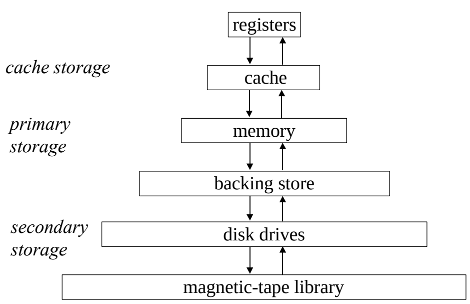
Silberschatz, et al.
Topics based on Chapter 11

File-Systems

File System Interface

- Physical storage mechanisms (see earlier chapters also)
- Files--logical storage unit (abstract) that is independent of actual storage device
- Directory structure--organizing the collection of files
- Partitions--an additional division sometimes found
- File protection
- Implementation
  - Organization
  - Allocation
  - Free-space management
  - Directory implementation

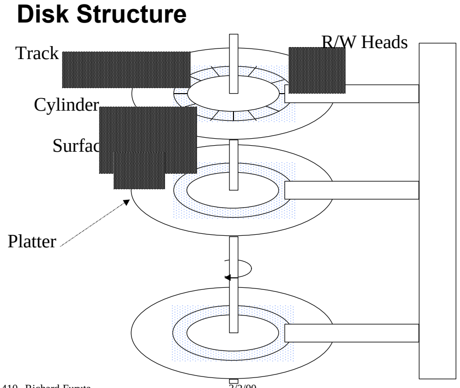
Storage Hierarchy



Primary and Secondary Storage

- Primary storage: small, volatile
- Secondary storage: large, <u>nonvolatile</u>. Able to hold very large amounts of data permanently. Example: magnetic disks, magnetic tapes, optical disks.
- Magnetic disks:

Disk Structure



Disk Terminology

- <u>Cylinder</u>: all <u>tracks</u> that can be accessed without moving the read/write head
- Tracks divided into <u>blocks</u>
- Fixed-size blocks define a <u>sector</u>
- Sector: smallest unit of information that can be transfered to/from disk

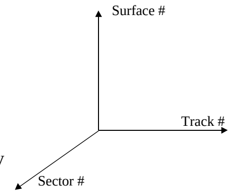
Disk Access Time

- seek time: time to position heads on cylinder (a fixed head disk does not require seek time but is more expensive than a moving-head disk)
- rotational latency: delay in accessing material once seek accomplished (time required to wait for data to rotate around under head)
- Transmission time: time to transfer information once it is under the head.
- access time = seek time + rotational latency
  +read/write transmission time
  seek time >> read/write time

<u>Disk Addressing</u>

- Disk address = (disk driver id,
  surface number,
  track number,
  sector number)

A disk is a three-dimensional array of sectors



Block address b, given cylinder i, surface j, sector k

b = k + s * (j + i * t)

s, the number of sectors per track
t, the number of tracks per cylinder (e.g., number of surfaces)

Accessing sequential disk addresses

- Question: what is the cost of accessing block b+1 given b?

Issues of Disk Management

- Allocation method
  - mapping: files ==> disk sectors/blocks
- Free space management
  - data structure and access method
- Disk head scan methods
  - implementation cost, run-time overhead, fairness

Magnetic Tape: Comparison

- 9 tracks on 1/2 inch wide tape (one bit per track)
- Variable length records (about 20-30000 bytes)
- Density in implementation varys from about 200 to 6250 bytes per inch (bpi) with higher densities more common now
- Tape speed: 20 to 200 inches per second. Takes time to come to speed and to stop. This can be megabytes/second when at speed, so DMA transfer may be required
- Inter-record gap is about .6 inch
- Can read or write at end but can only read in middle of tape (because of alignment of subsequent records)

Magnetic Tape

- Tape drives are not fully random-access devices.
- Disk can read/rewrite single sectors in middle of disk, can seek to locations relatively directly, etc.

File concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
    - source
    - object (load image)
  - Documents

File Systems

- A file is a named collection of related information that is recorded on secondary storage
- File is a *logical* storage unit: independent of actual storage device; mapped by OS onto physical devices
- Generally *persistent* (e.g., across power failures); *nonvolatile*
- Referred to by *name* (for convenience of human users)
- May have *types* (e.g., source, data, object, executable)
- A file is an abstract data object with specific attributes and operations provided by the system

File attributes

- Name: symbolic name; the only information kept in human-readable form
- Type: if supported by system (more later)
- Location: pointer to device & location of the file on device
- Size: current size and perhaps the maximum allowed size
- Protection: access-control information (e.g., reading, writing, executing, etc.)
- Time, date, and user identification: e.g., creation, last modification, last use
- Usage count, owner, etc.

File attributes

- Kept in the directory structure
  - Also resides on secondary storage
  - 16 to 1000 bytes to store file attribute information
  - Directories may themselves be very large

File operations

- Create: allocate space, enter into directory
- Write: given name and information to be written (system keeps *write* pointer to file)
- Read: given file name and destination of information (system keeps *read* pointer)
- Reposition within a file (also known as file *seek*)
- Delete: release space and erase from directory
- Truncate: keeps directory entry/attributes but deletes contents (resets length to 0)
- Open/close file

File operations Open/close files

- open/close file operations add/delete entry to open-file table. File accesses via open-file table
- Information associated with open file
  - file pointer (if no offset in read/write)
  - file open count (to keep from removing entry from open-file table prematurely)
  - disk location of the file

File Type

- By extension (.exe, .com, .bat, ...)
- By type attribute (e.g., Macintosh with type/creator attributes---creator identifies application to be launched).
- By “magic number” (as in Unix; embedded into file, at start)

Common file types

File type	Usual Extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	compiled machine language, not linked
Source code	c, p, as, f77, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data, documents

Common file types

File type	Usual Extension	Function
Word processor	wp, tex, rrf, ...	various word processor formats
Library	lib, a	libraries of routines
Print or view	ps, dvi, gif	ASCII or binary file
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed

File structure

- Potential structures
  - None - sequence of words, bytes
  - Simple record structure
    - Lines
    - Fixed length
    - Variable length
  - Complex Structures
    - Formatted document
    - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters. Operating system or program can establish file structure.

File Structure

- File type may indicate internal structure of file (e.g., source or object)
- IBM mainframe systems, for example, support a very wide range of access methods (see later)
- UNIX, MS-DOS, others, support only a minimal number of file structures. (UNIX files are sequence of 8-bit bytes)
- Macintosh resource fork and data fork
- Logical record size and physical block size (blocking factor)--packing a number of logical records into physical blocks. Block allocation results in internal fragmentation, in any case

File Access Methods

- Sequential Access (as in magnetic tape)
- Direct Access (or relative access). Logical records can be read/written in no particular order. read/write must include a block number as parameter
- Other access methods

Sequential access

read next
write next
reset or *skip n*
no *read* after last *write* (*rewrite*)

Direct access

read n
write n
position to n
  read next
  write next
rewrite n

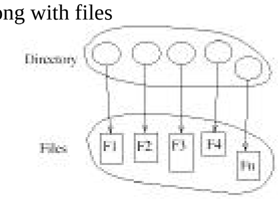
n = relative block number

Other access methods

- Indexed methods, e.g., ISAM (indexed sequential access method, which has file sorted on a defined key. Access look in master index for block number of secondary index. Secondary index read then binary searched for block with desired record. This block is then searched sequentially.)

Directory structure

- A collection of nodes containing information about all files
- Resides on disk, along with files



Information in a device directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

Directory Operations

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system (e.g., for backup purposes)

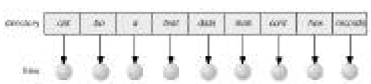
Goals in (logical) directory organization

- Efficiency--enable locating a file quickly
- Naming--convenient to users
  - Two users (or two directories) can have the same name for different files
  - The same file can have several different names
- Grouping--logical grouping of files by properties (e.g., extension, date changed, etc.)

Logical Directory Structures

- Single-level Directory
- Two-level Directory
- Tree-structured Directory
- Acyclic-graph Directory
- General-graph Directory

Single level directory



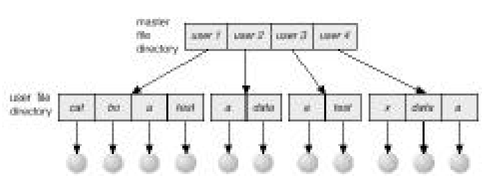
Single-Level Directory

- Advantage: simple to support and understand
- Disadvantage: files must have unique names (multiple users may clash; names may be limited in length; large directories may be hard to remember)

Two-Level Directory

- Each user has own user file directory (UFD)
- Master file directory (MFD) holds pointers to UFDs
- Disadvantage: discourages cooperation

Two-Level Directory

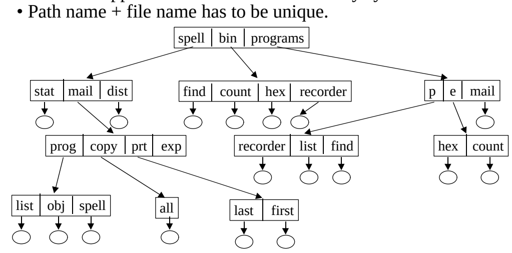


Tree-Structured Directories

- Natural generalization of two-level directories
- As in MS-DOS
- current directory, change directory operation
- policy decision: how to handle requests to delete a (non-empty) directory (prohibit, recurse, ...)

File System Organizations

- Objectives: Providing facility for locating, accessing, and protecting files in the system.
- Common approach: tree structured directory system
- Path name + file name has to be unique.

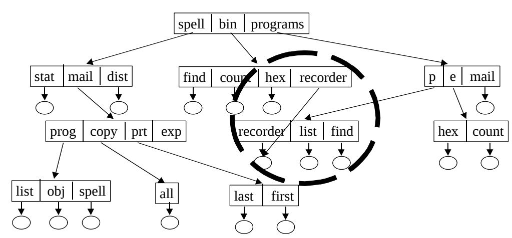


Acyclic-Graph Directories

- Add to tree-structured directories, for example, Unix **ln**
- permits the sharing of files and subdirectories
- the *same* file/subdirectory exists in the file system in two or more places at the same time

File System Organizations

recorder is now found in two locations in the file system



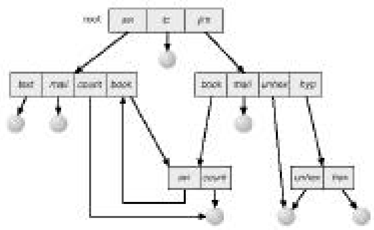
Acyclic-graph directory structures

- File now has multiple absolute path names (aliasing problem). Can backup avoid copying the same file twice?
- Deletion: when can the space be reclaimed? Reference counts, for example (or dangling links as another possible solution)
- Who gets charged for file space?
- Ensuring the absence of cycles
  - Allow links to file, not subdirectories (Unix hard links)

General Graph Directory

- As in acyclic-graph structure but also permits cycles!
- Traversal becomes more complicated (avoid traversing same entries repeatedly). How about ls -R?
- Self-referencing files create difficulties with reference counts. Garbage collection may be required. (Pass one: traverse and mark; pass two: collect).

General graph directory



Protection Mechanisms

- Simple protection by access right lists
  - read
  - write
  - execute
  - append
  - delete
  - list

Access lists and groups

- Mode of access: read, write, execute
- Classes of users
  - Owner
  - Group (membership carefully controlled)
  - Public
- Unix protection bits
  - RWXRWXRWX
  - O G P
- Operations: **chmod**, **chgrp**

Access lists and groups

- General access lists are even more flexible than Unix scheme
  - Example: allow everyone but one person to read a file
- Unix: interpretation of “x” bit for directories
- Unix: suid bit

File System Organization

- Secondary storage: disks
  - i/o transfers performed in units of *blocks* (one or more sectors)
  - blocks vary between 32 bytes and 4096 bytes. Generally 512 bytes
- *File system* used to provide structure for the information stored on a disk. Provides for efficient and convenient access
  - how should the file system look to the user (file, attributes, operations, directory structure)
  - how should the file system be mapped onto physical secondary-storage devices (algorithms, data structures)

File System Organization

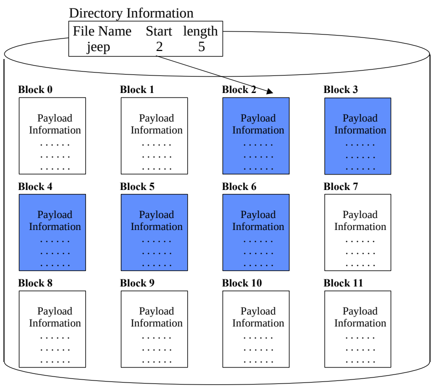
application programs
(open(), file descriptor, uses open-file table)
logical file system
(symbolic file name)
file-organization module
(files, logical blocks, physical blocks)
basic file system
(generic read/write physical block cmds)
I/O control
(device drivers, interrupt handlers)
devices

File System Allocation Methods

- How are blocks associated with a file stored on disk?
  - contiguous
  - linked
  - indexed

Contiguous Allocation

- Each file occupies a set of contiguous addresses on disk
  - a file n blocks long occupies addresses b through $b+n-1$
  - Sequential access is easy (just remember address of last block accessed and get the next)
  - Direct access also easy (access $b+i$ directly)
- Issue: how to find space for new file (instance of the general *dynamic storage-allocation* problem discussed earlier)



Contiguous Allocation

- Finding a section of contiguous free blocks
  - first fit
  - best fit
  - worst fit

Contiguous Allocation

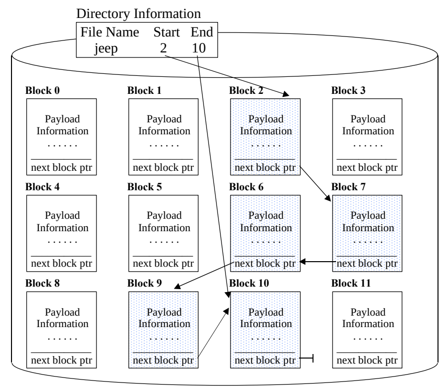
- Problems
  - external fragmentation (e.g., requiring re-packing to reduce external fragmentation--requires down time)
  - determination of needed file size at creation time
    - modification of file implies changing the size. how to handle expansion if no free space adjacent? terminate? relocate? expensive...
    - overestimation of size causes internal fragmentation if extra space left (perhaps for lifetime of file--years)

Contiguous Allocation

- A modification to contiguous allocation also incorporates an extent
- if additional space is needed, the extent is added to the initial allocation.
- Directory contains means to include pointer to the first block of the extent (in addition to the location of the initial allocation and its size)
- Can be generalized to permit multiple extents... See grouping in later discussion of directory implementation

Linked Allocation

- File: a linked list of disk blocks
- Directory: contains pointer to first and last blocks in file
- Initially the directory entry is **nil**
- Requires space for links. If physical block is *n* bytes and disk address takes *x* bytes, then available space in block is *n - x*



Linked Allocation

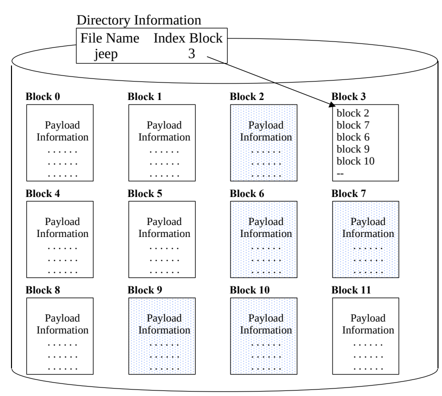
- Advantages:
  - no external fragmentation
  - no compaction needed
  - easy to modify file (insert/delete blocks)
- Disadvantages
  - access method: sequential access only
  - reliability: recovery difficult if pointers lost
  - disk space efficiency: needed space for pointers is "wasted"
    - one option: clusters (collect blocks into multiples and allocate cluster not block). Expense: internal fragmentation

Linked Allocation

- Variation: File Allocation Table (FAT)
  - Section of disk set aside at beginning of partition containing one entry for each disk block
  - Links represented by storing next address in slot in FAT
  - Must cache to keep from having to do two head seeks for each read...
  - Eases implementing random access because location of block can be determined by processing FAT (not traversing disk). But you still have to process the FAT.

Indexed Allocation

- *Index block* brings together file's pointers to disk blocks into one location
- Each file has its own index block
- Index block contains disk addresses for blocks allocated to the file
- Initially all are **nil**
- As blocks are allocated, they are added to the index block



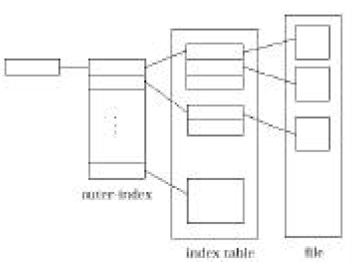
Indexed Allocation

- Advantages
  - no external fragmentation. no compaction required
  - access method: both sequential and random access
  - reliability: lost pointers have limited effect, (not global)
  - easy file modification (only have to rearrange block)
- Problems
  - pointers require disk space. Wasted space in index block for small file (unused indices)
  - size of index block limits size of file unless we develop a modified scheme

Indexed Allocation

- Incorporating multiple index blocks
  - linked indexed allocation
    - Last word in block is either **nil** or a pointer to another index block
  - multi-level indexed allocation
    - two level: index block that points to index blocks. With 2048 byte blocks and 4-byte addresses can get 512 pointers. Two levels is 4,194,304 data blocks or 8.5 gigabytes.
    - multi level: further levels of indirection

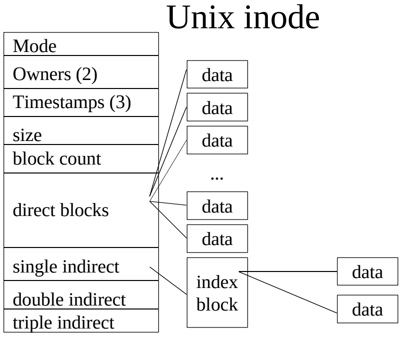
Indexed allocation



Indexed Allocation

- Combined scheme (BSD UNIX)
  - first 15 pointers of index block kept in file's index block (or inode). First 12 of those point to direct blocks (contain data). Next 3 point to indirect blocks. First is a single indirect block (points to index block which points to data blocks). Second is a double indirect block (points to index block which points to index block). Third is triple indirect blocks.

Unix inode



Unix inode

- Small file accessed directly from inode
- Larger files require additional indirections
- Total addressable blocks exceed that addressable by the 4-byte file pointers used by the OS.
- Indexed blocks can be cached in memory but data blocks may be spread all over a partition (requiring seeks for sequential access)

Free-Space Management

- How do you locate unused disk blocks?
- *Free-space list* records all disk blocks that are not allocated to a file or directory (i.e., *free*)

Free-Space List

- Bit Vector: each block represented by a bit which is 1 if free, 0 if allocated
- Simple and efficient to find first free block or *n* consecutive free blocks, often with processor bit-manipulation instructions
- But a large disk requires a large bit vector. For example 1.3 gigabyte with 512K blocks requires 310K of bit vector. (Clustering of blocks helps.)

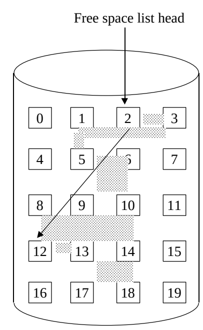
Free-Space List
Linked List

- Link together all free disk blocks keeping a pointer to the first free block in a special location (cached in memory). First block contains pointer to next free block, etc...

<u>Grouping</u>

- Linking all the free blocks
- Using some space on disk.
- Taking less time than linked list.
- Storing n-1 addresses in the first free block.
- The n-th address is used to point to the next "address block"

Block 2:	Block 12:
3	13
5	18
10	-
12	-



n = the size of block = 4

Free-Space List
Grouping

- Store addresses of *n* free blocks in the first free block. The first *n-1* are allocatable. The *n*th contains another list of addresses
- Addresses of a large number of free blocks can be found quickly (when compared to linked lists)

Free-Space List
Counting

- Keep address of first free block and the number, *n*, of contiguous blocks that follow the first
- Entry in free-space list is disk address and a count
- Each entry requires more space but there are fewer of them
- Assumption is that contiguous blocks are allocated or freed simultaneously

Counting

• Linking all the free blocks

• Using some space on disk.

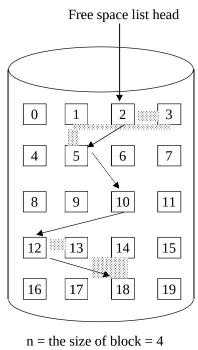
• In each item of the list, store the number of contiguous free blocks from the current one.

Block 2: 2 - - 5

Block 5: 1 - - 10

Block 10: 1 - - 12

Block 12: 2 - - 18



Directory Implementation

- Linear
  - new file: search for name then if not found add to end
  - delete file: find and release space. Either mark entry “unused” or replace with a valid entry (for example, the last one in the list)
  - search time can be a factor when directory gets large (search times in UNIX directories, for example). Cache, ordered lists, etc., can help.

Directory Implementation

- Hash table
  - both a linear list and also a hash table
  - as with any hash table have to deal with *collisions*

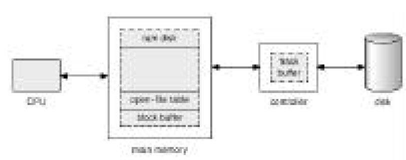
Design Issues in Disk Storage Management

- Block size
- Free space management
- Allocation methods
  - fragmentation, internal and external
  - access methods
  - reliability
  - space efficiency
  - run-time overhead
  - limits on file size
  - file modification limits

Efficiency and performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file’s directory entry
- Performance
  - *disk cache* – separate section of main memory for frequently used blocks
  - *free-behind* and *read-ahead* – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as *virtual disk*, or *RAM disk*

Various disk caching locations



Recovery

- Consistency checker – structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.