File System

Sanzheng Qiao

Department of Computing and Software

January, 2013

Introduction

Files as abstract data types provide a way to store information and hide the details of how they work.

Components of a file system:

files
directory structure
possible partitions

Internal file structure: logical record. In UNIX, the record size is one byte. A file is a stream of bytes.

Disk storage unit: sector (block), 32 bytes to 4K bytes, usually 512 bytes.
File system maps between logical records and physical sectors, packing logical records into physical sectors.

Major issues

File management

- Files structures
- Access methods
- Directory structures
- Mounting file systems
- Protection

Disk management

- Allocation methods
- Free sector management
- Efficiency

File structures

1. Different structures for different files:
   .TXT .PAS .BIN .DAT
   More support from system (opening a file by double
   clicking on the icon launches the creator automatically),
   less flexible. (Cannot copy a _.DAT_ file to a _.PAS_ file.)

2. One structure for all files:
   The logical record size is one byte. Some files have magic
   numbers at the beginiing to indicate the file type. Less
   support from system, more flexible.

Operations on files

create
: Create a file (an entry in the working directory) with no data. May specify some attributes (owner, time, etc)

open
: A file must be opened before using. For user, it returns an integer (file descriptor) to be used to read and write; in system, it returns a pointer to the file header (i-node). The file descriptor is an index to the per-process open file table. File system maintains a global open file table. A file may be opened by multiple processes.

close (unlink)
: Finish using the file.

delete
: Remove the file from the directory and free the disk space.

Operations on individual files

read Read from the file given by file descriptor (user) or file header (system), starting from the current position (a private variable). The current position is updated after read.

write Similar to read. Write may require read. When writing partial sector, the sector must be read into memory. (Remember: disk unit is sector.)
Two images of a sector: Memory and disk.

seek Move the current position.

Access methods

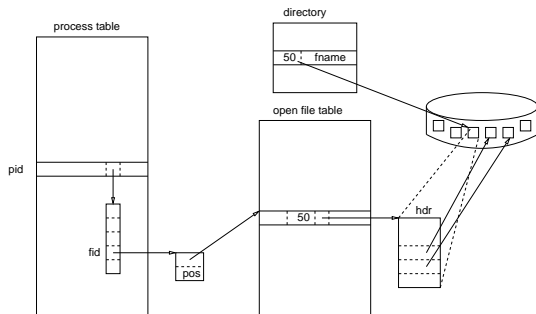Sequential access Always start from the beginning.

Direct (random) access Can go directly to the sector containing
the byte.
In modern operating systems, all files are random
access.
Device-independence
Making access the same no matter where (which
disk) the file is stored.

Direct access

In UNIX a directory is a file with special data structure, a table of entries (file name, sector number of the file header). Files and directories are represented by entries in a directory.

In UNIX a directory is a file with special data structure, a table of entries (file name, sector number of the file header). Files and directories are represented by entries in a directory.

Directory Structures

Single level: No file name sharing.
Two level: Users are isolated.
Tree structure: Search by complete path. A file is specified by its path name (absolute or relative). Path of the working directory (pwd).

Graph structure: Files can be linked across directories. Hard link, `ln`, keep track the reference count; Symbolic link, `ln -s`, keep the path name in a link file.
In this structure, users can share files, however, a file may have multiple absolute path names. The following problems should be considered:

- When traversing file system to collect statistics, a file may be visited multiple times
- When deleting files, some processes may have dangling pointers
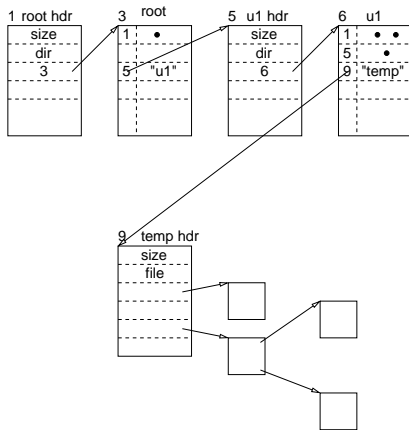- When backing up files, a file may be copied multiple times

How does the system find the file (file header) given the path name (absolute or relative) by user?

1. Find the entry in the directory using the file name;
2. Get the sector number of the file header from the entry;
3. Read the file header from disk into memory.

Example

Finding /u1/temp

Sharing files

Hard link

```
% ln file copy
```

Two files share the same inode number.

```
% ls -i
  105852      2     file
  105852      2     copy
```

Soft link

```
% ln -s file copy
```

Two files have different inode numbers and `copy` contains the pathname of `file`.

Protection

Who is allowed to do what.
Mechanisms

- An access-control list (ACL) associated with every file and directory.
  Condensed version: Three classes, owner, group, world.
- A password for every file and directory.

A user may have different access rights to the same file in graph structured directory system.

Allocation methods

Contiguous Store a file in contiguous sectors on the disk. All we have to know is the disk sector number of the first sector of the file. Easy access, few seeks, horrible external fragmentation.

Linked list Sequentially follow the link to locate the sector. Flexible on size, no fragmentation problem, sequential access is easy, direct access is hard, lots of seeking.

Indexed Use the sector number (logical) in the file as an index to find the disk sector number (physical). Both sequential and direct access are easy, lots of seeking (sectors are scattered).

Allocation methods

Contiguous Store a file in contiguous sectors on the disk. All we have to know is the disk sector number of the first sector of the file. Easy access, few seeks, horrible external fragmentation.

Linked list Sequentially follow the link to locate the sector. Flexible on size, no fragmentation problem, sequential access is easy, direct access is hard, lots of seeking.

Indexed Use the sector number (logical) in the file as an index to find the disk sector number (physical). Both sequential and direct access are easy, lots of seeking (sectors are scattered).

Example (4.3 BSD):
Multi-level indexed files (direct data blocks, indirect data blocks, doubly indirect)

file header (i–node)

direct

single indirect
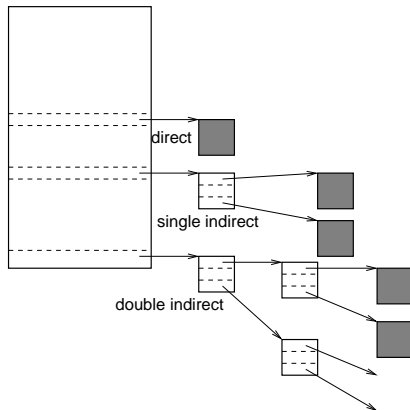
double indirect

- Bit map. Usually we can keep entire bit map in memory most of the time.
- Try to allocate contiguous blocks. Reduce seek time.
- Problem when disk becomes full. Solution: keep a reserve (e.g. 10% of disk) space.

Observations:

- Most files are small.
- Much of the disk is allocated to large files.
- Many of the I/O operations are made to large files.

Observations:

- Most files are small.
- Much of the disk is allocated to large files.
- Many of the I/O operations are made to large files.

Consequence:
per-file cost must be low (lot of them), but large files must have good performance (they take much of the disk).

Ordinary files A files is a linear array of bytes, sequential access (pointer).

Directories A directory is like a symbol table consisting of entries with names and i-numbers which are pointers pointing to inodes on the device (disk).

Special files They are in /dev (information about tape drivers, disks, terminals, etc). Character special files (terminals). Block special files (disks).

They have different i-node structures.

Example: UNIX file system

Old system (150 MB)

The disk contains a super block followed by i-nodes (4MB) and then data blocks (146MB). Block size 512B.

The super block contains basic information of the file system, such as the number of data blocks, a count for maximum number of files, and a pointer to the free list.

Each inode contains type, number of links, owner's user and group id, permissions, size, time of last access, last modification, pointers to disk blocks (direct and indirect).

Never transfer more than 512 bytes per disk transfer.

Example: UNIX file system

Problems

- Segregation of inodes and data: long seek time for accessing a file (from its inode to data);
- Files in the same directory usually are accessed consecutively, but their inodes are not located consecutively;
- Disk transfers are in 512-byte (small) blocks. Consecutive logical blocks are often not allocated in consecutive physical blocks;
- Even with large block size (1024 bytes), files tend to have their blocks allocated randomly over the disk causing long seek time.

Example: UNIX file system

New system (4.2 BSD)

Old UNIX file system is inadaquate for the applications which require high throughput, i.e., small amount of processing on large quantities of data.

Main goals

- increase throughput
- improve user interface

New File System Organization

- Superblock is replicated for protection.
- Block size can be any power of 2 greater than or equal to 4096 bytes. Large block size ensures only two levels of indirection. The block size is determined when the file system is created.
- A disk is partitioned into cylinder groups each of which contains a copy of the superblock, bit map replacing the free list. A static number of inodes is allocated for each cylinder group. One inode for each 2048 bytes of space in the cylinder group.

Example: UNIX file system

Improving storage utilization

- Large block size and cylinder groups reduce seek time and improve throughput, but large block size wastes space.
- A block is divided into fragments (2, 4, or 8) determined when the file system is created. The lower bound is the disk sector size, typically 512 bytes.
- Fig. 1 shows a bit map. Consecutive fragments must be allocated in the same block.

- When a file expands, during a write, three things can happen:
    1. If enough space left in an allocated block or fragment, then write in that block or fragment;
    2. Fill the allocated block first, then get new blocks if necessary; (this may result in scattering fragments)
    3. If scattered fragments exceed a full block, collect fragments into a block (fragment reallocation)
- To reduce the cost of reallocation, the user program should write a full block whenever it is possible.

File System Parameterization

Use parameters, such as the speed of processor, characteristics of the mass storage devices, to optimize the allocation of blocks.

Characteristics of the processor: Expected time to service an interrupt and schedule a new disk transfer (run disk interrupt handler).

Characteristics of the mass storage devices (disks): Number of blocks per track; rate at which the disk spins.

Layout Policies

- Two methods: increase the locality of reference to minimize seek latency; improve data layout to make large transfers possible.
- Top level (global) policies try to balance the two conflicting goals of localizing data while spreading out unrelated data.
- Lower level (local) policies try to place all data blocks for a file in the same cylinder group, preferably at rotationally optimal positions in the same cylinder.

Performance of the New File System

- transfer rates for the new file system do not appear to change over time.
- fast bandwidth is due to large block size.
- read is at least as fast as write. Blocks are more optimally ordered on the disk

Functional Enhancements

Long File Names

- directory size: 512-bytes = disk sector size
- entry data structure:
    - inode number (fixed size),
    - size of the entry (fixed size),
    - length of the file name (fixed size)
    - file name (variable length)
- when an entry is deleted, it is combined with its previous entry and increase the entry size if possible

Symbolic Links

A symbolic link is implemented by a file containing a pathname.
When the system encounters a symbolic link,

- if the pathname in the file is absolute, use it;
- if the pathname in the file is relative, it is added to the current path;

Reliability

Bad sectors:

- Hardware solution: Dedicate a block to bad sector list. During initialization map bad sectors to spare ones.
- Software solution: Make a file containing bad sectors and remove them from the free list or bit map.

Backup: Monthly full dumps and daily incremental dumps.

Example: UNIX file system

Consistency

Check consistency after a crash.

1. Count the number of times a sector is used by a file.
2. Compare with the bit map.
   - If both are zero (missing sector), put it on free list.
   - If both are one, remove it from free list.
   - If it is used more than once, make a copy.