COMP2007: Operating Systems & Concurrency
Week 10 – 9:00am Wednesday – 29 November 2023

valid for 65 minutes from 8:55am
generated 2023-10-17 03:13

Figure: Attendance Monitoring

Operating Systems and Concurrency

File Systems 3
COMP2007

Geert De Maere
(Dan Marsden)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

Recap

Last Lecture

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

- The **layout of the file system** and the **file allocation method** heavily influence the **seek movements**
  - Contiguous files reduce seek movements
- The **OS can prioritise/order requests** when implementing disk scheduling (vs. controller)

Goals for Today

Overview

- File system **implementations**
  1. Contiguous
  2. Linked lists
  3. File Allocation Table (FAT)
  4. i-nodes (lookups)
- Hard and soft links

- **Files** will be composed of a number of **blocks**
- Files are **sequential** or **random access** (essential for database systems)

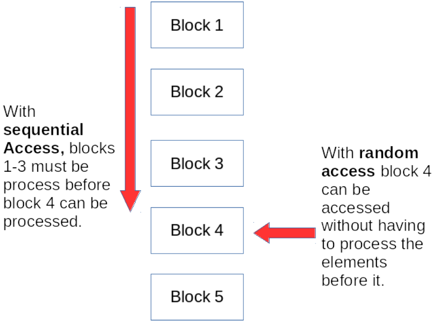

Figure: Types of access to a file

Contiguous Allocation

Concept

- **Contiguous file systems** are similar to **dynamic partitioning** in memory allocation:
  - Each file is stored in a single group of **adjacent blocks** on the hard disk
  - E.g. 1KB blocks, 100KB file, we need 100 contiguous blocks
- Allocation of free space can be done using **first fit**, **best fit**, **next fit**, etc.

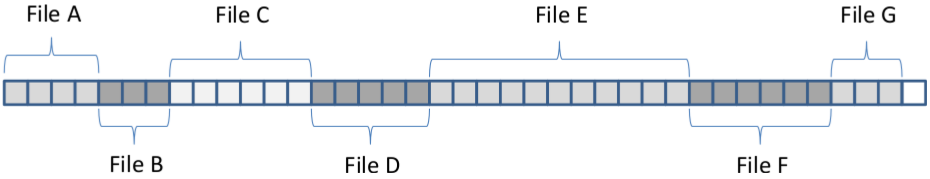

Figure: External Fragmentation when Removing Files

Contiguous Allocation

Advantages

- **Easy to implement**: only location of the first block and the length of the file must be stored (in the FCB)
- **Optimal read/write performance**: blocks are co-located/clustered in nearby/adjacent sectors (seek time is minimised)

File	**Start**	**Length**
req1.c	0	12
req1.o	30	10
req1	15	5
req1.txt	41	20

Figure: Directory table

Contiguous Allocation

Disadvantages

- **Disadvantages** of contiguous file systems include:
  - The **exact size** of a file (process) is not always known beforehand
  - **Allocation algorithms** are needed to decide which free blocks to allocate (e.g., first fit, best fit)
  - Deleting a file results in **external fragmentation** which requires **de-fragmentation** (which is **slow**)
- Contiguous allocation used for **CD-ROMS/DVDs**
  - External fragmentation is less of an issue since they are **write once**

Linked Lists

Concept

- To avoid external fragmentation, files are stored in **separate blocks** (similar to paging) that are **linked to one another**
- Only the **address of the first** is stored in the FCB
- Each block contains a **data pointer** to the next block (which takes up space)

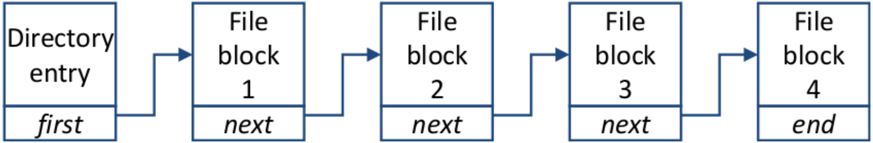

Figure: Linked List File Storage

Linked Lists

Advantages

- **Easy to maintain**: only the first block (address) has to be maintained in the directory entry
- Files can **grow dynamically**: new blocks/sectors are appended at the end
- Similar to paging, there is **no external fragmentation**
- **Sequential access** is straightforward, although **more seek operations** may be required (non-contiguous)

Linked Lists

Disadvantages

- **Random access** is very **slow**
- **Internal fragmentation**: the last half of the block is unused (on average)
  - Internal fragmentation will reduce for **smaller block sizes**
- May result in **random (slow) disk access**
  - **Larger blocks** (containing multiple sectors) will improve speed (but increase internal fragmentation)

Linked Lists

Disadvantages (Cont'ed)

- The data within a block is **no longer a power of 2**
- **Reduced reliability**: if one block is corrupt/lost, access to the rest of the file is lost

File Allocation Tables

Key Concept

- Store the linked-list pointers in a **separate index table**, called a **File Allocation Table (FAT)** (loaded in memory)

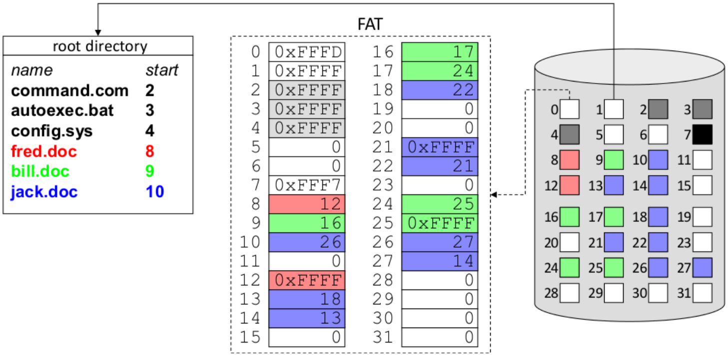

Figure: File Allocation Tables

File Allocation Tables

Advantages and disadvantages

- Advantages:
  - **Block size remains power of 2** (no space is lost in block due to the pointer)
  - **Index table** can be **kept in memory** allowing fast non-sequential/random access (table access itself remains sequential)
- Disadvantages:
  - The **size** of the **file allocation table** grows with the number of blocks/disk size
  - **200 million entries** are required for a **200GB disk** with a **1KB block size** (**800MB** at 4 bytes per entry)

i-nodes

Concept

- Each file has a small data structure (on disk) called **i-node** (index-node) that contains its attributes and block pointers.
    - In contrast to FAT, an i-node is **only loaded when the file is open** (stored in system wide open file table)
    - If every i-node consists of n bytes, and at most k files can be open at any point in time, at most $n \times k$ bytes of main memory are required
- i-nodes are composed of **direct block pointers** (usually 10), **indirect block pointers**, or a combination thereof (e.g., similar to **multi-level page tables**)

i-nodes

Concept

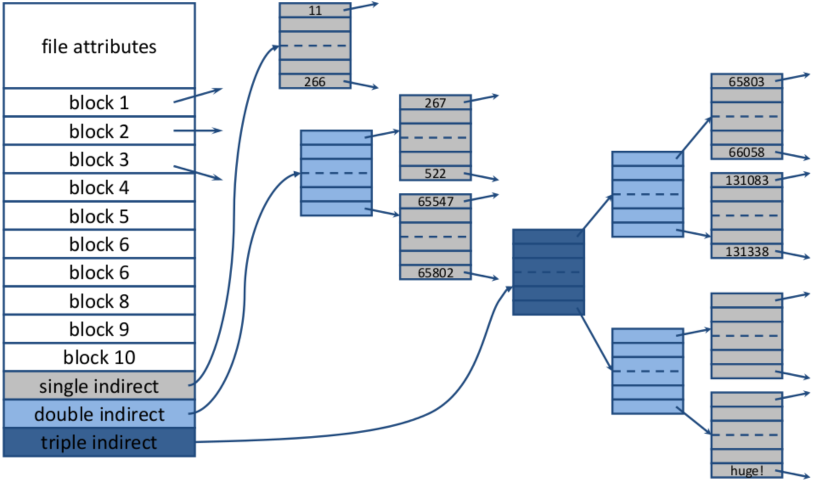

Figure: File Storage Using I-Nodes

i-nodes

Concept

- Assuming a block size of 1KB, and 32-bits disk address space
- With **only direct block pointers** the maximum file size is 10KB (1KB $\times$ NUMBER_OF_DIRECT_BLOCK_POINTERS)
- With a **single indirect block** the maximum file size is $(10+256) \times 1KB = 266KB$
- With a **double indirect block** 256 blocks containing 256 pointers each
  - The maximum file size is $(10+256+256^2) \times 1KB = 65802KB$
- If we need files larger than this, we will need a triple indirect blocks for a maximum file size of $(10+256+256^2+256^3) \times 1KB$

Directories

Implementation with i-nodes

- In UNIX/Linux/MacOS:
  - All **metadata** for a file (type, size, date, owner, and block pointers) is stored in an **i-node**
  - **Directories** are very simple data structures composed of file name and a pointer to the i-node
- Directories are no more than a special kind of **file**, so they have their own i-node

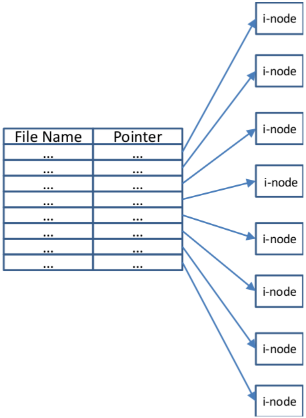

Figure: i-node Directory Structure

File System Comparison

Contiguous vs. Linked vs. Indexed

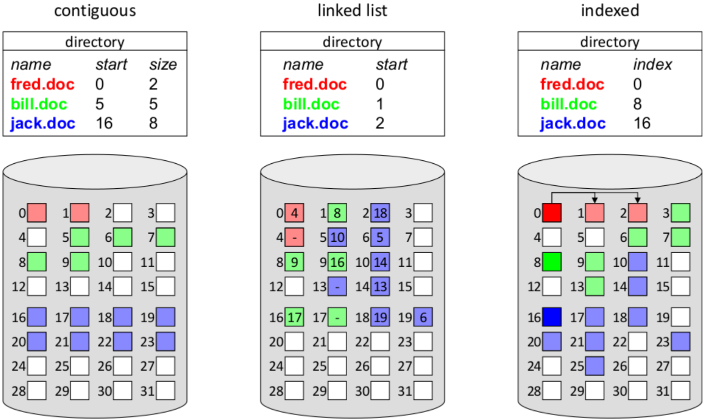

Figure: Contiguous vs. Linked List (or FAT) vs. i-nodes (or indexed)

i-nodes

Lookups

- **Opening a file** requires the disk blocks to be located
  - **Absolute file names** are located relative to the root directory
  - **Relative file names** are located based on the current working directory
- E.g. Try to locate `/usr/gdm/mbox`

/ inode 1

size mode times
2

/ contents 2

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

/usr inode 6

size mode times
132

/usr contents 132

6	.
1	..
19	fred
30	bill
51	jack
26	gdm
45	cfi

/usr/gdm inode 26

size mode times
406

/usr/gdm contents 406

26	.
6	..
64	research
92	teaching
60	mbox
17	grants

Figure: Locating a File

i-nodes

Lookups

- Locate the root directory of the file system
  - Its i-node sits on a fixed location at the disk (the directory itself can sit anywhere)
- Locate the directory entries specified in the path:
  - Locate the i-node number for the first component (directory) of the path that is provided
  - Use the i-node number to index the i-node table and retrieve the directory file
  - Look up the remaining path directories by repeating the two steps above
- Once the file's directories have been located, locate the file's i-node and cache it into memory

i-nodes: Sharing files between directories

Hard and Soft Links

- There are two approaches to **share a file**, e.g. between directory B and C, where C is the 'real' owner:
  - **Hard links**: maintain two (or multiple) references to the same i-node in B and C
    - the i-node link reference counter will be set to 2
  - **Symbolic links**:
    - The owner maintains a reference to the i-node in, e.g., directory C
    - The "referencer" maintains a small file (that has its own i-node) that contains the location and name of the shared file in directory C
- What is the **best approach**? $\Rightarrow$ both have advantages and disadvantages

i-nodes: Sharing files between directories

Hard Links

- Hard links are the **fastest way of linking files**!
- Disadvantages of hard links:
  - Assume that the owner of the file **deletes** it:
    - If the i-node is also deleted, any hard link will, in the best case, **point to an invalid i-node**
    - If the i-node gets **deleted** and “**recycled**” to point to an other file, the hard links will **point to the wrong file**!
  - The only solution is to **delete the file**, and **leave the i-node intact** if the “**reference count**” is larger than 0 (the original owner of the file still gets “charged” for the space)

i-nodes: Sharing files between directories

Hard Links

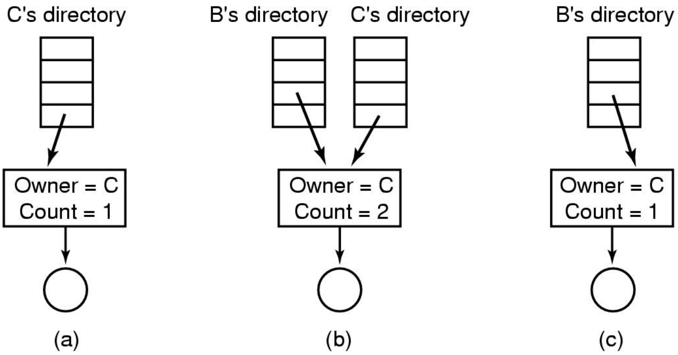

Figure: (a) Before creating a link. (b) After creating a link. (c) after the orignal owner removes the file (Tanenbaum)

i-nodes

Soft Links

- Disadvantages of soft links:
  - They result in an **extra file lookup** (once the link file has been found, the original file needs to be found as well)
  - They require an **extra i-node** for the link file
- Advantages of symbolic links:
  - There are **no problems with deleting** the original file $\Rightarrow$ the file simply does not exist any more
  - They can **cross the boundaries of machines**, i.e. the linked file can be located on a different machine

i-nodes

Hard and Soft Links in Unix

```
[pszgd@severn ~]$ pwd
/home/pszgd
[pszgd@severn ~]$ ls -i labs/
3250013 req1b.c
[pszgd@severn ~]$ ln labs/req1b.c hardLink
[pszgd@severn ~]$ ln -s labs/req1b.c softLink
[pszgd@severn ~]$ ls -ali hardLink softLink
3250013 hardLink
3250021 softLink -> labs/req1b.c
[pszgd@severn ~]$ rm labs/req1b.c
```

File System Examples

Unix vs. Windows

- The Unix V7 File System:
  - **Tree structured** file system with links
  - Directories contain **file names** and **i-node numbers**
  - i-nodes contain **user and system attributes** (e.g. count variable)
  - One single, double, and triple **indirect blocks** can be used
- More sophisticated File Systems were developed later (e.g. ext3/4)
- Windows:
  - Up to XP used FAT-16 and FAT-32
  - From XP moved to NTFS (64 bits) because of file size limitations
    - NTFS uses File Tables, with bigger i-nodes that can also contain small files and directories
  - More recently uses ReFS

Recap

Take-Home Message

- Contiguous, linked list, FAT and i-nodes as file system implementations.
- Lookups with i-nodes.
- Hard and Soft Links

Test Your Understanding

File Systems

Exercises

With i-nodes, the maximum file size that we can have depends on the block size and the number of indirections.

- Assuming a 32-bit disk address space, what is the maximum (theoretical) file size for a FAT file system with a drive of 500GB and a block size of 1KB? (without accounting for directory metadata)
- The most used implementation of FAT is known as FAT-32. Investigate why there is a theoretical limitation of 4GB per file (and sometimes even less than 2GB).