操作系统原理与设计

第11章文件系统的实现1

陈香兰

中国科学技术大学计算机学院

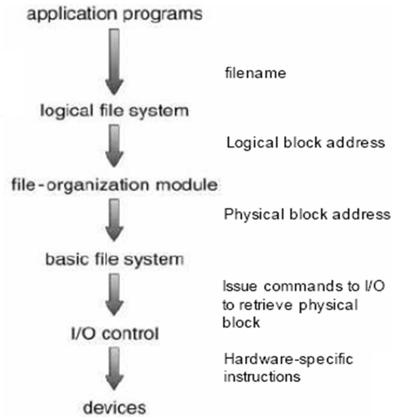
2009年09月01日

提纲

File-System Structure I

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organization
  - How the file system should look to the user
  - How to map the logical file system onto the physical secondary-storage devices
- File system organized into layers
- Layered File System

File-System Structure II



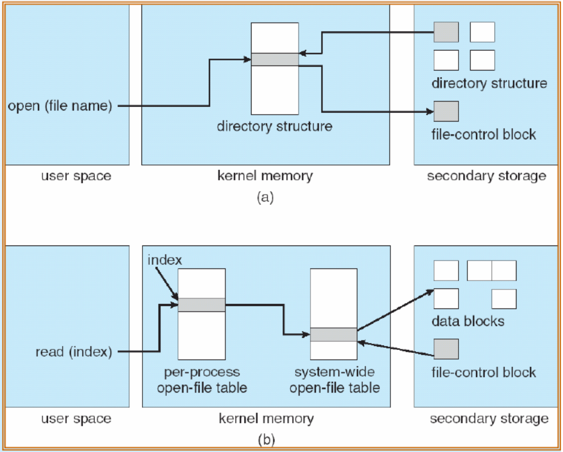
FS Implementation I

- structures and operations used to implement file system operation, OS- & FS-dependment
  - on-disk structures
  - in-memory structures
- on-disk structures
  - boot control block
    - to boot an OS from the partition (volume)
    - if empty, no OS is contained on the partition
  - volume control block
  - directory structure
  - per-file FCB

FS Implementation II

- in-memory information
  - used for both FS management and performence improvement via caching
  - data are loaded at mount time and discarded at dismount
  - structures include:
    - in-memory mount table
    - in-memory directory-structure cache
    - system-wide open-file table
    - per-process open-file table

In-Memory File System Structures



FCB

- A Typical File Control Block

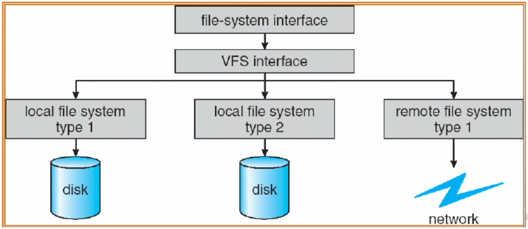
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

partitions and mounting

- partition
  - raw (E.g. UNIX swap space & some database) VS. cooked
  - boot information, with its own format
    - boot image
    - boot loader unstanding multiple FSes & OSes
      dual-boot
- root partition is mounted at boot time
- others can be automatically mounted at boot or manually mounted later

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.
- Schematic View of Virtual File System



Directory Implementation

- Linear list of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - collisions – situations where two file names hash to the same location
  - Fixed & variable size or chained-overflow hash table

Allocation Methods

- An allocation method refers to
  how disk blocks are allocated for files
  so that disk space is **utilized effectively**
  & files can be **accessed quickly**

  1. Contiguous allocation
  2. Linked allocation
  3. Indexed allocation
  4. Combined

1. Contiguous Allocation I

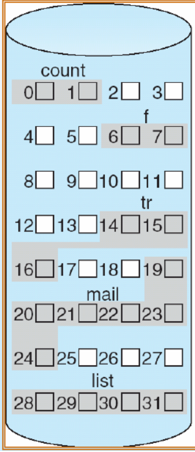
- Each file occupies **a set of contiguous blocks on the disk**
- Simple – directory entry only need
  - **starting location (block #)**
  - & **length (number of blocks)**
- Mapping from logical to physical

  $LogicalAddress/512 = Q \ldots \ldots R$

  Block to be accessed = Q + starting address

  Displacement into block = R

1. Contiguous Allocation II



directory

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

1. Contiguous Allocation III

- Advantages:
  - Support **both random & sequential** access
    - Start block: b;
      Logical block number: i
      ⇒physical block number: b + i
    - **Fast** access speed, because of short head movement
- Disadvantages:
  - **External fragmentation**
    - **Wasteful of space** (dynamic storage-allocation problem).
  - **Files cannot grow**,
    or File size must be known in advance.
    ⇒**Internal fragmentation**

Extent-Based Systems

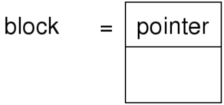
- Many newer file systems (I.e. Veritas File System) use **a modified contiguous allocation scheme**
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks
    - Extents are allocated for file allocation
    - A file consists of one or more extents.

2. Linked Allocation I

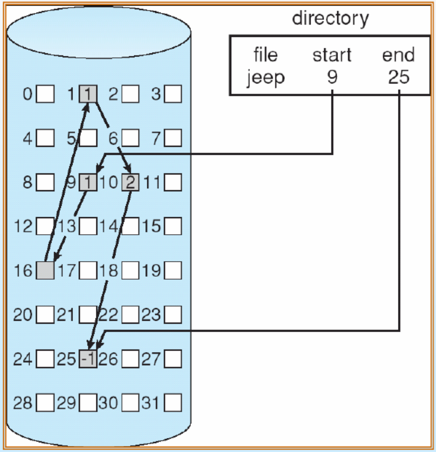
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Directory
  - First block
  - Last block

- Two types
  1. **Implicit**
  2. **Explicit**

1. **Implicit**



2. Linked Allocation II



2. Linked Allocation III

- **Allocate as needed, link together**
  - Simple – need only starting address
- Free-space management system – **no waste of space**
- Disadvantage:
  - **No random access**
  - **Link pointers need disk sapce**
    - E.g.: 512 per block, 4 per pointer ⇒0.78%
    - Solution: **clusters**

    ⇒ disk throughput ↑

    But internal fragmentation↑

2. Linked Allocation IV

- Mapping

  $$512 - 1 = 511$$
  $$LogicalAddress/511 = Q \ldots\ldots R$$

  Block to be accessed is the Q^{th} block in the linked chain of blocks representing the file.

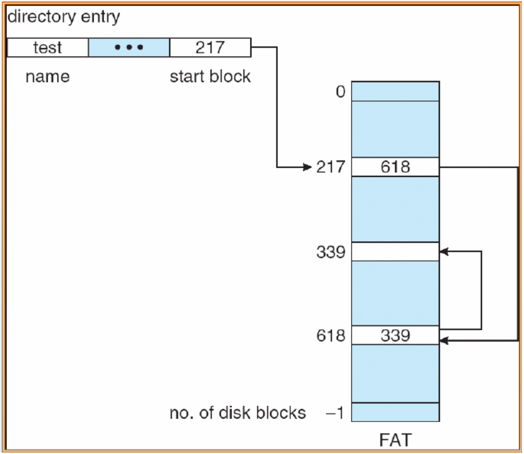
  Displacement into block = R + 1

- **How to reduce searching time?**

File-Allocation Table (FAT) I

- FAT: Explicit linked allocation
  Disk-space allocation used by MS-DOS and OS/2
  - A section of disk at the beginning of each partition is set aside to contain the table
    - Each disk block, an entry, contains the index of the next block in the file
    - Last block entry, end-of-file
    - Unused, 0
- Directory entry contains the first block number
- Now support random access, but still not very efficient
- May result in a significant disk head seeks: Cached FAT

File-Allocation Table (FAT) II



File-Allocation Table (FAT) III

- Compute FAT size

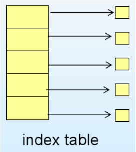
$$\begin{aligned} \textit{Disk space} &: \quad 80\textit{GB} \\ \textit{Block size} &: \quad 4\textit{KB} \\ \textit{Totalblocknumber} &: \quad 80 \times 2^{30}/2^{12} = 5 \times 2^{22} \\ 4 \times 2^{22} = 2^{24} < \quad 5 \times 2^{22} & \quad < 8 \times 2^{22} = 2^{25} \end{aligned}$$

- **Length of each FAT entry?**
- **Length of FAT?**

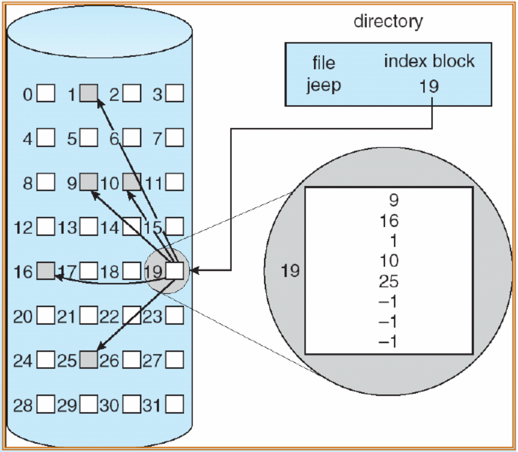
3. Indexed Allocation I

- Brings all pointers together into one location – the index block.
  - Each file has its own index block
  - Index block address is stored in directory entry.
  - Each block is an array of pointers
    - Logical block number i, the i^{th} pointer

- Logical view. index table



3. Indexed Allocation II



3. Indexed Allocation III

- Need index table
- Advantage:
  - **Random** access
  - Dynamic access **without external fragmentation**
- Disadvantage:
  - have **overhead** of index block.
  - File **size limitation**, since one index block can contains limited pointers

3. Indexed Allocation IV

- Mapping from logical to physical
  - maximum file size 256K words & block size of 512 words.

$$256K/512 = 2^{18}/2^9 = 2^9 = 512$$

We need only 1 block for index table.

$$LogicalAddress/512 = Q \ldots\ldots R$$

Q = displacement into index table

R = displacement into block

- **How to support a file of unbounded length? (suppose block size of 512 words).**
  1. linked scheme
  2. multi-level index scheme

3. Indexed Allocation V

1. Linked scheme
    - Link blocks of index table (no limit on size).
    - Mapping

      $$LogicalAddress/(512 \times 511) = Q_1 \ldots\ldots R_1$$

      Q_1 = block of index table

      R_1 is used as follows:

      $$R_1/512 = Q_2 \ldots\ldots R_2$$

      Q_2= displacement into block of index table

      $R2$ displacement into block of file:

3. Indexed Allocation VI

2. Two-level index (maximum file size is 512^3)

   $LogicalAddress/(512 \times 512) = Q_1 \ldots\ldots R_1$

   Q_1 = displacement into outer-index

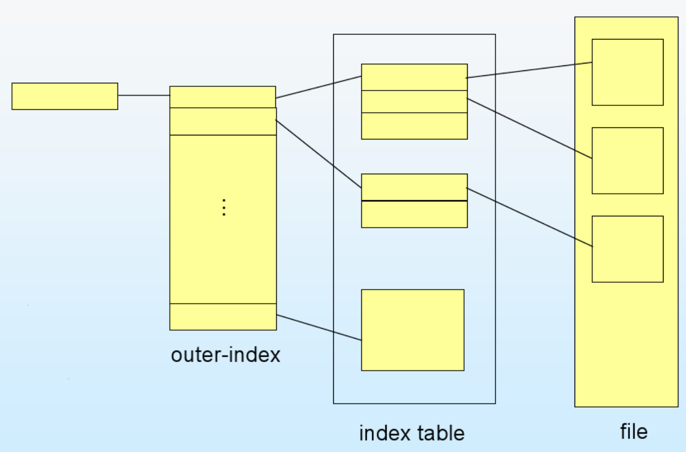
   R_1 is used as follows:

   $R_1/512 = Q_2 \ldots\ldots R_2$

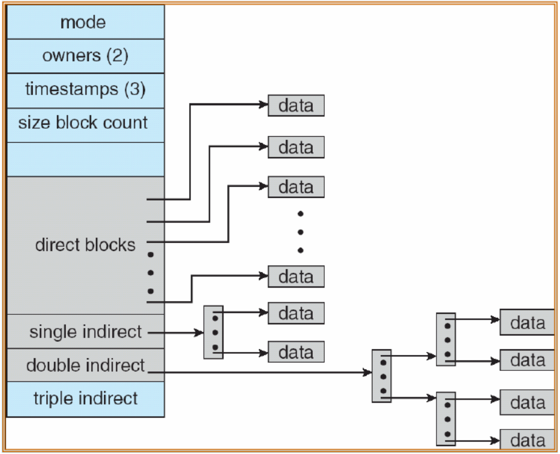
   Q_2 = displacement into block of index table

   R_2 displacement into block of file:

3. Indexed Allocation VII



4. Combined Scheme: UNIX (4K bytes per block) I



4. Combined Scheme: UNIX (4K bytes per block) II

- if 4KB per block, 4B per entry

$$
\begin{aligned}
\textit{Direct blocks} &= 10 \times 4KB = 40KB \\
\textit{Number of entries per block} &= 4KB/4B = 1K \\
\textit{Single indirect} &= 1K \times 4KB = 4MB \\
\textit{Double indirect} &= 1K \times 4MB = 4GB \\
\textit{Triple indirect} &= 1K \times 4GB = 4TB
\end{aligned}
$$

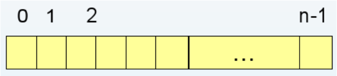
Maximnm file size = ?

Free-Space Management

- Disk Space
  - limited
- Free space management
  - To keep track of free disk space
  - Free-space list
- Algorithms
  1. Bit vector
  2. Linked list
  3. Grouping (成组链接法)
  4. Counting

1. Bit vector I

- Free-space list is implemented as a **bit map** or **bit vector**
  - **Each block $\longleftrightarrow$ 1 bit**
    - 1=free;
    - 0=allocated
  - E.g.: Bit vector (n blocks)



$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \;\; free \\ 0 \Rightarrow block[i] \;\; occupied \end{cases}$$

1. Bit vector II

- **Block number calculation**
  - Base unit:
    - word (**16** bits) or other
  - n blocks:
    - bit map length $= (n + 15)/16$
  - If first K words is 0, & $(K+1)^{th}$ word > 0, the first $(K+1)^{th}$ word's first 1 bit has offset L, then

    first free block $\# = ?$

first free block number $N = K \times 16 + L$

1. Bit vector III

- **Simple**
- Must be kept on disk
  Bit map requires extra space, Example:
  block size $= 2^{12}$ bytes
  disk size $= 2^{30}$ bytes (1 gigabyte)
  n $= 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
  - Solution: Clustering
- **Efficient** to get the first free block or n consecutive free blocks, **if we can always store the vector in memory**.
  - BUT **Copy in memory and disk may differ.**
    E.g. **bit[i] = 1 in memory & bit[i] = 0 on disk**

1. Bit vector IV

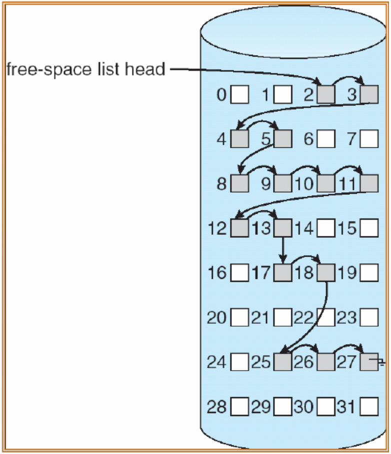
- Solution:
  Set bit[i] = 1 in memory.
  Allocate block[i]
  Set bit[i] = 1 in disk

- Need to protect:
  - Pointer to free list
  - Bit map

2. Linked Free Space List on Disk I

- link together all the free disk blocks
  - First free block
  - Next pointer
- Not efficient
- Cannot get contiguous space easily
- No waste of space

2. Linked Free Space List on Disk II



3. Grouping

- To store the addresses of n free blocks (a group) in the first free block
  - First n-1 group members are actually free
  - Last one contain the next group
  - And so on
- E.g.: UNIX

4. Counting

- Assume:
  - Several contiguous blocks may be allocated or freed simultaneously
- Each = first free block number & a counter (number of free blocks)
- Shorter than linked list at most time, counter > 1

小结

1. File-System Structure
2. FS Implementation
3. Directory Implementation
4. Allocation Methods
5. Free-Space Management
6. 小结和作业

谢谢！