

0117401: Operating System 计算机原理与设计

Chapter 11: File system implementation(文件系统实现)

陈香兰

xlanchen@ustc.edu.cn

<http://staff.ustc.edu.cn/~xlanchen>

Computer Application Laboratory, CS, USTC @ Hefei
Embedded System Laboratory, CS, USTC @ Suzhou

May 17, 2017

温馨提示：



为了您和他人的工作学习，
请在课堂上**关机或静音**。

不要在课堂上接打电话。

提纲

- 1 File-System Structure
- 2 FS Implementation
- 3 Directory Implementation
- 4 Allocation Methods (分配方法)
- 5 Free-Space Management
- 6 Efficiency (空间) and Performance (时间)
- 7 Recovery
- 8 Log Structured File Systems
- 9 小结和作业

1 File-System Structure

File-System Structure

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

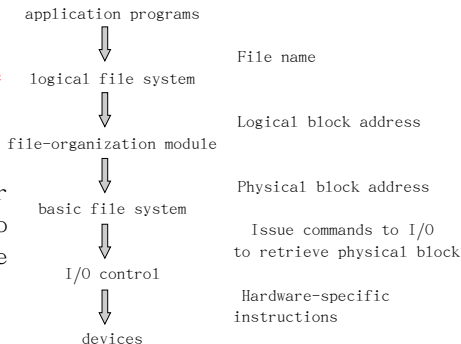


Figure: Layered File System

2 FS Implementation

FS Implementation

- Structures and operations used to implement file system operation, OS- & FS-dependment
 - 1 On-disk structures
 - 2 In-memory structures

① On-disk structures

① Boot control block

- To boot an OS from the partition (volume)
- If empty, no OS is contained on the partition

② Volume control block

③ Directory structure

④ Per-file FCB

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

- ② **In-memory information:** For both FS management and performance 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

② In-memory information: For both FS management and performance improvement via caching

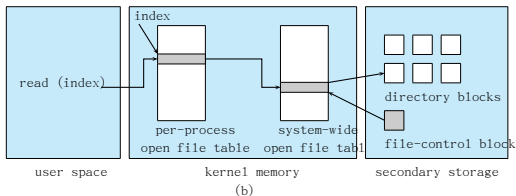
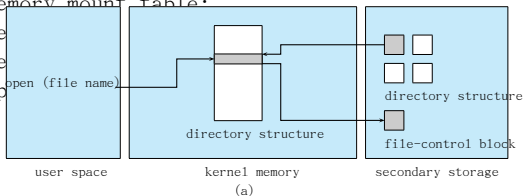
- Data are loaded at mount time and discarded at dismount
- Structures include:

- in-memory mount table:

- in-me

- syste

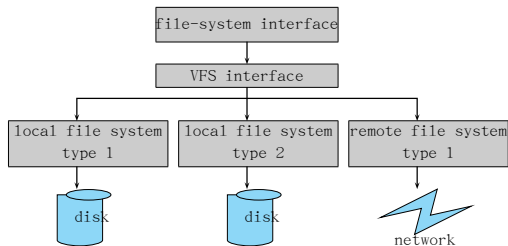
- per-p



- Partition (分区)
 - 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 (虚拟文件系统)

- **Virtual File Systems (VFS, 虚拟文件系统)** 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

3 Directory Implementation

- ① **Linear list** of file names with pointer to the data blocks.
 - Simple to program
 - Time-consuming to execute
- ② **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

4 Allocation Methods (分配方法)

Allocation Methods (分配方法)

- 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 (连续分配)
 - 2 Linked allocation (链接分配)
 - 3 Indexed allocation (索引分配)
 - 4 Combined (组合方式)

1. Contiguous Allocation (连续分配) I

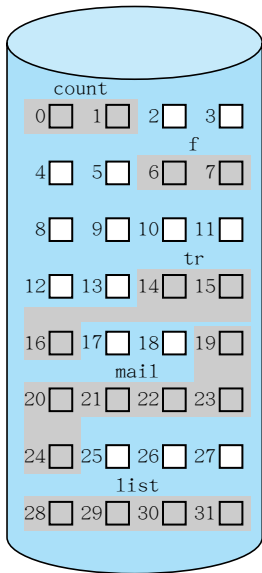
- Each file occupies a set of **contiguous** blocks on the **disk**
- Simple — directory entry only need
 - starting location (block #)
 - & length (number of blocks)
- Mapping from logical to physical

$$\text{LogicalAddress}/512 = Q \dots R$$

Block to be accessed = Q + starting address

Displacement into block = R

1. Contiguous Allocation (连续分配) II



directory

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

1. Contiguous Allocation (连续分配) III

- Advantages:

- Support **both random & sequential** access
 - Start block: b ;
Logical block number: i
 \Rightarrow 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.
 \Rightarrow **Internal fragmentation**

- 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 (链接分配)

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Two types
 - ① **Implicit (隐式链接)**
 - ② **Explicit (显式链接)**

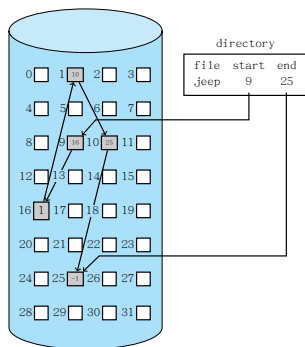
2. Linked Allocation (链接分配)

- 1 **Implicit (隐式链接)**
- **Directory** contains a pointer to the first block & last block of the file.
- **Each block** contains a pointer to to the next block.

a block =

pointer

- **Allocate as needed, link together**
 - Simple — need only starting address
 - Free-space management system — **no waste of space**



2. Linked Allocation (链接分配)

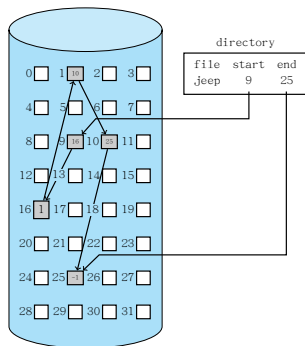
① Implicit (隐式链接)

● Disadvantage:

- No random access
- Link pointers need disk space
E.g.: 512 per block, 4 per pointer $\Rightarrow 0.78\%$

Solution: clusters

\Rightarrow disk throughput \uparrow
But internal fragmentation \uparrow



2. Linked Allocation (链接分配)

① Implicit (隐式链接)

• Mapping:

Suppose

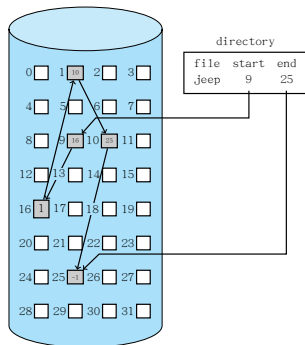
- ① block size=512B,
- ② block pointer size=1B, using the first byte of a block
- ③ Logical addr in the file to be accessed=A

we have

- ① Data size for each block = $512 - 1 = 511$
- ② $A/511 = Q \dots R$

then

- ① Block to be accessed is the Q^{th} block in the linked chain of blocks representing the file.
- ② Displacement into block = $R + 1$



• How to reduce searching time?

2. Linked Allocation (链接分配)

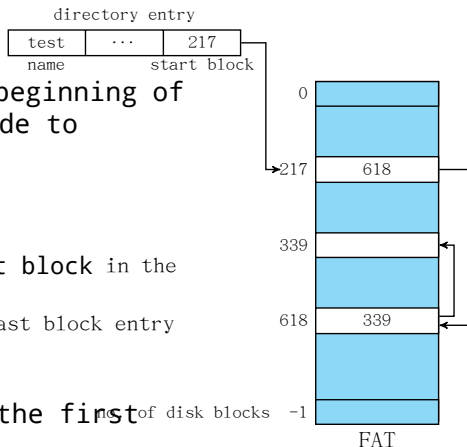
② Explicit linked allocation: File Allocation table, FAT

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 FAT

- Each disk block one entry
- The entry contains
 - (1) the index of the next block in the file
 - (2) end-of-file, for the last block entry
 - (3) 0, for unused block

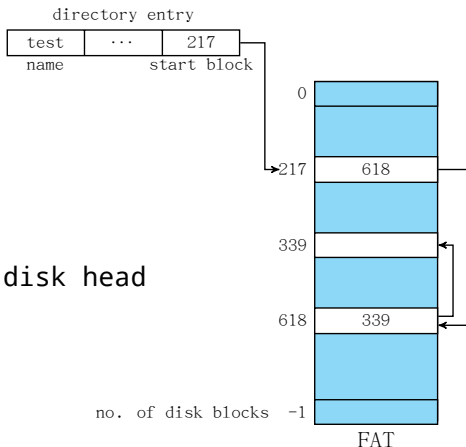
- Directory entry contains the first block number



2. Linked Allocation (链接分配)

② Explicit linked allocation: File Allocation table, FAT

Disk-space allocation used by MS-DOS and OS/2



- Now support random access, but still not very efficient
- May result in a significant disk head seeks.

Solution: Cached FAT

2. Linked Allocation (链接分配)

② Explicit linked allocation: File Allocation table, FAT

Disk-space allocation used by MS-DOS and OS/2

• How to compute FAT size?

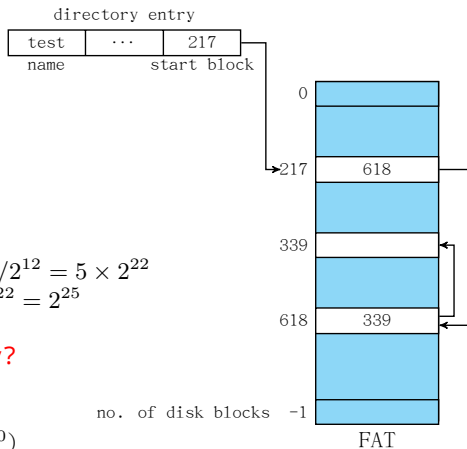
Suppose

- ① Disk space = 80 GB
- ② Block size = 4 KB

Then

- ① Total block number = $80 \times 2^{30} / 2^{12} = 5 \times 2^{22}$
- ② $4 \times 2^{22} = 2^{24} < 5 \times 2^{22} < 8 \times 2^{22} = 2^{25}$

- Length of each FAT entry?
(25bits? 28bits? 32bits?)
- Length of FAT?
($5 \times 2^{22} \times 4B = 80MB = 80GB/2^{10}$)



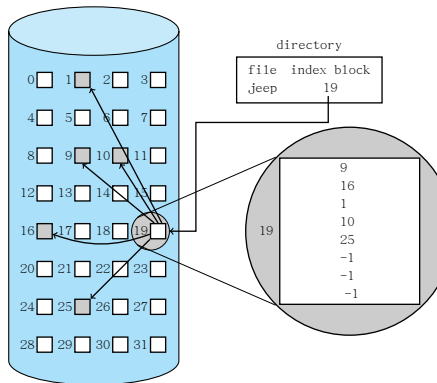
3. Indexed Allocation (索引分配)

- **Indexed Allocation (索引分配):**

Brings all pointers together into one location — **the index block.**

- **Each file** has its own index block
- **Directory entry** contains the index block address
- **Each index block:** An array of pointers (an index table)

Logical block number i
= the i^{th} pointer



3. Indexed Allocation (索引分配)

- **Indexed Allocation (索引分配):**

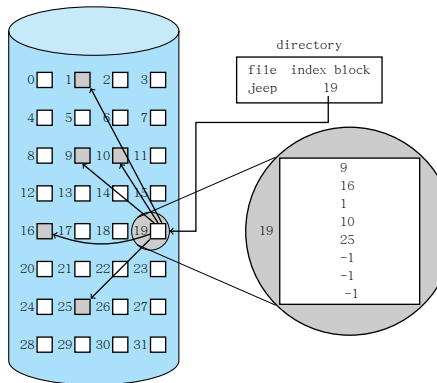
Brings all pointers together into one location — **the index block**.

- **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 (索引分配)

- **Indexed Allocation (索引分配):**

Brings all pointers together into one location — **the index block**.

- **Mapping** from logical to physical

Suppose

(1) Block size = 1KB

(2) Index size = 4B

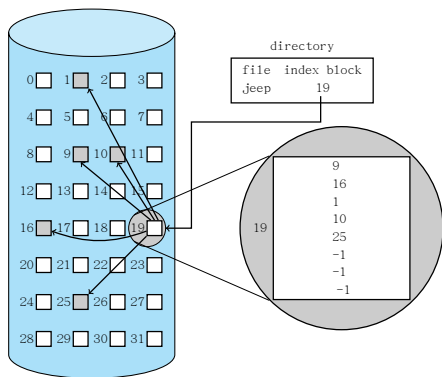
Then for logical address LA, we have

$$LA/512 = Q...R$$

(3) Q = the index of the pointer

(4) R = displacement into block

We also have **Max file size**
= $2^{10}/4 \times 1KB = 256KB$



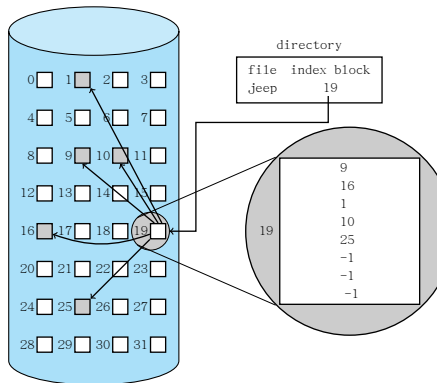
3. Indexed Allocation (索引分配)

- Indexed Allocation (索引分配):

Brings all pointers together into one location — the index block.

- How to support a file of unbounded length?

- 1 linked scheme
- 2 multi-level index scheme



3. Indexed Allocation (索引分配)

① Linked scheme

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

Suppose

- (1) Block size=1KB
- (2) Index or link pointer size = 4B

Then

$$LA / (1KB \times (1K/4 - 1)) = Q_1 \dots R_1$$

(3) Q_1 = block of index table

(4) R_1 is used as follows:

$$R_1 / 1K = Q_2 \dots R_2$$

(5) Q_2 = index into block of index table

(6) R_2 = displacement into block of file:

3. Indexed Allocation (索引分配)

② multi-level index scheme

Example: **Two-level index** (maximum file size is ?)

- We have

$$LA / (1K \times 1K / 4) = Q_1 \dots R_1$$

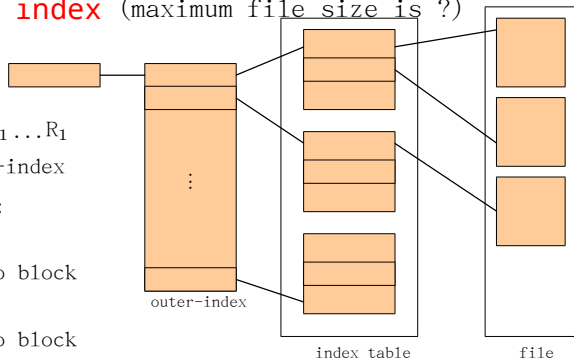
(1) Q_1 = index into outer-index

(2) R_1 is used as follows:

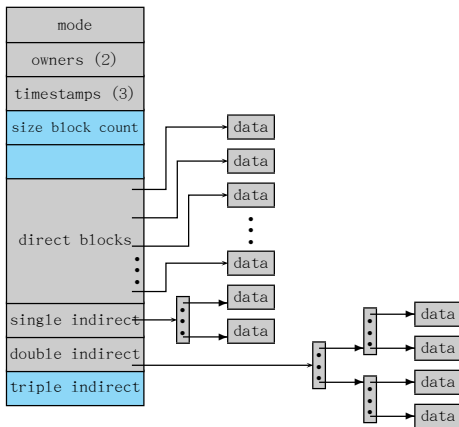
$$R_1 / 1KB = Q_2 \dots R_2$$

(3) Q_2 = displacement into block of index table

(4) R_2 = displacement into block of file



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



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

- if 4KB per block, and 4B per entry

$$\text{Direct blocks} = 10 \times 4\text{KB} = 40\text{KB}$$

$$\text{Number of entries per block} = 4\text{KB}/4\text{B} = 1\text{K}$$

$$\text{Single indirect} = 1\text{K} \times 4\text{KB} = 4\text{MB}$$

$$\text{Double indirect} = 1\text{K} \times 4\text{MB} = 4\text{GB}$$

$$\text{Triple indirect} = 1\text{K} \times 4\text{GB} = 4\text{TB}$$

Maximmm file size = ?

5 Free-Space Management

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

Bit vector

- **Free-space list** is implemented as a **bit map** or **bit vector**

- **1 bit for each block**

1=free;

0=allocated

- Example:

a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26,27 are free and the rest blocks are allocated. The bitmap would be

0011 1100 1111 1100 0110 0000 0111 0000 0...

- **Bit map length.**

For n blocks, if the base unit is word, and the size of word is 16 bits, then

$$\text{bit map length} = (n + 15)/16$$

```
U16 bitMap[bitMapLength];
```

① Bit vector

- How to find the first free block or n consecutive free blocks on the disk?
 - Many computers supply **bit-manipulation instructions**
 - To find the first free block:
Suppose: base unit = word (16 bits) or other
 - (1) find **the first non-0 word**
 - (2) find **the first 1 bit** in the first non-0 word
 - 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

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

① Bit vector

- 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

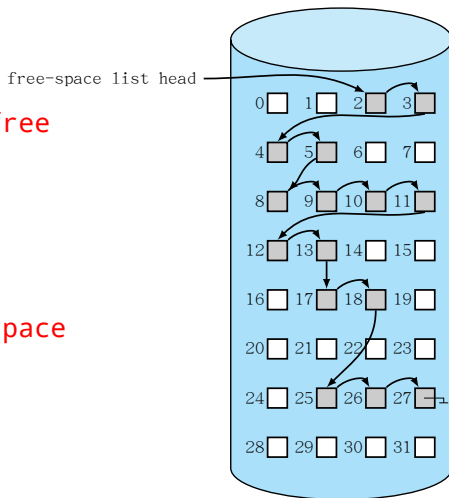
① Bit vector

- **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
 - **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

Free-Space Management

② Linked Free Space List on Disk

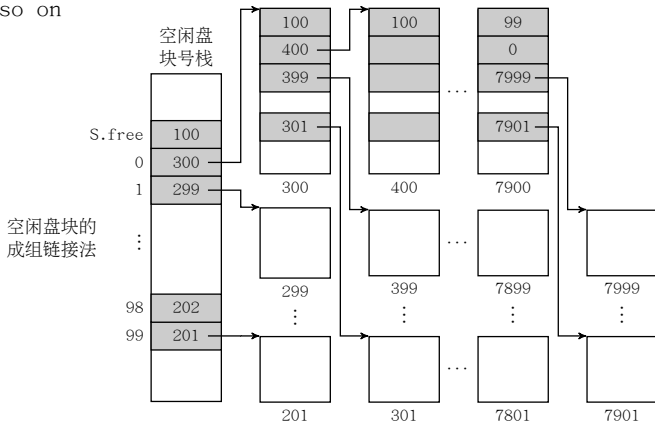
- Link together all the free disk blocks
 - **First** free block
 - **Next** pointer
- Not efficient
- **Cannot get contiguous space easily**
- No waste of space



Free-Space Management

● **Grouping**(成组链接法): To store the addresses of n free blocks (a group) in the first free block. E.g.: UNIX

- First n-1 group members are actually free
- Last one contain the next group
- And so on



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, generally counter > 1

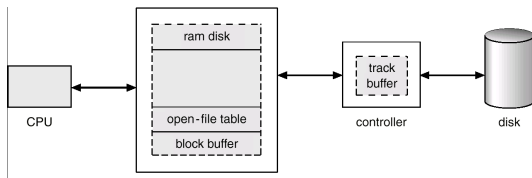
6 Efficiency (空间) and Performance (时间)

Efficiency in usage of disk space dependent on:

- ① Disk allocation and directory algorithms
- ② **Various approaches**
 - Inodes distribution
 - Variable cluster size
 - Types of data kept in file's directory entry
 - Large pointers provides larger file length, but cost more disk space

2 Performance (时间)

- **Performance:** other ways
 - **disk cache** - on disk controllers, large enough to store entire tracks at a time.
 - **buffer cache** - separate section of main memory for frequently used **blocks**
 - **page cache** - uses virtual memory techniques to **cache file data as pages** rather than as file-system-oriented blocks
 - **Synchronous writes** VS. **Asynchronous writes**
 - **free-behind and read-ahead** - techniques to optimize sequential access
 - improve PC performance by dedicating section of memory as virtual disk, or **RAM disk**



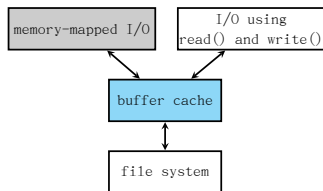
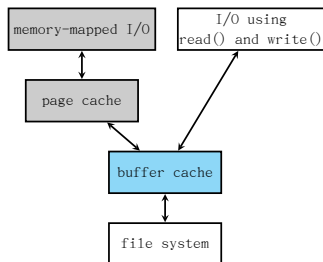
Unified Buffer Cache

• I/O Without a Unified Buffer Cache

- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- **Problem:** double caching

• I/O Using a Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O



7 Recovery

- Consistency checking (一致性检查)

- **compares** data in directory structure with data blocks on disk, and **tries to fix** inconsistencies
- UNIX: fsck
- MS-DOS: chkdsk

- Backup & restore

- Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup
- A typical backup schedule may be:
Day1: **full backup**;
Day2: **incremental backup**;
...
DayN: incremental backup. Then go back to Day1.

8 Log Structured File Systems

- **Log-based transaction-oriented** (or journaling, 日志) 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

9 小结和作业

小结

- 1 File-System Structure
- 2 FS Implementation
- 3 Directory Implementation
- 4 Allocation Methods (分配方法)
- 5 Free-Space Management
- 6 Efficiency (空间) and Performance (时间)
- 7 Recovery
- 8 Log Structured File Systems
- 9 小结和作业

12.6 设想一个在磁盘上的文件系统的逻辑块和物理块的大小都是512字节。假设每个文件的信息已经在内存中。对三种分配方法（连续分配、链接分配和索引分配），分别回答下列问题：

（1）逻辑地址到物理地址的映射在系统中是怎样进行的？（对于索引分配，假设文件总是小于512块长）；

（2）假设现在处在逻辑块10（最后访问的块是10），现在想访问逻辑块4，那么必须从磁盘上读多少个物理块？

谢谢!