

Secondary Storage

(Chp. 5.4—disk hardware, Chp. 6—File Systems, Tanenbaum)

Introduction

Secondary storage is the *non-volatile* repository for (both user and system) data and programs.

As (integral or separate) part of an operating system, the file system manages this information on secondary storage.

Uses of secondary storage include storing various forms of programs (source, object, executable) and temporary storage of virtual memory pages (paging device or swap space).

Information in secondary storage may be in a variety of forms, including readable text and raw data (e.g., binary).

File concept

A file is a *named* collection of related information, usually as a sequence of bytes, with two views:

- *Logical (programmer's) view*, as the users see it.
- *Physical (operating system) view*, as it actually resides on secondary storage.

What is the difference between a file and a data structure in memory? Basically,

- files are intended to be non-volatile; hence in principle, they are long lasting,
- files are intended to be moved around (i.e., copied from one place to another), accessed by different programs and users, and so on.

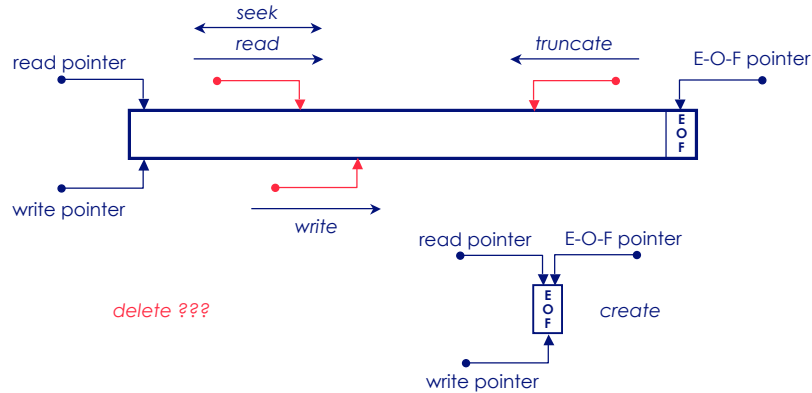
File attributes

Each file is associated with a collection of information, known as *attributes*:

- NAME, owner, creator
- type (e.g., source, data, binary)
- location (e.g., I-node or disk address)
- organization (e.g., sequential, indexed, random)
- access permissions
- time and date (creation, modification, and last accessed)
- size
- variety of other (e.g., maintenance) information.

File operations

There are six basic operations for file manipulation:
create, write, read, delete, reposition r/w pointer (a.k.a. seek), and truncate (not very common.)



Copyright © 1996-2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 4

File types

A common implementation technique (as organizational help with consistent usage) is to include the type as an extension to the file name:

<u>File type</u>	<u>Extension</u>	<u>Function</u>
Executable	exe, com, bin	ready-to-run code
Text	txt, doc	textual data, documents
Source	c, f77, asm	source in various languages
Object	obj, o	object code
Library	lib, a	library routines
Archive	tar, zip, arc	grouped files
Compressed	Z, gz	compressed
Print/view	ps, eps	printing or viewing
Word processor	ppt, wp, tex	various word processors

Files are structured internally to meet the expectations of the program(s) that manipulate them.

Copyright © 1996-2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 5

File access methods

The information stored in a file can be accessed in a variety of methods:

- *Sequential*: in order, one record after another.
- *Direct (random)*: in any order, skipping the previous records.
- *Keyed*: in any order, but with particular value(s); e.g., hash table or dictionary. *TLB lookup is one example of a keyed search.*

Other access methods, such as indexed, can be built on top of the above basic techniques. IBM's indexed sequential access method (ISAM) is built on random and sequential access.

Directories

A directory is a *symbol table*, which can be searched for information about the files. Also, it is the fundamental way of organizing files. Usually, a directory is itself a file.

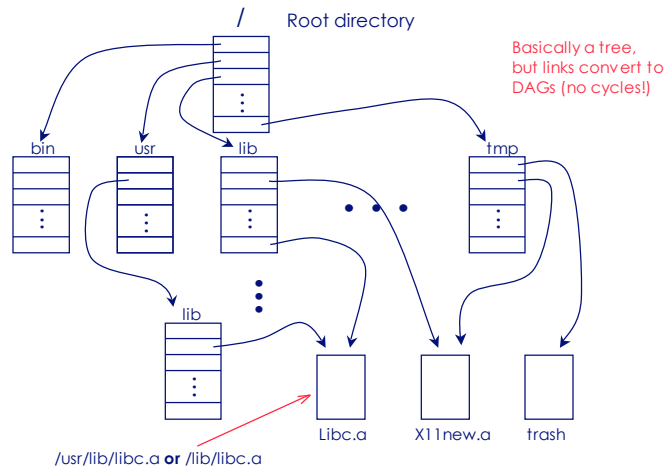
A typical *directory entry* contains information (attributes) about a file. Directory entries are added as files are created, and are removed when files are deleted.

Common directory structures are:

- **Single-level (flat)**: shared by *all* users.
- **Two-level**: one level for *each* user.
- **Tree**: arbitrary (sub)-tree for *each* user.

An example: UNIX directories

UNIX uses an advanced form of tree structure, known as *directed acyclic-graph (DAG)* directory.



Copyright © 1996-2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 8

File sharing

Allowing users to share files raises a major issue: *protection*.

A general approach is to provide *controlled access* to files through a set of operations such as read, write, delete, list, and append. Then permit users to perform one or more operations.

One popular protection mechanism is a condensed version of access list, where the system recognizes three classifications of users with each file and directory:

- user
- group
- other

Copyright © 1996-2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 9

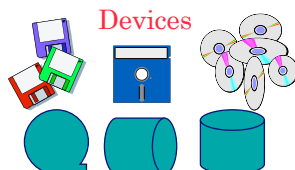
File systems

A file system provides a mapping between the logical and physical views of a file, through a set of services and an interface. Simply put, the file system hides all the device-specific aspects of file manipulation from users.

The basic services of a file system include:

- keeping track of files (knowing location),
- I/O support, especially the transmission mechanism to and from main memory,
- management of secondary storage,
- sharing of I/O devices,
- providing protection mechanisms for information held on the system.

File system abstraction

	<i>Objects</i>	<i>Typical operations</i>
Interactive (Shells)	files	copy, delete, rename
Applications and system programs	logical elements (records)	open/close, buffering seek (logical)
File System		file system check soft repair partitioning
Devices 	physical elements (head, cylinder, ...)	raw read/write, seek (physical) low-level format

Addressing levels

There are three basic mapping levels (abstractions) from a logical to physical view of a file (contents):

- **File relative:**
 <filename, offset> form is used at the higher levels, where the file system is viewed as a collection of files.
- **Volume (partition) relative:**
 device-independent part of a file system use <sector, offset> (e.g., a partition is viewed as an array of sectors.)
- **Drive relative:**
 at the lowest level, <cylinder, head, sector> (also known as <track, platter, sector>) is used.

File organization

One of the key elements of a file system is the way the files are organized. File organization is the “logical structuring” as well as the access method(s) of files.

Common file organization schemes are:

- Sequential
- Indexed-sequential
- Indexed
- Direct (or hashed)

File allocation

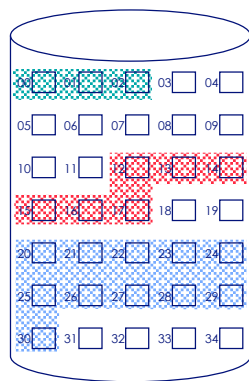
The file system allocates disk space, when a file is created. With many files residing on the same disk, the main problem is how to allocate space for them. File allocation scheme has impact on the efficient use of disk space and file access time.

Common file allocation techniques are:

- Contiguous
- Chained (linked)
- Indexed

All these techniques allocate disk space on a per block (smallest addressable disk unit) basis.

Contiguous allocation



Directory

name	start	len.
a.out	00	3
hw1.c	12	6
report.tex	20	11

Allocate disk space like paged, segmented memory. Keep a free list of unused disk space.

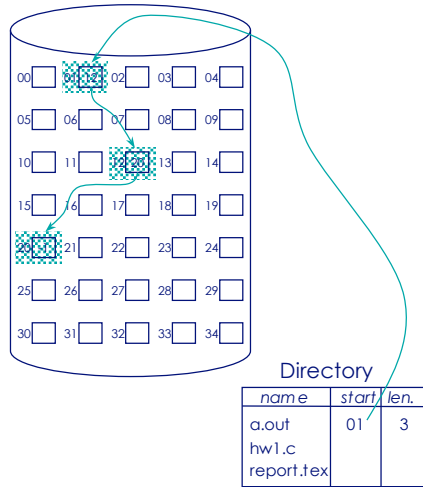
Advantages:

- easy access, both sequential and random
- simple
- few seeks

Disadvantages:

- external fragmentation
- may not know the file size in advance

Chained (linked) allocation



Space allocation is similar to page frame allocation. Mark allocated blocks as in-use.

Advantages:

- no external fragmentation
- files can grow easily

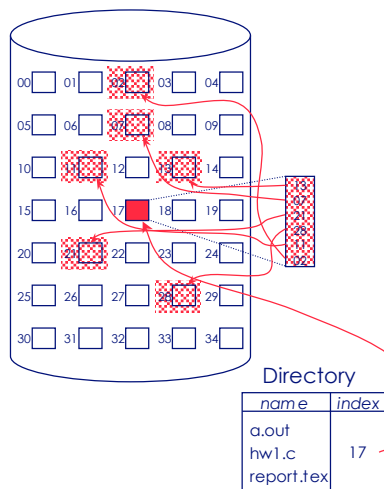
Disadvantages:

- lots of seeking
- random access difficult

Example:

MSDOS (FAT) file system

Indexed allocation



Allocate an array of pointers during file creation. Fill the array as new disk blocks are assigned.

Advantages:

- small internal fragmentation
- easy sequential and random access

Disadvantages:

- lots of seeks if the file is big
- maximum file size is limited to the size of a block

Example:

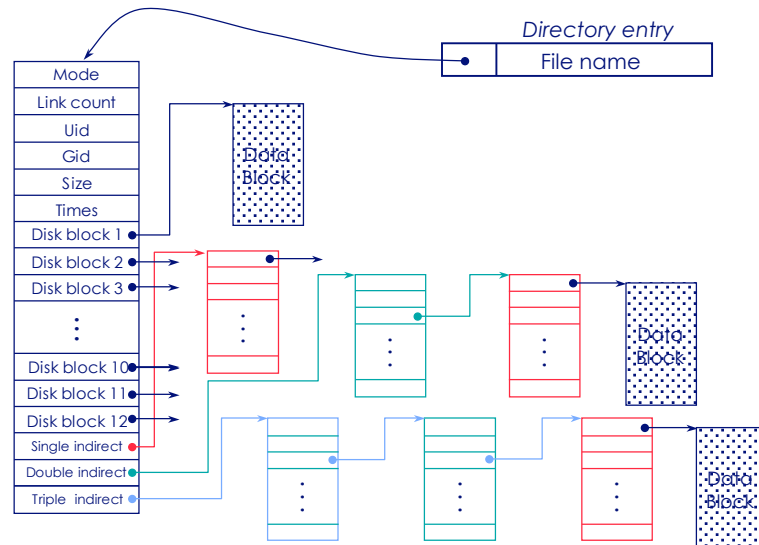
UNIX file system

Free space management

Since the amount of disk space is limited (posing a management problem similar to that of physical memory), it is necessary to reuse the space released by deleted files. In general, file systems keep a list of *free* disk blocks (initially, all the blocks are free) and manage this list by one of the following techniques:

- Bit vectors
- Linked lists or chains
 - single list of a set of free block lists
- Indexing
 - single level, multiple levels

An example: UNIX I-nodes

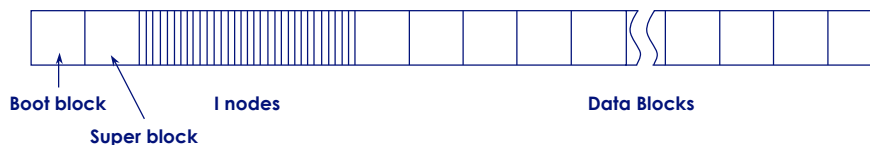


Other file system issues

- *Disk blocking*
 - multiple sectors per block for efficiency
- *Disk quotas*
- *Reliability*
 - Backup/restore (disaster scenarios)
 - File system (consistency) check (e.g., UNIX **fsck**)
- *Performance*
 - Block or buffer caches (a collection of blocks kept in memory)

Case study—UNIX file system

Disk (partition) layout in traditional UNIX systems



The boot block usually contains (bootstrap) code to boot the system.

The super block contains critical information about the layout of the file system, such as number of I-nodes and the number of disk blocks.

Each I-node entry contains the file attributes, except the name. The first I-node points to the block containing the root directory of the file system.

Case study—UNIX file system *cont.*

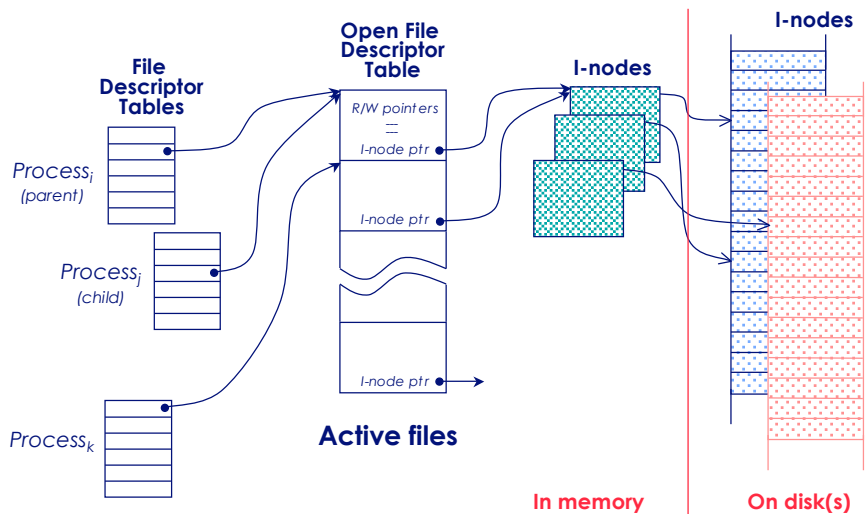
There are three different indirection to access a file:

- *File Descriptor Table*: one per process, keeping track of open files.
- *Open File Table*: one per system, keeping track of all the files currently open.
- *I-node Table*: one per system (disk volume or partition) keeping track all files.

Directories are stored just like ordinary files. User programs can read directories, but special care is needed to write a directory.

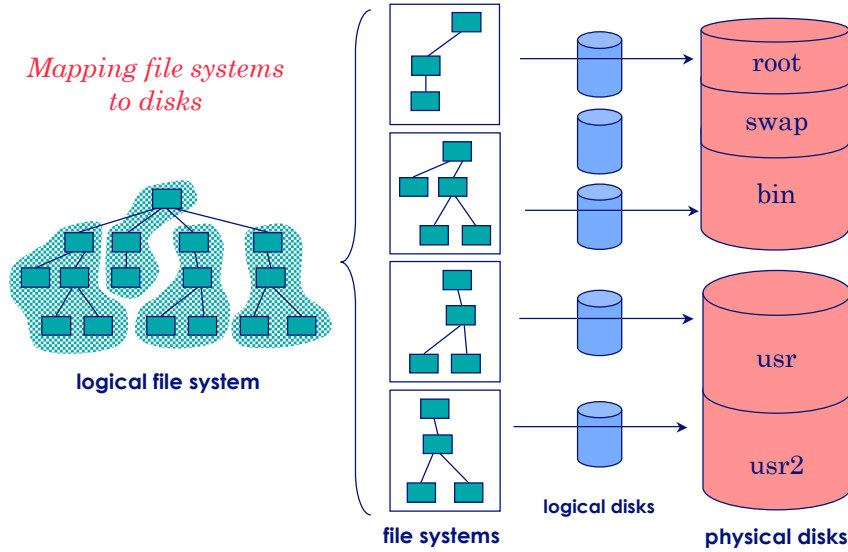
Each directory contains $\langle \text{file name, I-node number} \rangle$ pairs. *Root* (i.e., /) is a special directory with no name.

Case study—UNIX file system *cont.*



Case study—UNIX file system *cont.*

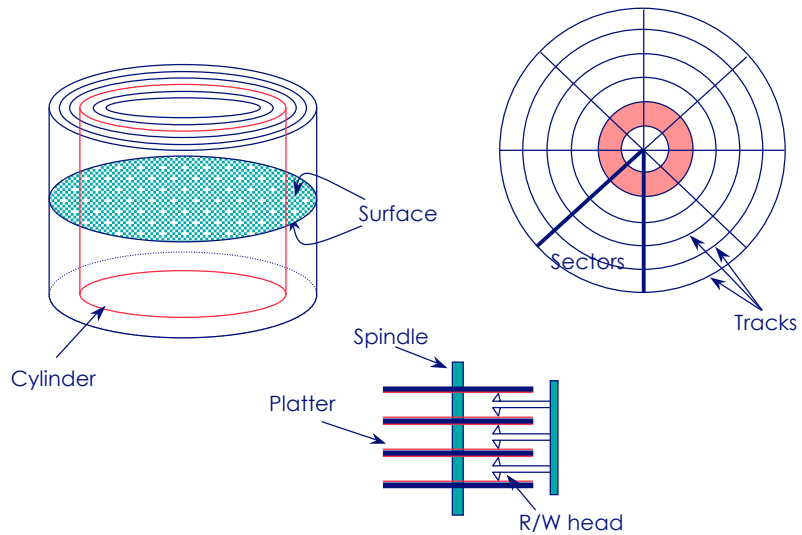
Mapping file systems to disks



Copyright © 1996–2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 24

Disk structure revisited

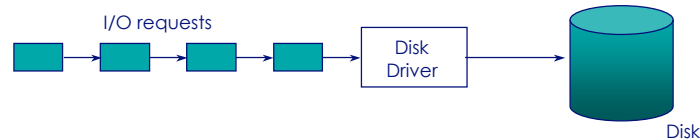


Copyright © 1996–2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 25

Disk scheduling

In multiprogramming systems, there may be several disk I/O requests at the same time. As a result, a disk driver is typically faced with a pool of I/O requests:



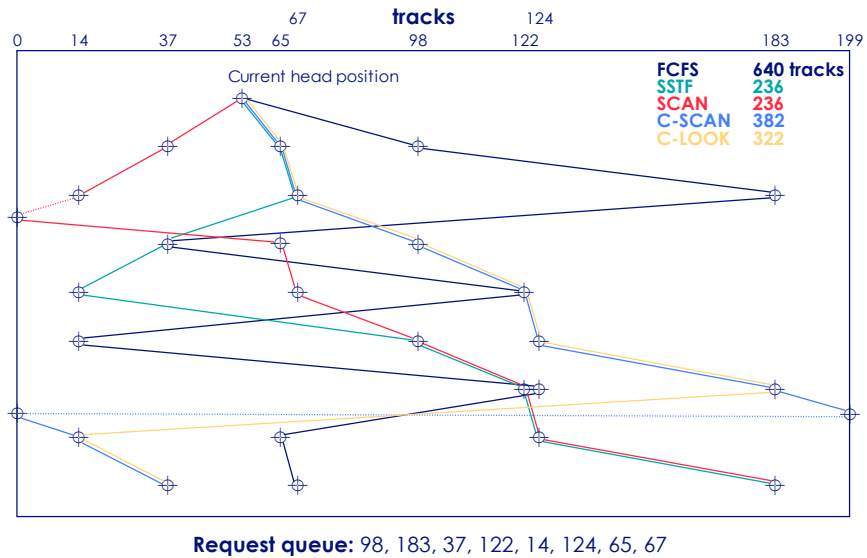
The most costly component of a disk I/O operation is the *seek time*. By scheduling multiple disk requests, the total seek time can be reduced. For example, shortest seek time first.

Disk scheduling strategies

Commonly used strategies include (in addition to some common CPU scheduling policies!):

- *First Come First Served (FCFS)* or FIFO
- *Shortest Service Time First (SSTF)*
- *SCAN*—back and forth over disk
- *C-SCAN*—circular SCAN or one way SCAN and fast return
- *LOOK*—look for a request before moving in that direction
- *C-LOOK*—circular LOOK

A comparative example



Copyright © 1996-2005 Eskioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 28

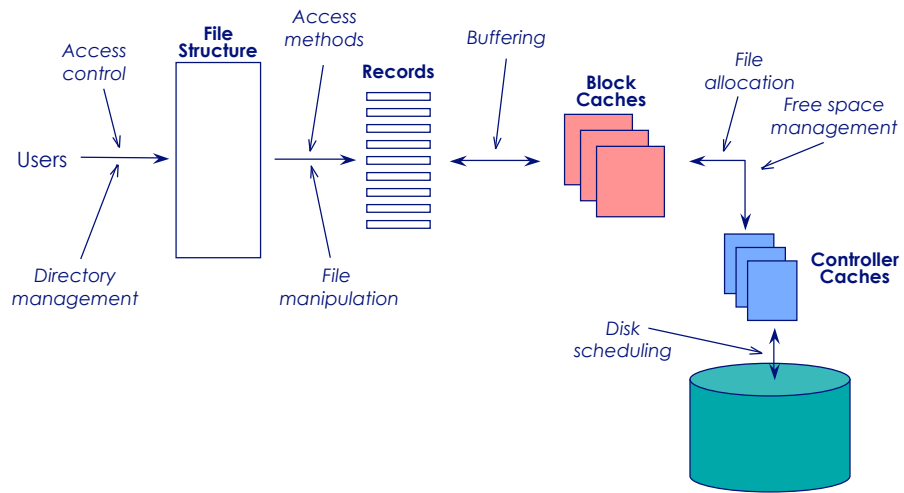
Disk management issues

- *Formatting*
 - *Physical*: divide the blank slate into sectors identified by headers containing such information as sector number; sector interleaving
 - *Logical*: marking bad blocks; partitioning (optional) and writing a blank directory on disk; installing file allocation tables, and other relevant information (file system initialization)
- *Reliability*
 - disk interleaving or striping
 - RAIDs (Redundant Array of Inexpensive Disks): various levels, e.g., level 0 is disk striping)
- *Controller caches*
 - newer disks have on-disk caches (128KB—512KB)

Copyright © 1996-2005 Eskioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 29

Elements of storage management



Copyright © 1996-2005 Eskicioglu and Masland (and Prentice-Hall and Paul Lu)

Secondary Storage 30