CS5600: Operating Systems | Lecture Note 9-1 @ Nov. 10, 2010.
Instructor: Alan Mislove | Subscriber: Wu Jiang

Lecture Note 9-1

1 Announcements

- There are only two hours of lectures tonight. The 3rd hour lecture will be used for canceled project 2 interviews
- Project 2 result will be out next week (Nov. 15 - Nov 19.)
- Project 3 milestone is due next Tuesday (Nov. 16)
- Lab session: this coming Tuesday (Nov. 16), 17:00 - 19:00

2 Outline

last lecture
- functions of file systems
- interfaces to file systems
- sharing and protection

today
- how to track files on disk
- file and directory structure on disk
- how to evaluate profermances

3 Directory Layout

Basic questions to ask

1. What information should be stored for a file?
2. How to track these information?

The first and most important thing we should store is the file name. Besides its name, we need to keep track of its modification data, access permission, access time, etc. One straightforward approach is storing them together into directory. One improved approach is to put all the information except directory name into a block – file control block (FCB), and then keep a pointer pointing to the FCB. In other words, we only keep file name and the FCB pointer within the file's data structure. There are two main reasons why the second approach is better than the first one: more efficient, and easier to share.

Data structure. As what we discussed above, we need to keep the name and the pointer pointing to FCB in the file's data structure. There are 3 basic data structure we may use — static array, list, and hash table. The single element within the array, list, or hash table has two contributes — file name and pointer to FCB. These are some comparison between the three data structure (See table 1).

In general, there are 4 basic operations on a directory.

	static array	**dynamic list**	**hash table**
Advantage	- quick access	- easy to extend - easy to delete	- quick access - easy to extend - easy to delete - easy to search
Disadvantage	- poor memory usage - hard to extend	- poor memory usage - slow access - hard to search	

Table 1: Comparison between different directory data sturctures

- CREATE (dir_name, . . .)
- DELETE (dir_name, . . .)
- LIST (dir_name)
- LOOKUP (dir_name, file_name)

If we use array, since it is static, each time we create a new directory, we need to create a new bigger array, and then copy the old array's content. For a linked list, it can solve this problem by just add a node to the linked list. Deletion is also trivial for a link list. But when dealing with lookup (searching), we have to traverse the list from the beginning because linked list does not support random access. Hash table does not have any of these problems.

4 File layout

Files are stored as data blocks on disk. We'll discuss several different allocation approaches.

Contiguous allocation. Data blocks are stored contiguously on disk for each file. For a file, we only need to keep track of its base location on the disk and the file length. As long as we know these, we can access any data block by offsetting.

Advantages:

- Simple
- Good performance (fewer disk seeks, only one piece of data for each file)
- Direct map between physical data to logical file

Disadvantages:

- External fragmentation — some small holes can not fit for bigger files
- Dynamic allocation problems (best fit, worst fit both have draws)

- Have to know file size before allocation. But a file's size usually changes over time.

One solution to solve the third problem is using linked list — whenever a file's size needs extend, allocate new blocks and use a pointer pointing to the new blocks.

Linked allocation. Each block has a pointer.
Advantages:

- No external fragmentation
- Simple

Disadvantages:

- Less efficient
- Slow random access (can only find pointer, not offset)
- Single pointer of failure (one fails, all fail)

One solution for the third problem is using more pointers – one pointer pointing to the next block, another pointer pointing to the next of next block, etc. Or we may use **doubly linked list**.

Linked allocation in practice — FAT (File Allocation Table). There are two data structures for meta data. For directory, we have file name and a pointer pointing to the first block. Then we have a file allocation table, in which, each block is a pointer entry (pointing to the next entry). This can provide faster random access because it has already been in the memory, but accessing memory is still very expensive (think about the case that we are trying to find the 1,000th block).

Index allocation. Keep all block information for one file in one place. Create an index of the pointers instead of just pointing to the next. An block on the disk will have file data and an index.
Advantages:

- Access fast. If we want to find the 1,000th block, just find the 1,000th index.
- No external fragmentation

Disadvantages:

- Limits on the number of pointers, which will limit file size. Suppose a disk block is 4KB, and a pointer is 4B, then we can have no more than $4KB \div 4B = 1K$ pointers. Then the file size can be only $1K \times 4KB = 4MB$.
- High overhead on small files

One solution to the first problems is using **indirect blocks**. We have more than one layers for pointers — each pointer points to a block of pointers which points to file blocks. But this will increase the overhead degree for small files. One improvement is to combine one-layer approach and more-than-one-layer approach — The first n_1 pointers directly point to file blocks, the following n_2 pointers using two layers, and the next n_3 pointers using three layers, and so on. This will make sure that the first access to the file faster.

Free blocks list. Keep a free blocks list to keep track of all unallocated blocks on disk.