# File system implementation

Tran, Van Hoai

Faculty of Computer Science & Engineering
HCMC University of Technology

E-mail: hoai@hcmut.edu.vn

*(partly based on slides of Le Thanh Van)*

# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. Allocation methods
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. Allocation methods
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

# Design principles

## Two design problems

1. Defining how the file system should look like to the user

   files, attributes, file operations, directory structure,...

2. Creating algorithms/data structures to map logical file system to physical secondary-storage devices

# Design principles

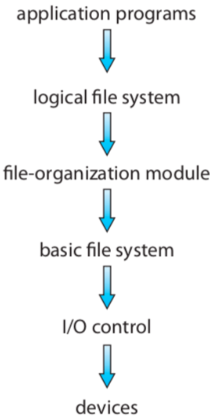
## Two design problems

1. Defining how the file system should look like to the user

   files, attributes, file operations, directory structure,...

2. Creating algorithms/data structures to map logical file system to physical secondary-storage devices

- **Basic file system**: to read/write physical blocks on disk
- **File-organization module**: files, logical/physical blocks
- **Logical file system**: all metadata information (no actual data)



# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. Allocation methods
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

# On-disk and in-memory structures

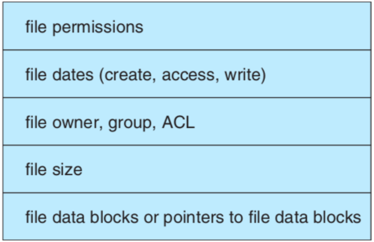
## On-disk

- Boot control block (per volume): first block of volume, contain boot information of an OS
- Volume control block (per volume): partition details (#blocks, block size, free-blocks, free-FCBs)
- Directory structre: file organization
- per-file FCB

## In-memory

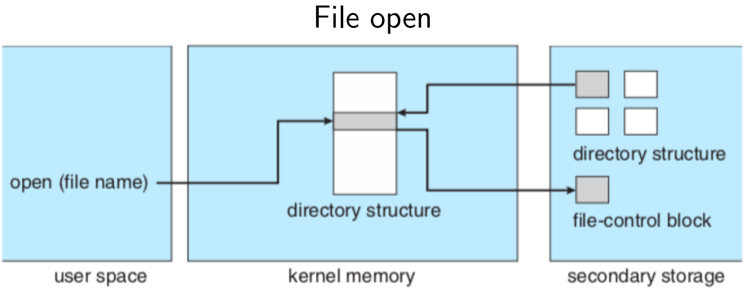
- Mount table
- Directory structure cache
- System-wide open-file table
- Per-process open-file table
- Buffer holding file-system blocks

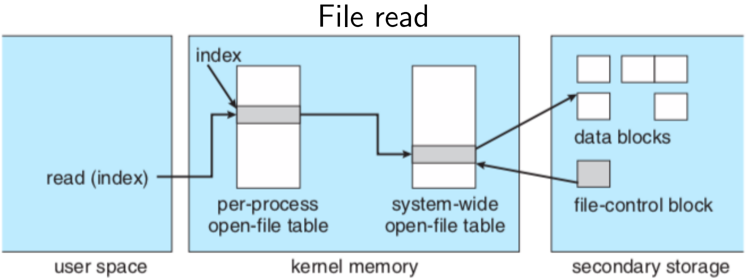
File Control Block



| |
|---|
| file permissions |
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |

# File open/read





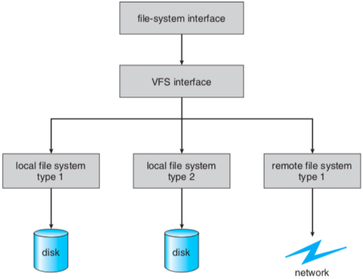
# Virtual file system (VFS)

How to support multiple file systems in a directory structure ?

# Virtual file system (VFS)

How to support multiple file systems in a directory structure ?

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems
- VFS allows the same API to be used for different types of file systems



## API

API (`open`, `close`, `read`, `write`, `mmap`) defined on objects (inode object, file object, superblock object, dentry object)

# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. Allocation methods
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

# Directory implementation

- **Linear list**: a linear list of file names with pointers to data blocks
  - Simple to program
  - Time-consuming to execute
- **Hash table**: a linear list with hash data structure
  - Small directory search time
  - Collisions - situations where 2 file names hash to same location
  - Fixed size

# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. **Allocation methods**
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

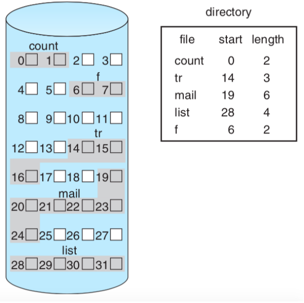
# Types of allocation

Direct-access nature of disks gives us flexibility in the implementation of files.

- An allocation method refers to how disk blocks are allocated for files
- Types:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

# Contiguous allocation (1)

- Each file occupies a set of contiguous blocks on disk
- Simple – only starting location (block #) and length (number of blocks) are required.
- no (or small) disk head movement for random or sequential access



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

- External fragmentation (as in dynamic storage-allocation problem)
- Files cannot grow

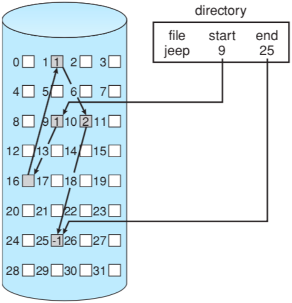
# Contiguous allocation (2)

## Extent-based systems

- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks. Extents are allocated for file allocation. A file consists of one or more extents

# Linked allocation

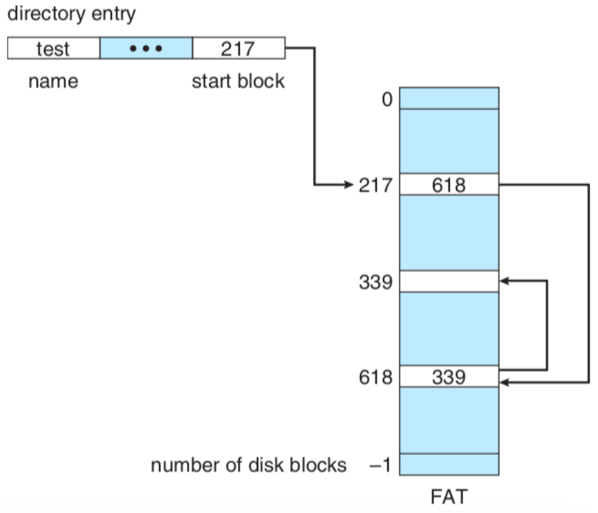
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple - need only starting address
- Free-space management system → no waste of space (no external fragmentation)



- No random access
- Low reliability - especially when losing a block in a chain (no pointer)
- FAT - File Allocation Table (by MS-DOS, OS/2)

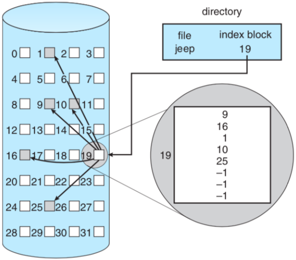
# Linked allocation

## FAT



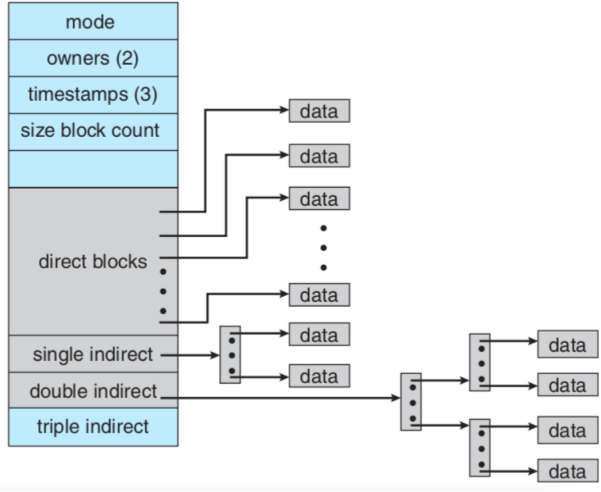
# Indexed allocation

- Bring all pointers together into index block
- random access
- Dynamic access without external fragmentation
- Overhead of index block. How index block adapt with different file sizes ?
  - Linked scheme
  - Multilevel index
  - Combined scheme



# Indexed allocation

## UNIX inode



# Outline

1 File system structure

2 File system implementation

3 Directory implemention

4 Allocation methods

5 **Free-space management**

6 Efficiency and performance

7 Recovery

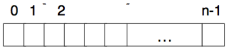
8 Network file system

# Bit vector

## Free-space management

Free-space management is done by free-space list which keeps track disk space

- Bit vector (or bit map):



$$\text{bit}_i = \begin{cases} 0 & \text{block } i \text{ occupied} \\ 1 & \text{block } i \text{ free} \end{cases}$$

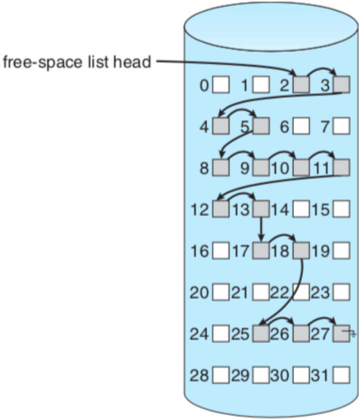
- Block number calculation

  (Number of bits per word)×
  (Number of 0-valued words)+
  offset of first 1 bit

- Copy of bit vector in memory and in disk must be consistent
- Keeping whole bit-vector in memory is expensive
  Block size $= 2^{12}$ bytes
  Disk size $= 2^{30}$ bytes (or 1GB)
  $\Rightarrow n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

# Linked list

- A linked list to link together all the free disk blocks

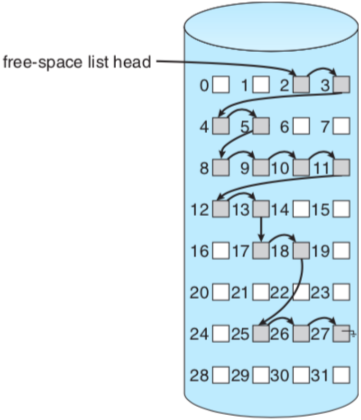


# Linked list

- A linked list to link together all the free disk blocks

**Other free-space management**

- Grouping
- Counting
- Space maps



# Outline

1 File system structure

2 File system implementation

3 Directory implemention

4 Allocation methods

5 Free-space management

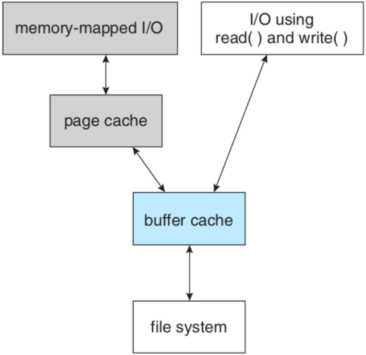
6 Efficiency and performance

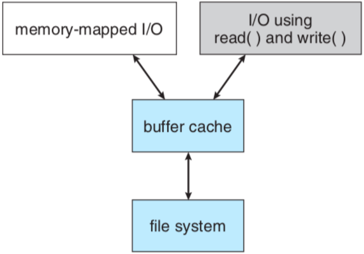
7 Recovery

8 Network file system

# Improving performance by cache

- Disk controllers has its own on-board cache
- OS maintains a separate section of main memory for buffer cache to store blocks used again shortly
- Page cache: using virtual memory techniques to cache file data (in pages, not in file-system oriented blocks)
  ⇒ Unified virtual memory (in Solaris, linux, Windows)





# Outline

1. File system structure
2. File system implementation
3. Directory implemention
4. Allocation methods
5. Free-space management
6. Efficiency and performance
7. Recovery
8. Network file system

# Improving performance by cache

- Consistency checking: compares data in directory 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
- Log structured (or journaling) file systems record each update to the file system as a transaction
  - All transactions are written to a log. A transaction is considered committed once it is written to the log. However, the file system may not yet be updated
  - The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log
  - If the file system crashes, all remaining transactions in the log must still be performed

# Outline

1 File system structure

2 File system implementation

3 Directory implemention

4 Allocation methods

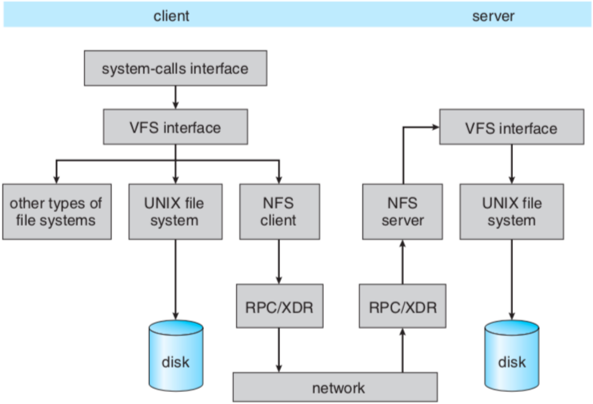
5 Free-space management

6 Efficiency and performance

7 Recovery

8 Network file system

# NFS architecture


client
server
system-calls interface
VFS interface
other types of file systems
UNIX file system
NFS client
RPC/XDR
disk
network
NFS server
RPC/XDR
VFS interface
UNIX file system
disk