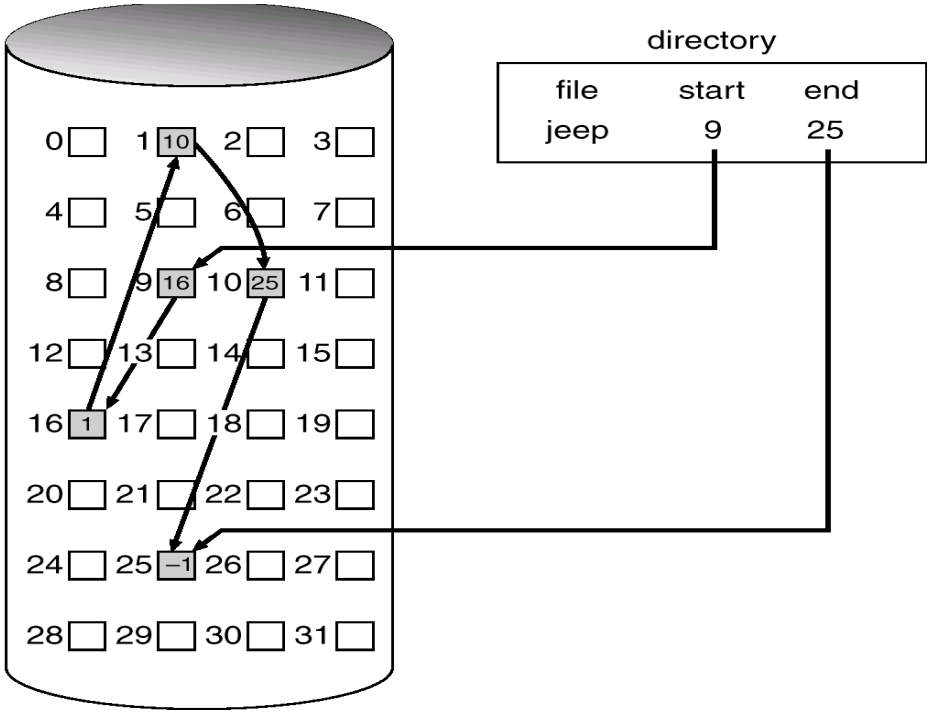
- External Fragmentation
  - As files are allocated and deleted, the free disk space is broken into little pieces thus resulting in non-contiguous blocks.
  - The solution to this problem is to perform compaction on the disk blocks. All the free space is compacted into one large hole. However, the cost of compaction is time.
- Unknown file size
  - When a file is created, the total amount of space it needs must be allocated, but in some cases, example, output file, the size is not known and may be difficult to extend later.
- Known file size
  - Even if the file size is known in advance, pre-allocation may be insufficient because file will grow over a long period of time.

Linked Allocation

- Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.
- To create a file, we simply create a new entry in the directory and the pointer to the first disk block of the file is initialized to nil (the end-of-list pointer value) to signify an empty file.
- A write to a file removes the first free block from the free-space list and linked to the end of the file.
- To read a file, simply follow the pointers from block to block.

Linked Allocation - Example

- In the example below, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally block 25.



Linked Allocation - Advantages

- Disk space need not be contiguous
- No external fragmentation
  - Any free block on the free-space list can be used to satisfy a request, since all blocks are linked together.
  - There is no necessity to compact disk space.
- No need to declare the size of file
  - The size of a file can grow as long as there are free blocks

Linked Allocation - Disadvantages

- Inefficient for direct access
  - To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get to the ith block.
- Pointer space
  - Extra space in each block is required to store the pointers. Thus, each file would require slightly more space.
- Reliability problem
  - As the files are linked together by pointers scattered all over the disk, consider what would happen if a pointer was lost or damaged. A bug in the OS or a disk hardware failure might result in picking up the wrong pointer thus accessing the wrong block.

Indexed Allocation

- Linked allocation does not support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order. Indexed allocation solves this problem by bringing all the pointers together into one location: the index block.

- Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block.

- To read the ith block, use the pointer in the ith index block entry to find and read the desired block.

- When the file is created, all pointers in the index block are set to nil. When the ith block is first written, a block is removed from the free-space list and its address is put in the ith index-block entry.

Indexed Allocation - Advantages

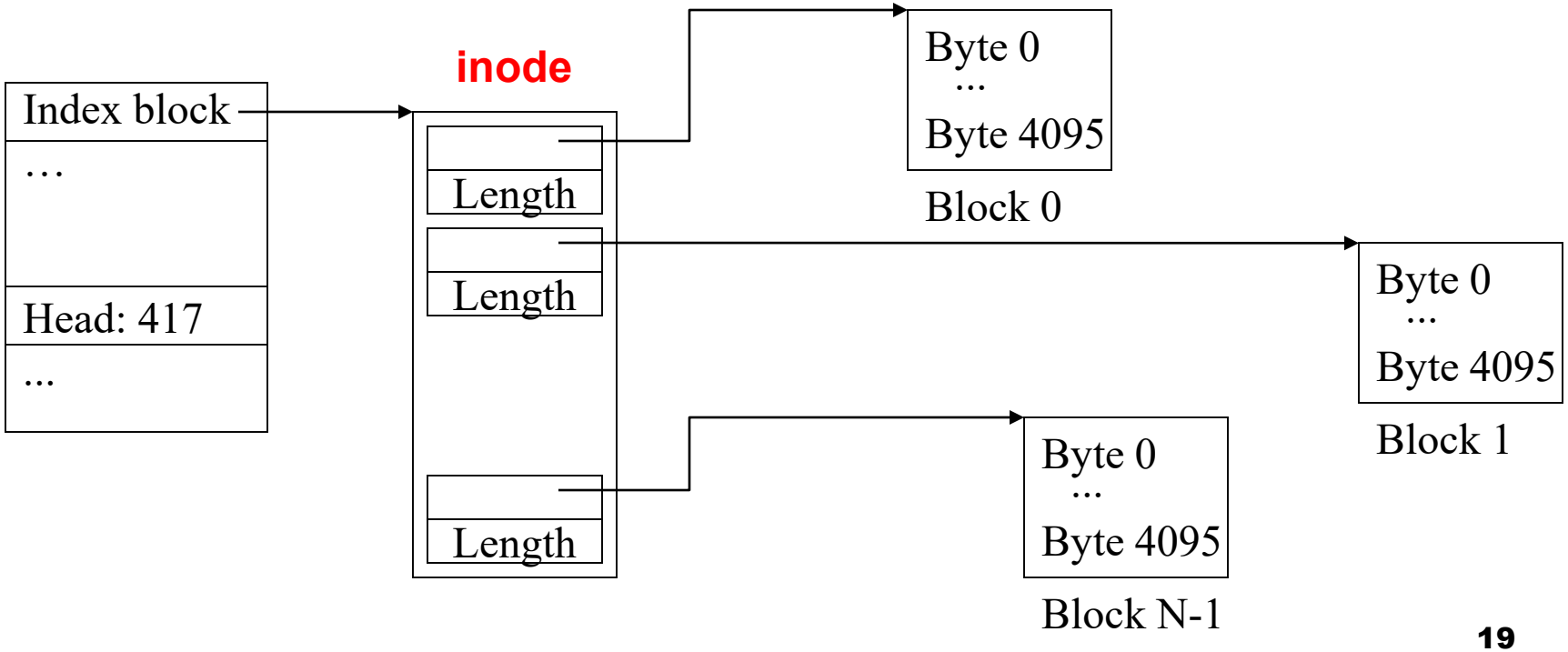
- Supports direct access
- No external fragmentation as any free block on the disk may satisfy a request for more space

Indexed Allocation - Disadvantages

- Wasted space
  - The pointer overhead of index block is generally greater than the pointer overhead of linked allocation.
  - An entire index block is allocated to a file even if only one or two pointers are used.
  - Inevitably, we have to decide on the size of the index block (If index block is too large, space are wasted. Too small an index block will not be able to hold enough pointers for large file).

Indexed Allocation

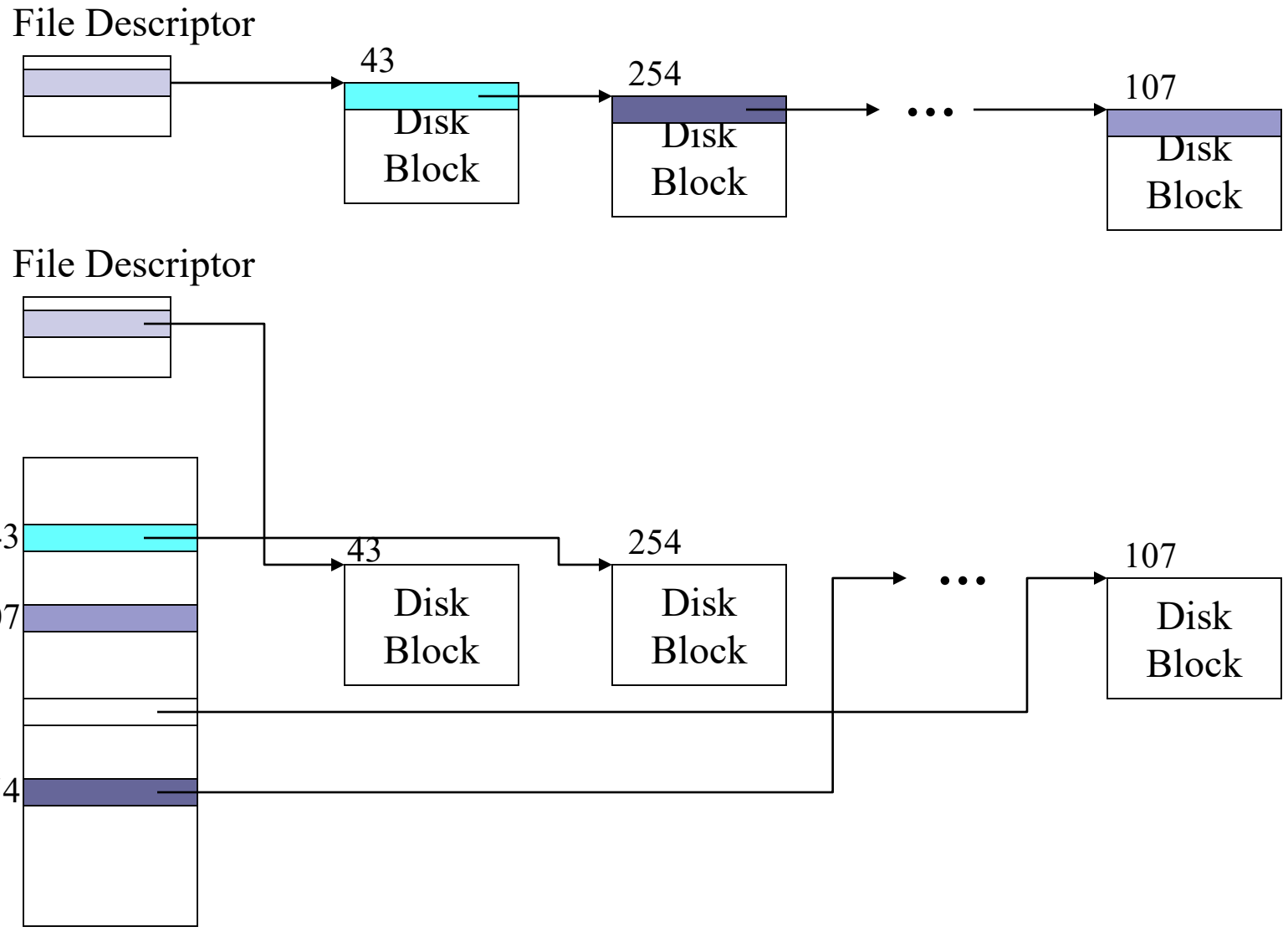
- Extract headers and put them in an index
- Simplify seeks
- May link indices together (for large files)



Linked Allocation - FAT (NTFS)

- An important variation on the **linked allocation** is the use of File-Allocation Table (FAT).
- The use of a FAT is a simple but efficient method of disk-space allocation used by MS-DOS(Windows) and OS/2 operating systems. A section of disk at the beginning of each partition is set aside to contain the table. The table has one entry for each disk block and is indexed by block number. The FAT is used much as is a linked list.
- The directory entry contains the block number of the first block of the file. The table entry indexed by the block number then contains the block number of the next block in the file. This chain continues until the last block, which has a special end-of-file value as the table entry. Unused blocks are indicated by a 0 table value.
- FAT allocation scheme results in significant number of disk head seek, unless the FAT is cached.
- Important limitation is the size of the file system is limited. This has resulted in various improvements in the FAT system, including FAT32. But this requires one to upgrade the file system, which can be sensitive.

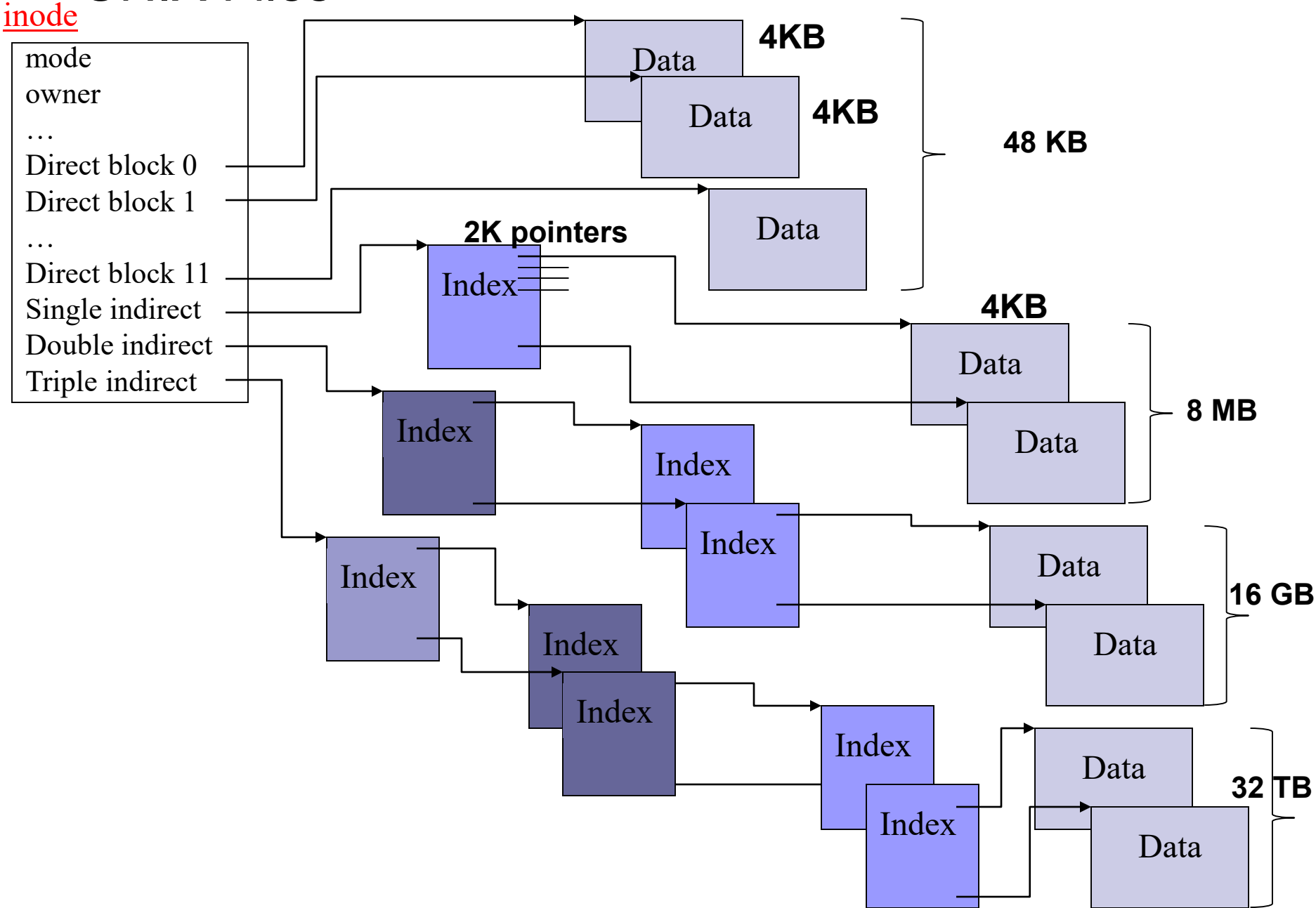
DOS FAT Files



Indexed Allocation – UNIX File Structure (Extended file formats :Ext2, Ext3, Ext4)

- UNIX/Linux uses a variant of the indexed allocation scheme.
- Uses indirect indices to manage large files.
- Free of the limitations placed by FAT, in particular, the maximum size of the file system.

UNIX Files


inode
mode
owner
…
Direct block 0
Direct block 1
…
Direct block 11
Single indirect
Double indirect
Triple indirect
Data
4KB
Data
4KB
48 KB
Data
2K pointers
Index
4KB
Data
Data
8 MB
Index
Index
Index
Index
Data
Data
16 GB
Index
Index
Index
Index
Data
Data
32 TB

File Descriptors

- The file manager, in addition to storing the files, also manages the various information that describe the files. (file descriptors).
  - External name
  - Current state
  - Sharable
  - Owner
  - User
  - Locks
  - Protection settings
  - Length
  - Time of creation
  - Time of last modification
  - Time of last access
  - Reference count
  - Storage device details

Directories

- File manager not only needs to provide mechanisms for applications to create, write, read files, it also needs to provide ways to manage *collections* of files.
- A file directory is a set of logically associated files and other sub-directories of files.
- We use directories to help us organize *logically* the different files that we may handle.
- For e.g., we can store the files associated with a project in one directory and those with another project in another directory.

Directories

- The file manager provides a set of commands to allow users to administer directories, including :
  - Enumerate: Returns a list of all the files and nested directories. E.g., dir in Windows, or ls in UNIX/Linux.
  - Copy: Create a duplicate of an existing file. cp
  - Rename: Changes the name of the file. mv
  - Delete: Removes the file from the directory. Can also be used to delete directories. rm
  - Traverse: Allows a user to explore the directories by 'moving' into or out of directory structures. cd, pwd

Directories

- In a directory, files can be sorted by their various **attributes**, including
  - Name,
  - Size,
  - Last access time,
  - owner, among many others.

Directory Structures

- How should files be organized within directory?
  - Flat name space
    - All files appear in a single directory
    - Not good once number of files get too large, or exceed a threshold, say 20 files.
  - Hierarchical name space
    - Directory contains files and subdirectories
    - Each file/directory appears as an entry in exactly one other directory -- a *tree*
    - Popular variant: All directories form a tree, but a file can have multiple parents.
    - Popular as it allows users to subdivide files into multiple directories, which themselves may be nested.

Hierarchical Directories

- Files can be stored in directories, and so can directories.
- Directories can be stored into a ‘super’ directory of directories.
- Files can be ordered within the directory and so can sub-directories.

Hierarchical Directories (Linux)

Standard sub-directories are:

S/N	Sub-directory	Description
1.	/bin	Contains user executable programs. For example, the **ls** program is located in /bin.
2.	/sbin	Contains system executable programs used by the root user and the system. For example, the **clock** program is located in /sbin.
3.	/lib	Contains shared library files used by /bin and /sbin.
4.	/dev	Contains special file system entries for devices attached to the system.
5.	/boot	Contains the Linux kernel and bootloader programs. The Linux kernel program is typically known as "vmlinuz".
6.	/etc	Contains system configuration files. Files contain user account information are located here.
7.	/proc	Contains special files pertaining to the state of the running Linux system. These files are virtual files.
8.	/mnt	Contains temporarily mounted file systems.
9.	/usr	Contains programs that can be run any users of the system.
10.	/var	Contains variable data files pertaining to the on-going system status such as log files of system activities.
11.	/home	Contains sub-directories of user accounts to store personal data files.
12.	/tmp	Contains temporary files.
13.	/root	This is the home directory of the root user.
14.	/opt	Contains software packages for installation. Packages are stored in sub-directories under the /opt.

Conclusion

- File system defines the ability of the computer system to store data.
- OS provides the interfaces required by application programs to manage data in files.
- File manager also provides mechanisms to organize files into folders called directories.
- Different methods of implementing files and directories result in differing performance.