File Systems

OPS Lecture 13, G53OPS/G52OSC

Geert De Maere
(Jason Atkin – OSC)
Geert.DeMaere@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2015

Recap

Last Lecture

- **Traditional hard drives** are still used for most of secondary storage, **solid state disks** are becoming more popular
    - They require less disk scheduling (reads are equally fast, writes are clustered)
    - They must balance wear and tear!
- Traditional hard drives have **physical limitations** (seek times, rotational latency) $\Rightarrow$ **disk scheduling** can help to minimise the impact of this
- This **influences file system design**

Goals for Today

Overview

1. User view of file systems
    - System calls
2. Implementation view of file systems
    - Disk and partition layout
    - File tables
    - Free space management
    - . . .

File Systems

Different Views

- A **user view** that defines a file system in terms of the **abstractions** that the operating system provides
- An **implementation view** that defines the file system in terms of its **low level implementation**

User View

Definition

- Important aspects of the user view include:
  - The **file abstraction** which hides away implementation details to the user (similar to processes and memory)
  - File **naming policies** (abstracts storage details), **extensions**, “user file **attributes**” (e.g. size, protection, owner, protection, dates)
    - There are also **system attributes** for files (e.g. non-human readable file descriptors (similar to a PID), archive flag, temporary flag, etc.)
  - **Directory structures** and organisation
  - **System calls** to interact with the file system
- The **user view** defines what the file system looks like to regular users (and programmers) and relates to **abstractions**

Implementation View

Definition

- Important aspects of the **implementation view** include:
    - **Partitions** and their implementation
    - The **file system implementation** (FAT-16, FAT-32, exFAT, UFS, NTFS, EXT2,3,4, ReFS, etc)
        - Logical to physical **mapping**
    - **Free space** management
    - File system consistency
- The **implementation view** relates to **low level details**, technical aspects, **data structures**, block sizes (influencing internal fragmentation), etc.

Files

Types

- Common **file types** include:
  - **Regular files** contain user data in **ASCII** or **binary** (well defined) **format**
  - **Directories** group files together (but are files on an implementation level)
  - **Character special files** are used to model **serial I/O devices** (e.g. keyboards, printers)
  - **Block special files** are used to model, e.g. hard drives
- Files are **sequential** or **random access**

System Calls
Types

- Two different categories of **system calls** exist:
  - **File manipulation**: open(), close(), read(), write(), . . .
  - **Directory manipulation**: create(), delete(), readdir(), rename(), link(), unlink(), list(), update()
- System calls enable a **user application** to **ask the operating system** to carry out an **action** on its behalf (in kernel mode)
- In practice, a lot more happens underneath!

System Calls

Files

- `Open()` system call:
  - **Search directory** entries for the file location
  - Copy the directory entry to the **open file table** (specify access rights)
  - Increase the **open count**
  - `new Writer/Reader(‘‘D:/.../.../.../file.txt’’)`
- `close()` to free up the entry in the internal table

System Calls

Files (Cont'ed)

- System calls **after opening a file**:
  - `Write()`: write information to the file staring at the position of the current **write pointer**, update **write pointer**
  - `Read()`: read information starting at the current **read pointer**, store the information in the process memory space
- System calls **prior to opening a file**:
  - `Create()`: space is allocated, an **entry is made in a directory**, attributes are initialised by OS
  - `Delete()`: find the file, **remove directory entry** and release the file's space
- Others include `seek`, `append`, `getAttributes`, ...

File System Structures

Overview

- Different directory structures have been used over the years
  - **Single level**: all files in the same directory (reborn in consumer electronics)
  - **Two or multiple level directories** (hierarchical): tree structures
    - Absolute path name: from the root of the file system
    - Relative path name: the current working directory is used as the starting point
  - **Directed acyclic graph (DAG)**: allows files to be shared (i.e. **links** to files or sub-directories) but **cycles are forbidden**
  - **Generic graph structure** in which links and **cycles can exist**
- The use of **DAGs** and **generic graph structures** results in significant **complications** in the implementation

Directories

Possible Implementations

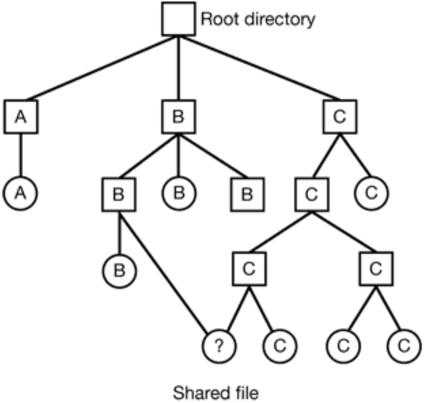


Figure: DAG Directory Implementation (Tanenbaum)

File System Structures

DAG and Graph Complications

- When searching the file system:
    - Cycles can result in **infinite loops**
    - Sub-trees can be **traversed multiple times**
- Files have **multiple absolute file names**
- **Deleting files** becomes a lot more complicated (i.e. links may no longer point to a file, **inaccessible cycles** may exist)
- A **garbage collection scheme** may be required to remove files that are no longer accessible from the file system tree (that are part of a **cycle only**)

Directories

System Calls

- Similar to files, **directories** are manipulated using **system calls**
  - `create/delete`: a new directory is created/deleted
  - `opendir`, `closedir`: add/free directory to/from internal tables
  - `readdir`, return the next entry in the directory file
  - Others: `rename`, `link`, `unlink`, `list`, `update`
- **Directories are special files** that **group files** together and of which the **structure is defined** by the **file system**
  - A bit is set to indicate that they are directories!

Directories

Structure

- Directories contain a list of **human readable file names** that are mapped on **unique identifiers** and **disk locations**
  - They provide a mapping of the logical file onto the physical location
- **Retrieving a file** comes down to using the path to search the directory file and retrieve the exact location of the file, e.g.:
  - E.g. first address and number of blocks
  - I-node number
- They can **store all file related attributes** (e.g. file name, disk address – Windows) or they can **contain a pointer** to the data structure that contains the details of the file (Unix)

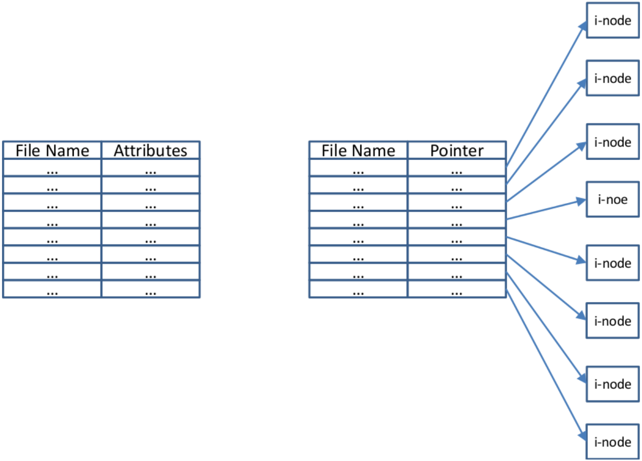
Directories

Internal Structure

- Retrieving a file comes down to **searching a directory file** as fast as possible:
    - A **simple random order of directory** entries might be insufficient (search time is linear as a function of the number of entries)
    - Indexes or **hash tables** can be used
- Commonly accessed files/directories are **cached**

Directories

Possible Implementations



File Name	Attributes
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...

File Name	Pointer
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...

Figure: Directory Implementations

Implementation

Context

- Irrespective of the type of file system, a number of **additional considerations** have to be addressed, including
  - Disk **partitions**, **partition tables**, **boot sectors**, etc.
  - System wide and per process **file tables** (⇒ process tables)
  - Free **space management** (⇒ free memory)
- **Low level formatting** writes sectors to the disk, **high level formatting** imposes a file system on top of this (using **blocks** that can can cover multiple **sectors**)

Hard Disk Structures
Partitions

- Disks are usually divided into **multiple partitions**
  - An independent file system may exist on each partition
- **Master is boot record** located at start of the entire drive:
  - Used to boot the computer (BIOS reads and executes MBR)
  - Contains **partition table** at its end with active partition
  - One partition is listed as **active** containing a boot block to load the operating system

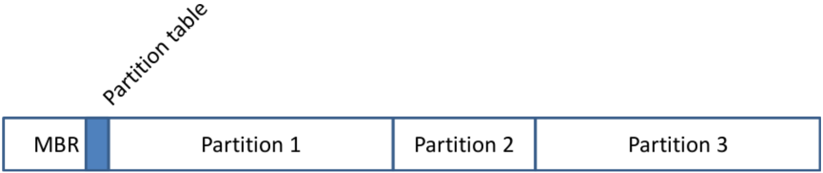


Figure: Layout of a Disk

Partition Layouts

A Unix Partition

- The layout of a partition differs depending on the file system
- A UNIX partition contains:
  - The partition **boot block**:
    - Contains code to boot the operating system
    - Every partition has boot block – even if it does not contain OS
  - **Super block** contains the partition’s details, e.g., partition size, number of blocks, I-node table size
  - **Free space management** contains, e.g., a bitmap or linked list that indicates the free blocks

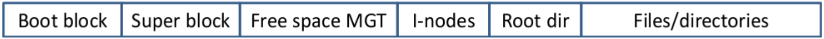
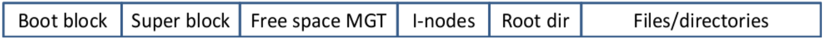


Figure: Layout of a Partition

Partition Layouts

A Unix Partition

- A UNIX partition contains (cont'ed):
    - **I-nodes** contains information on files, commonly maintained in I-nodes
    - **Root directory**
    - **Data**: files and directories



Boot block | Super block | Free space MGT | I-nodes | Root dir | Files/directories

Figure: Layout of a Partition

Disk Space Management

Free Space Management

- Two methods are commonly used to keep track of free disk space: **bitmaps** and **linked lists**
  - Note that these approaches are very similar to the ones to keep track of free memory
- **Bitmaps** represent each block by a single bit in a map
  - The **size of the bitmap** grows with the size of the disk but is constant for a given disk
  - Bitmaps take **comparably less space than linked lists**, unless the disk is nearly full

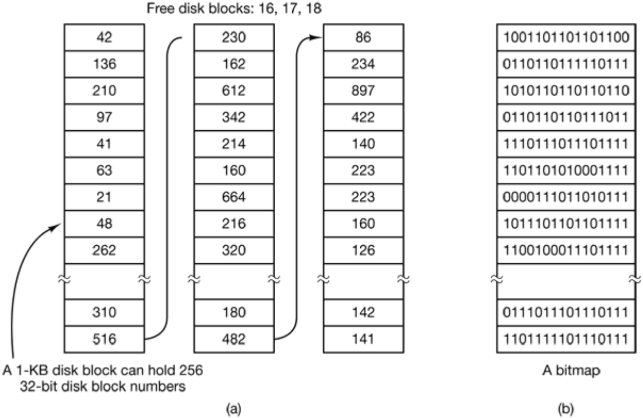
Disk Space Management

Free Space Management

- Linked Lists:
  - Free blocks are used to hold the **numbers of the free blocks** (hence, they are no longer free)
    - Since the free list shrinks when the disk becomes full, this is not wasted space
  - **Blocks are linked together**, i.e., multiple blocks list the free blocks
  - The size of the list **grows with the size of the disk** and **shrinks with the size of the blocks**
- Linked lists can be modified by **keeping track of the number of consecutive free blocks** for each entry

Disk Space Management

Free Space Management


Free disk blocks: 16, 17, 18
A 1-KB disk block can hold 256
32-bit disk block numbers
A bitmap
(a)
(b)

42	230	86	1001101101101100
136	162	234	0110110111110111
210	612	897	1010110110110110
97	342	422	0110110110111011
41	214	140	1110111011101111
63	160	223	1101101010001111
21	664	223	0000111011010111
48	216	160	1011101101101111
262	320	126	1100100011101111
≈	≈	≈	≈
310	180	142	0111011101110111
516	482	141	1101111101110111

Figure: Free Block Management with Linked Lists (Tanenbaum)

File Tables

Implementation

- The OS maintains **two types of file tables**:
  - **System wide file table** contains **process independent information**
  - **Per process open file table** contains **process dependent information**
- **Process independent information** includes location on disk, file size, and “open count” (#processes that use the file)
- **Process dependent information** includes **read/write pointers**, a **reference** to the system wide file table

File Tables

Illustration

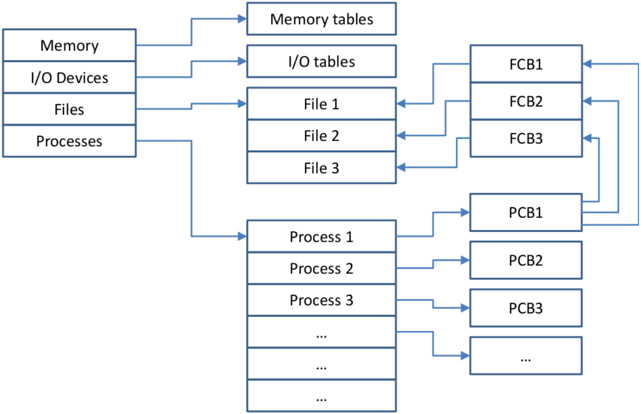


Figure: Illustration of File Tables

File Locking

Types

- Two different **lock types** exist:
    - A **shared lock** allows multiple processes to access a file at the same time
    - An **exclusive lock** allows at most one process to access the file at a time
- Locking enforcement can be:
    - **Mandatory**: no other process can access the file until the exclusive lock has been released, i.e., OS enforces locking (Windows)
        - Hold the lock only as long as necessary
        - Deadlock situations can occur
    - **Advisory locking**: other processes may have access to the file, i.e., OS does not enforce locking – programs themselves are responsible (UNIX)

File System Implementations

How are Files Stored

- File system **implementations**
  1. Contiguous
  2. Linked lists
  3. File Allocation Table (FAT)
  4. I-nodes
- File system **paradigms**, including journalling file systems, log structured file systems, virtual file system

Summary

Take-Home Message

- User vs. implementation view
- Implementation of files and directories
- System calls for file and directory management
- File tables, free space management, partitions, boot sectors, etc.