Implementing File System

Daniel Bosk[1]

Department of Information and Communication Systems (ICS),
Mid Sweden University, Sundsvall.

fs.tex 280 2018-12-13 08:52:07Z jimahl

Overview

1. File Systems
    - File-System Structure
    - Volumes
    - Kernel Data-Structures
    - Operations on Files

2. Virtual File Systems
    - File System Layers

3. Block Allocation and Free Block Algorithms
    - Directories
    - Contiguous Allocation
    - Linked Allocation
    - Indexed Allocation

Mittuniversitetet
MID SWEDEN UNIVERSITY

Literature

The lectures gives an overview of Chapter 11 “Implementing File-Systems” in [SGG13a].

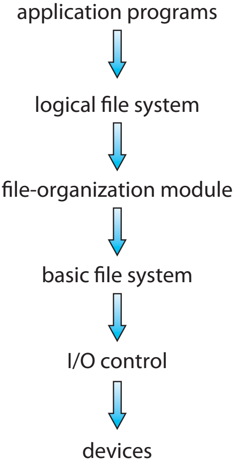
Overview

1. File Systems
    - File-System Structure
    - Volumes
    - Kernel Data-Structures
    - Operations on Files
2. Virtual File Systems
    - File System Layers
3. Block Allocation and Free Block Algorithms
    - Directories
    - Contiguous Allocation
    - Linked Allocation
    - Indexed Allocation

File-System Structure

- The disk stores data in blocks, normally 512 to 4096 bytes.
- FS provides an easy and convenient way of accessing this data.
- The first problem is how the FS should look to the user.
- The second problem is creating algorithms and data structures to map the logical file system (what the user sees) onto the physical secondary storage devices.
- We will now focus on the second problem.

File-System Structure



application programs

logical file system

file-organization module

basic file system

I/O control

devices

Figure: Layered hierarchy of file system structure. Image: [SGG13b].

File-System Structure

I/O Control

- This layer consists of device drivers and interrupt handlers.
- This layer transfers information between memory and disk.
- From upper layers: fetch block 123.
- To lower layers: hardware specific instructions.
- The device driver writes specific bit patterns to special locations in the I/O controller's memory.

File-System Structure

Basic File System

- The basic file system issues generic commands to the appropriate device driver; read from block X, write to block Y.
- Each physcial block is identified by its numeric disk address; e.g. drive 1, cylinder 73, track 2, sector 10.
- This layer also manages buffers and caches that hold various directory and data blocks.
- The manager allocates a block in the buffer before transfer can occur. Thus it needs to keep track of free places in the buffer, and free space when there is none.
- The caches are used to speed up system performance, these must be kept up-to-date to make this work.

File-System Structure

File-Organisation Module

- This part of the FS keeps track of actual files and maps these to their blocks in the physical device.
- The file-organisation module translates logical block addresses to physical block addresses for the basic file system to transfer.
- This layer also keeps track of free physical blocks in the device.

File-System Structure

Logical File System

- This layer manages meta-data information, i.e. everything except the actual data (file contents).
- The logical file system manages the directory structure.
- It converts symbolic file names into the IDs the file-organisation module needs.
- It maintains this information in a file-control block (FCB), or inode.
- This layer is also responsible for protection and security, so it handles access permissions for all files.

Volumes

- Each volume contains a boot control block and a volume control block.
- In UFS the boot control block is called the boot block and the volume control block is called a superblock.
- In NTFS it's called the partition boot sector. The volume control block is called the master file table (MFT).
- A directory structure is used to organise the files: in UFS, these include file names and inode numbers; in NTFS this is stored in the MFT.
- Finally, a per file FCB has all information about a file, it contains a unique identifier to match directory entries. (In NTFS this is in the MFT.)

Volumes

- Data about the FS is kept in memory via caches.
- This data is loaded on mount time.
- It's updated during operation.
- It's discarded on dismount.

Kernel Data-Structures

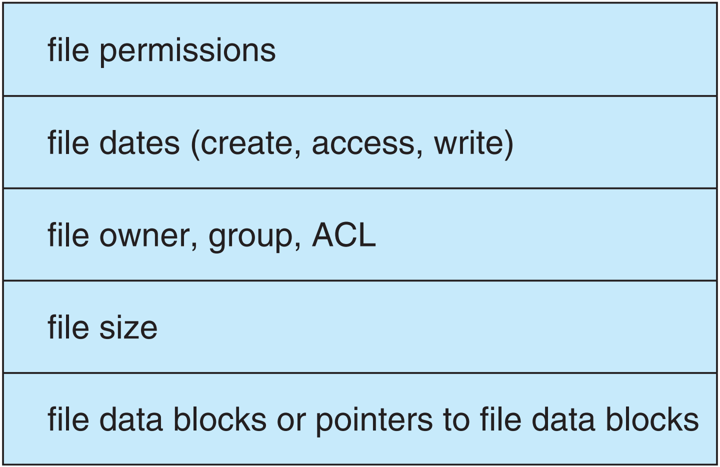
- The OS keeps an in-memory mount table about each mounted volume. (Look at mount(8).)
- An in-memory directory-structure cache holds the directory information about recently accessed directories. (Try `time find /some/subdir -print > /dev/null` two times in a row.)
- The system-wide open-file table keeps a copy of the FCB of each open file. (Look at lsof(8).)
- The per-process open-file table keeps a pointer to the appropriate entry in the system-wide open-file table.
- Buffers hold file-system blocks when they are being read from or written to disk.

Kernel Data-Structures

```
\$ time find ~ -print > /dev/null
real    0m47.535s
user    0m0.479s
sys     0m1.779s
\$
```

```
\$ time find ~ -print > /dev/null
real    0m0.180s
user    0m0.080s
sys     0m0.100s
\$
```

Kernel Data-Structures



file permissions

file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks

Figure: A typical file-control block. Image: [SGG13b].

Operations on Files

- A process which wants to create a new file make a request from the logical file system.
- The logical file system allocates an FCB, either creates a new or take one of the available in case the FCB:s are created at FS creation.
- The open(2) system call passes a symbolic name to the logical file system.
- This first searches the system-wide open-file table, if found it points an entry in the process's open-file table there.
- If not found, the directory structure is searched, when found the FCB is copied into a new entry in the system-wide open-file table.
- We must also track how many processes keep the file open.

Operations on Files

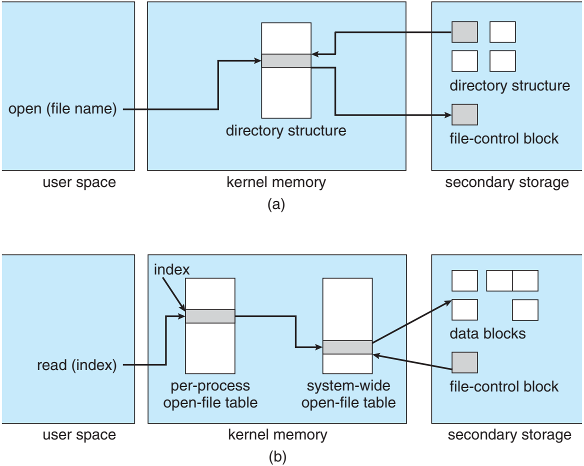


Figure: In-memory file-system structures; (a) file open, (b) file read. Image: [SGG13b].

Overview

1. File Systems
   - File-System Structure
   - Volumes
   - Kernel Data-Structures
   - Operations on Files
2. Virtual File Systems
   - File System Layers
3. Block Allocation and Free Block Algorithms
   - Directories
   - Contiguous Allocation
   - Linked Allocation
   - Indexed Allocation

File System Layers

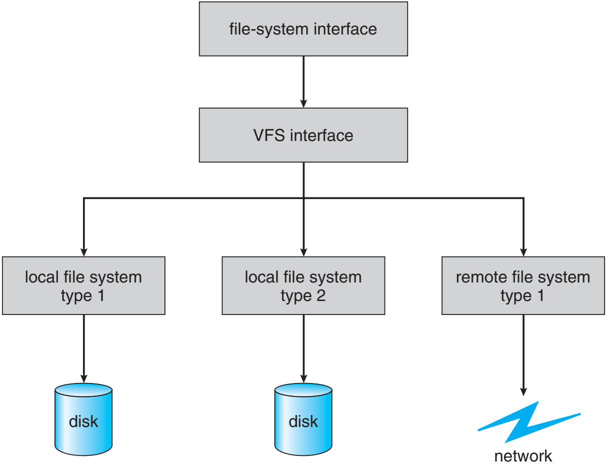


Figure: A schematic view of the virtual file system layer. Image: [SGG13b].

File System Layers

- The virtual file system (VFS) separates generic operations from their implementation. I.e. each FS implements the VFS interface.
- The the OS's FS operations just use the VFS interface, no need to bother about what's underneath.
- This way we can even implement remote file systems like NFS.

File System Layers

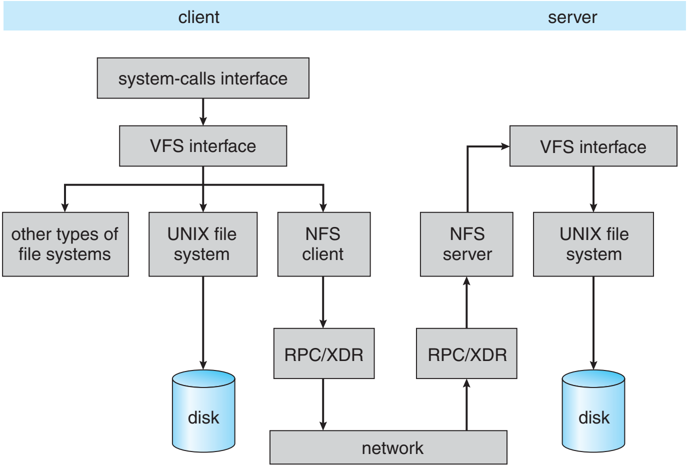


Figure: A schematic of VFS with NFS. Image: [SGG13b].

Overview

Directories

- We need to store and represent our directory structure in some way.
- We have two easy alternatives: linear lists and hash tables.

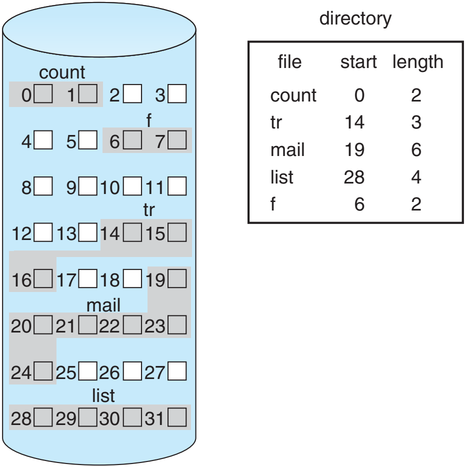
Directories

- The linear list is simple.
- We simply store the file information in a list, from beginning to end.
- To create a file we go through the list to see there's no existing file by that name yet, then we create a new entry.
- To remove a file we free its allocated entry, e.g. by setting it to null or replacing it with the last directory entry and reduce the length of the directory.

Directories

- This is computationally bad, since searching through a list takes time.
- It's easier if it's sorted, then the search is faster. Problem is to keep it sorted.
- We can use a hash table instead, hash function computes a hash value from the symbolic name.
- Problem is collisions, if the collisions are few and evenly spread we can use a linked list for this.

Contiguous Allocation



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figure: An example of contiguous allocation of file data. Image: [SGG13b].

Linked Allocation

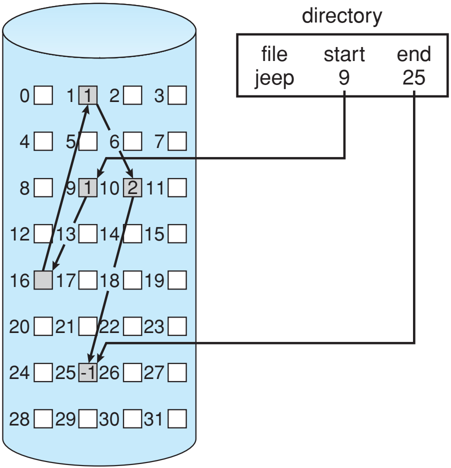


Figure: An example of linked allocation of file data. Image: [SGG13b].

Linked Allocation

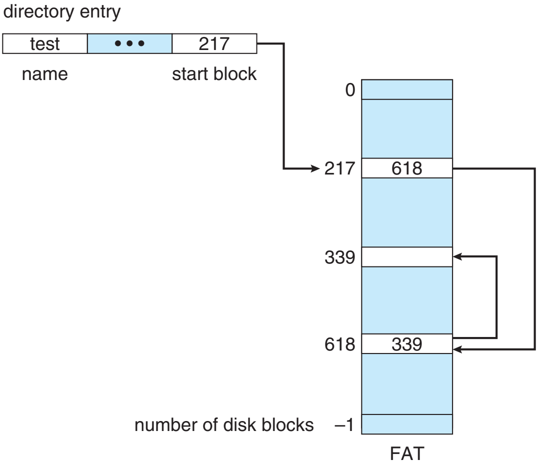


Figure: An example of a file-allocation table (FAT). Image: [SGG13b].

Indexed Allocation

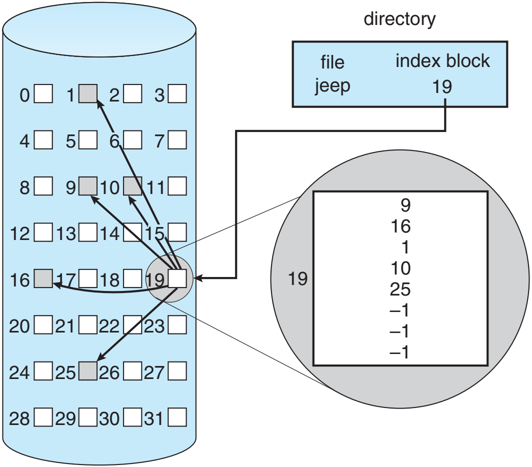


Figure: An example of indexed allocation. Image: [SGG13b].

Indexed Allocation

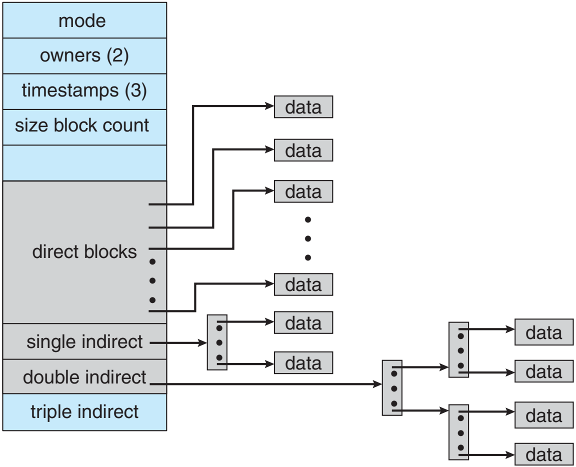


Figure: An example of the UFS inode. Image: [SGG13b].

Referenser I

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. International Student Version. Hoboken, N.J.: John Wiley & Sons Inc, 2013.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. Hoboken, N.J.: John Wiley & Sons Inc, 2013.