CS2106 Live Class: File Management Introduction

$_/|/|U_C h@NgRu!$

May 10, 2021

1 Function of File Manage System

File system is introduced on top of external storage to store persistent information, in contrast to physical memory storing volatile(relatively temporary) information
File system should provide:

- Abstraction from physical properties of physical storage into a logical storage unit, file
- A high level resource management scheme
- Protection between users and processes
- Sharing between processes and users

The requirement of file system needs to be

- Self-contained, where information stored on a media is enough to describe the entire organization; should be able to "plug and play" on another system
- Persistent: data should persist beyond the lifetime of OS and processes(non-volatile)
- Efficient: It should provide good management of free and used space, and there should be minimum overhead for bookeeping information

	Memory management	File System Management
Underlying Storage	RAM	DISK
Access Speed	Constant(relative)	Variable disk I/O time(disk rotate time v
Unit of Addressing	Physical memory address	Disk sector
Usage	Address space for process implicit when process runs	Non-volatile data explicit access(every tim
Organization	Paging/Segmentation: determine by HW & OS	Many different FS: ext*(Linux), FAT*(W

2 File System Abstraction

File system consists of a collection of files and directory structures, where

- File: an abstract storage of dta
- Directory: Organization of files

File systems provide an abstraction of accessing and using the files and directories.
FIle represents a logical unit of information created by process. It can be viewd as an abstract data type, in which there are

3 Component of File

data and meta data

- Data: information structured in some ways
- Metadata:Additional information associated with the file, also known as file attributes. Common metadata fields include
  - Name: a human readable reference to the file
    Different file system will have different naming rule. Some system include file extension in file name like Name. Extension so as to use the extension to indicate the file type(zip, mp4, pdf)(Windows is using the extension system to distinguish files while linux is not)
  - Identifier: A unique id for the file used internally by file system, the user cannot understand the identifier
  - Type: indicate different type of files
  - Position: a pointer to a device and the location on that device
  - Size: current size of file
  - Protection: Access permission, can be classified as reading, writing, and execution rights
    We can specify the type of access certain user or group of user can do on the file.
  - Time: data and owner information: creation, last modification time, owner id etc
  - Table of content: information for the file system to determine how to access the file

In directory, typically file name and identifier are recorded. The identifier can locate to other information of the file. It may take kByte for recording identifier and name, some big directories may itself be mByte

3.1 Types of file

there are 3 common file types:

1. Regular file: contains user information. Inside regular files, there are two major types:
   - ASCII files, which can ber displayed as it is
   - Binary files, which include executable, PDF files etc. It has predefined internal structure than can be processed by specific programme

   We can use either file extension(windows) or embedded information(UNIX) to distinguish file type.
2. Directories: System files for File System structure
3. Special File: character/block oriented

3.1.1 Another types classification

1. Text file: sequence of characters organized in lines
2. Source file: sequence of characters organized in subroutine and functions, each of which is further organized as declaration and executable
3. Object file: a sequence of characters organized in block understandable by system's link

4. Executable file: a sequence of characters organized as code section, which the loader can bring into memory and execute
5. batch file: like .bat or .sh, which commands the interpreter

The unix uses crude magic number stored at the beginning to roughly indicate the type of the file(png, pdf, zip),Not all files have magic numbers, so system features cannot be based solely on this information.
In linux, the extension is used for user or application to clarify
Windows uses extension to clarify the file types

3.2 Protection of file

The type of access include:

- Read: retrieve information from file
- Write: write /rewrite the file
- Execute: load file into memory and execute it
- Append: Add new information to the end of the file(question: what is the difference between append and wirte)
- Delete: remove file from File system
- List: Read metadata of the file

The most general scheme to specify these rights is to use a access control list, in the format of a list of user identity with their allowed access types. It is very customizable but additional information needs to be associated with the file. Permission bit is another way to protect the file

3.2.1 Permission bit

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

1. Owner: The first three characters (2-4) represent the permissions for the file's owner. For example, -rwxr-xr– represents that the owner has read (r), write (w) and execute (x) permission.
2. Group: The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, -rwxr-xr– represents that the group has read (r) and execute (x) permission, but no write permission.
3. Other users: The last group of three characters (8-10) represents the permissions for everyone else. For example, -rwxr-xr– represents that there is read (r) only permission.

3.3 Operation on file metadata

- Rename
- Change attributes, like file access permissions, dates, ownership etc.
- Read attribute, like file creation time

3.4 Ways of abstraction

Array of bytes: there is no interpretation of data, so the data are just raw bytes. Each byte has a unique offset from the file start

- Fixed length records: array of records, which can frow/shrink. It can jump to any record easily since the offset of nth record is size of record x (n-1)
- Variable length records: which is flexible but harder to locate a record(but unable to access n quickly

3.5 Access method

File data can have three different ways for Access:

- Sequential access, which reads data in order, starting from beginning. Data can be skipped and rewound
- Random access, where data can be read in any order. This can be realized using
  - read(offset), which explicitly state the position to be accessed
  - seek(offset), which moves from old to a new location in the file(used by windows and unix) (difference from read: set the position first)
- Direct access: which is used for files that contains fixed length records. It allows randomly access to any record directly
  Basic random access method is a special case of direct access, where each record equals 1 byte

4 Operations on file data

- Create: create a new file with no data; Two steps: 1. free space must be found in file system; 2. an file entry must be made in a directory
- open: perform before the further operation, so as to prepare necessary information for file operations later
- Read: read data from file, usually starting from current position; Procedure: 1. A system call is invoked, where the user specify both the file name and the location needs to be read inside the file; 2. The system will search the name in the directory; 3. Once got, the system will use the location pointer to read the contents in the file; 4. the pointer needs to be updated whenever read happens; 5. Because read/write happends a lot in process, a per-process file-location pointer can be maintained, which is shared by read and write
- Write: write data to file, usually starting from current position; Procedure: 1. A system call must be made where the user provide both the name of the file and the data need to write; 2. the system will search the file name in the directory; 3. If the system find the file, a pointer must be maintained, which records the position needs to to write next; 4. the pointer must be updated whenever write happens; 5. Because read and write happens frequently, the current location of the pointer can be kept as a per-process file-position pointer, the write and read share the same pointer
- Reposition: The directory is searched for appropiate entry, and the current file-location pointer is replaced by a given value; This also called file seek

- Truncate: If the user want to delete the file contents but keep the attributes, the Truncate can keep all attributes of the file remain unchanged except the file length(file length is reset to 0); All the file data space is released so that other file can make use
- Delete: first search the directories for certain name; having found the file entries, then we free all file spcae, so that it can be reused by other files

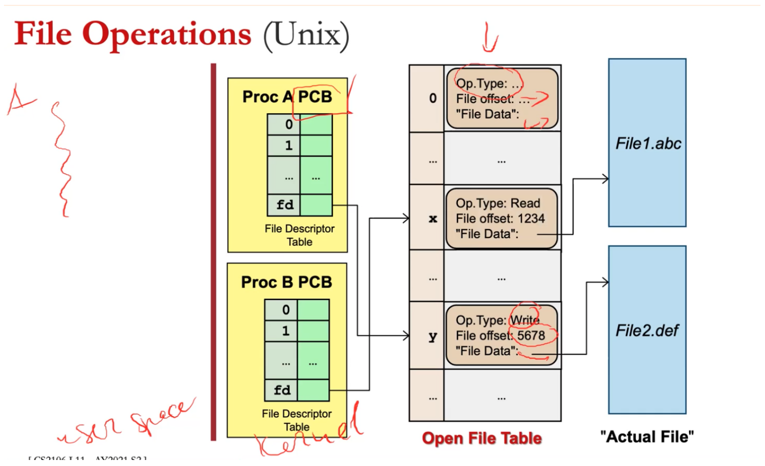
These operations are provided by OS via system calls, which provides protections, concurrent and efficient access, and also maintain necessary information. Especially, OS need to keep information for opened file in a open-file table:

- File pointer: current location in file
- Disk location actual file location on disk
- Open count: how many process has the file opened

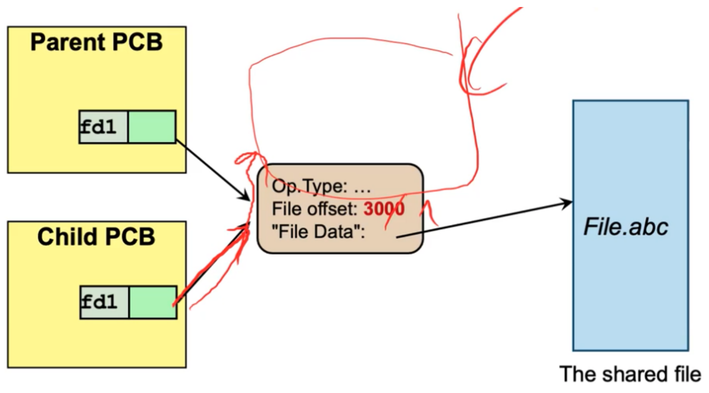
To keep track of these information, we need a system-wide open-file table and also some per-process open-file table. The system-wide table's entry corresponds to a unique file, whereas the per-process open-file table entry corresponds to an entry in a system-wide table

For every process, in the PCB, it records file information that process used, which is an entry of the system-wised table
IMPORTANT: A fie may have different descript

4.1 An example



Two processes using different file descriptors is like above, though they visit the same file, the file descriptors are totally different
The situation above can happen when we call fork() first then open file



The image above shows two processes using the same file descriptors
The situation above can happen when we open file first then call fork()

4.2 File-open table

All file operation needs searching the directory, so a file-open table can be kept for opened files, which will keep the information of open files
Usually system uses two level of internal table, one level is per-process file table. which stores the file information opened by a process(name, file pointer, accounting, access right)
The per-process file table points to system-wise open file table
System-wide page table contains process independent information(location of files, access date and file size)

4.3 information associated with open file

1. File pointer: record the read/write location of a process on the file, the information is process dependent and can applied on only open file
2. File-open count: there may be multiple processes open the same file; the file entries in the open-file table can only be cleared when all process close
3. Disk location of the file; If the process needs to modify the file(happens frequently), it needs to locate the disk location
4. Access rights: the access right of a process to certain file can be stored in the per-process file table so that the system can allow/deny the process's access

4.4 file lock

behave like semaphore, the exclusive file lock is like mutex

5 Directory

Directory is used to

- Provide a logical grouping of files
- Keep track of files

We can structure the directory as

- Single-level
- Tree structure, which allows us to have an absolute pathname, and a relative pathname from the current working directory
- DAG, by allowing files to be shared. This sharing can be done using
  - Hard link, which can only be used on files
    A hard link of a file F in directory a to another directory B makes A and B have seperate pointers pointing to the actual file in disk. This has low overhead, since only pointers are added in directory
    In linux, the hard can be built with ln
  - Symbolic Link, which can be used on files and firectory.
    In the same scenario above, now B will create a special link file G where G contains the path name of F. Whne G is accessed, it finds out where is F and access F.
    This solves the problems of deletion, but has large overhead.
    In linux, ln -s can achieve sybolic link
- General graph
  This is not desirable, since we may encounter infinite loop in traversal and it is hard to determine when to remove a file or directory

6 File system implementation

General disk structure can be treated as a 1-D array of logical blocks(the logical block is the smallest accessible unit)
Each logical block is mapped to a hardware sectors; The layout of the hardware sectors is system independent
In hardware sector 0, there is a master boot record(MBR), which stores partition table. One or more partition tables follow the MBR; each partition contains an independent file system

6.1 Components of a file system

- OS bootup information
- **partition details**,Total number of blocks, numbers, location of free blocks
- Directory structure
- File information
- Actual file data

6.2 Implementing File

File is a collection of logical blocks. However, the file size may not be multiple of logical block size, which will leads to internal fragmentation
A good implementation of file system can

1. Keep track of logical blocks
2. efficient access
3. Disk space is utilized effectively

6.3 Contiguous Block Allocation

In contiguous block allocation, we allocate **consecutive blocks** to a file. It is simple to keep track of continuous file because we only need to remember the starting position and the length(offset)
However, the contiguous block allocation may have external fragmentation issues
Also this scheme needs the file size to be specified in advance

6.4 Linked List Allocation

A linklist of disk blocks, each block contains file data and block number of the next block
The file information keeps the first and the last block number
The linked list allocation solves the external fragmentation problem, but

- Slow for random access
- each block needs memory to store the block number of the next block(less disk efficient)
- not reliable, each pointer can be a point of failure

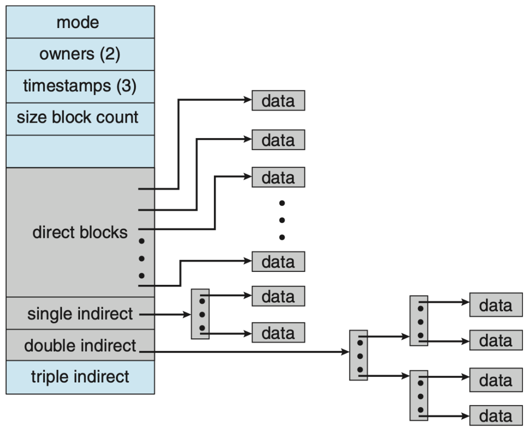
Why not use a file allocation table(FAT) in memory, which can keep track of each block and help achieve random access?
Note that the disk is much large, which means remember the block entry will consumes a lot of valuable memories. Also the external storage should be self-contained

6.5 Indexed Allocation

In indexed allocation, each file has a indexed block, which stores the address of each blocks(link the linked list allocation, but the index is stored in the index store rather than memory).
This solved overhead of memory and can also achieve relative fast random access
However, because the storage of index block is limited, which means there is possibly index block overhead, a feasible solution is to use a index linked list block, each index block contains pointer to the next index block
Another possible solution is to apply multilevel index(high level block stores index of low level index)

6.5.1 Multilevel index

- direct pointers: point to disk block directly
- single indirect block: contains a number of direct pointers
- double indirect block: points to a number of single indirect blocks
- triple indirect block: points to a numbner of double indirect blocks



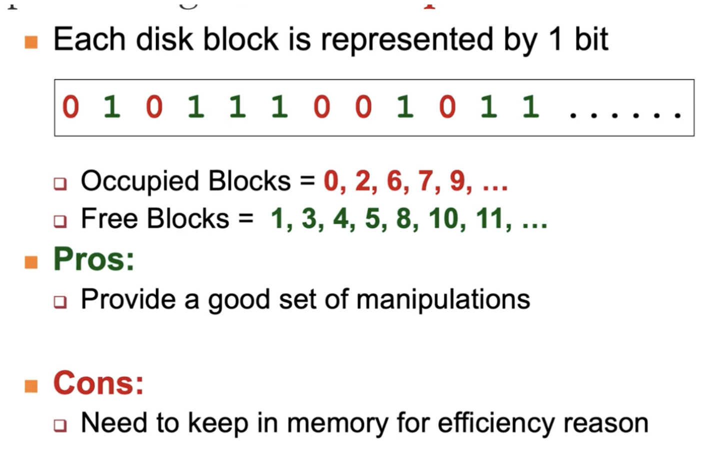
7 Free Space Management

We need free file information for file allocation during

1. File creation/enlargement
2. File deletion/truncation

The free space information can also be kept in both

- a continuous map(with the block number as key and 1 bit value: 1 means free while 0 means not free)(this map should be kept in memory for efficiency)

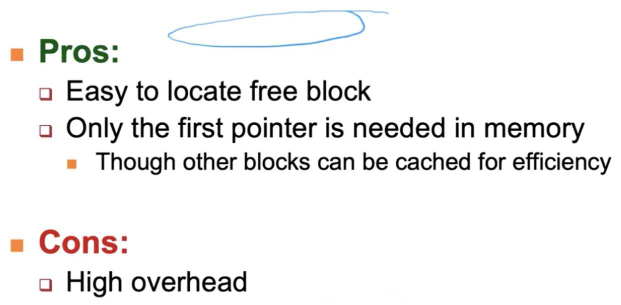


- Each disk block is represented by 1 bit

  0 1 0 1 1 1 0 0 1 0 1 1 . . . . . .

  - Occupied Blocks = **0, 2, 6, 7, 9, …**
  - Free Blocks = **1, 3, 4, 5, 8, 10, 11, …**
- **Pros:**
  - Provide a good set of manipulations
- **Cons:**
  - Need to keep in memory for efficiency reason

- a linked list of free index block or linked list that contains index of free block



- **Pros:**
  - Easy to locate free block
  - Only the first pointer is needed in memory
    - Though other blocks can be cached for efficiency
- **Cons:**
  - High overhead

Because the pages are free, so it's ok to use the memory inside to store the pointer

8 Directory implementation

The directory implementation needs:

1. keep track of files and sub-directories(the subdirectories can be viewed as a special file here) in the directory(possibly with file's metadata)
2. map the filename to the file data

When a file name is given, the system will recursively search along the path until arriving the information location

8.1 Linked List

Directory is a linked list where each entry represents a file, each entry contains

- file name(at least), file meta data
- file data or pointer to file data

Locating a file using list requires a linear search, which is inefficient
A way to resolve this is to use a cache to store latest few searches

8.2 Hashtable directoty

We can use a hash table to hash the file name to the location, each block should contain

- file name, meta-information
- only file name and pointer to the other informations

8.2.1 Pro

fast lookup

8.2.2 Con

1. Limited size
2. rely on good hash function

8.3 Directory to store file information

There are two common approaches:

- Store everything in directory entry
- Store only file name and points to some data structure for other info

9 File system in action

During the runing, the runtime information of the files also matters, which is stored by the OS
There are already some examples(system-wised table and per-process open file table and buffer

10 File Creation

We need a directory to locate the parent directory. We need to avoid duplicate by searching the directory.
Then we can store the relevant information in the home directory such as file name and disk block information which can be used to further locate child information.

10.1 File open

1. We first seach the system-wide file open table for the entry to the file
   (a) If found(say E), we build a new entry P in per-process table pointing to E, and we return a pointer pointing to E
   (b) If not found, we come to the next step
2. We search the full path of the file
   (a) If did not find, we return an error
   (b) If found
      i. We build a new entry E in the system-wised table
      ii. We build a pointer P in the per-process open file table and the E points to P
      iii. We return the pointer to E

The return pointer is used for further read/write to/from the file
This process is in some way like the process of virtual memory(TLB and page fault)

11 Disk I/O Scheduling

The time taken for read/write = seek time+ rotational latency+ transfer time

1. Seek time: the time taken for the head to locate the right track
2. Rotational latency: the time taken for the head to locate to the right sector within the track
3. Transfer time: The time taken to transfer the data

The time taken to transfer the data=$\frac{Transfersize}{Transferrate}$ Usually, the first two are more significant in the length of the time comparing with the transfer time, hence, it is worthwhile for the **OS** to do shceduling.

11.1 Scheduling algorithm of Disk I/O

1. FCFS
2. Shortest Seek First
3. SCAN family, which can be bidirectional or unidirectional