Chapter 4.3

File System's Physical Data Layer

Implementation, Fragmentation, Links and Backup

Print Version of Lectures Notes of *Operating Systems*

Technical University of Cluj-Napoca (UTCN)
Computer Science Department

Adrian Colesa

March 25, 2020

4.3.1

Purpose and Contents

The purpose of today's lecture

- Presents details about the way files and directories are implemented.
- Presents related strategies and problems like: fragmentation, links, backup.

4.3.2

Bibliography

- A. Tanenbaum, *Modern Operating Systems*, 2nd Edition, 2001, Chapter 6, pg. 399 – 421

4.3.3

Contents

4.3.4

1 File's Data Allocation

Context

- files are
  - **provided to user application as sequences at bytes**
    - a logical view
    - a contiguous area
  - **allocated in terms of blocks**, i.e. group of bytes, on HDD
    - a physical view
    - not necessarily a contiguous area
- we are interested in
  - how blocks of a file are allocated on HDD
  - how does the allocation strategy influence the user applications

4.3.5

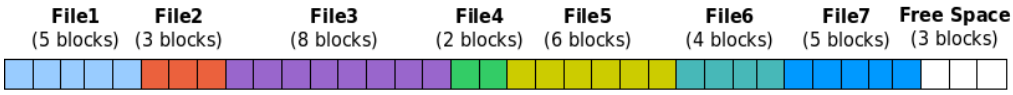


Figure 1: HDD Partition Structure

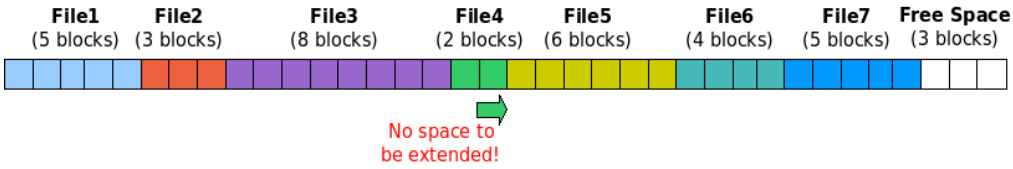


Figure 2: HDD Partition Structure

Contiguous Allocation

- Files are allocated in **just one contiguous area**
- Advantages
  - Reading large areas from the file is very fast
  - Keeping track of allocated blocks (BAT) is very simple: starting block and the number of allocated blocks
- Disadvantages
  - Difficult to increase the file size: see for example **File 4**
  - Leads to **external fragmentation**
  - Complex allocation strategies: **first fit**, **best fit** etc.

4.3.6

Any-Free-Block Allocation

- The file can be allocated any free block
- Advantages
  - there is no external fragmentation; any free block can be used
  - file size can be easily extended
  - could be combined with contiguous allocation: the file is allocated more contiguous areas (as large as possible)
- Disadvantages
  - data access (e.g. read entire file) not so efficient
  - BAT structure more complicated
  - still suffers from **internal fragmentation** and **data fragmentation**

4.3.7

Fragmentation Types: External Fragmentation

- context
  - **free space scattered in small areas** over the entire HDD
  - alternating with allocated areas

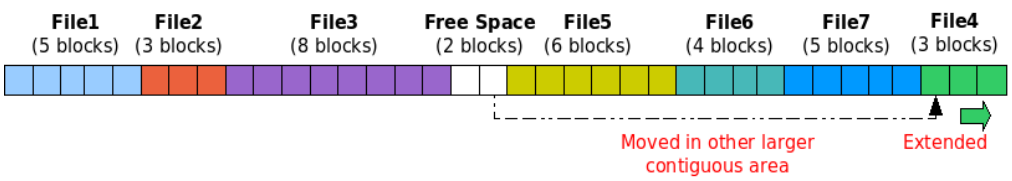


Figure 3: HDD Partition Structure

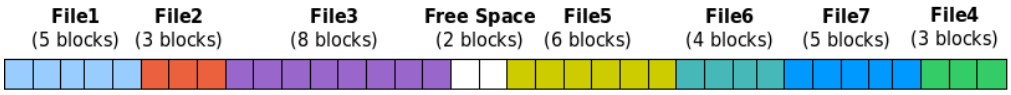


Figure 4: HDD Partition Structure

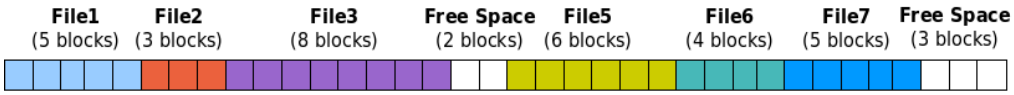


Figure 5: HDD Partition Structure

- problem
  - **free space cannot be used** (in some situations)
  - example
    - need to allocate a contiguous area of a give size
    - but no free contiguous area could be available
    - even if total free space would be enough
- specific to
  - contiguous allocation strategies
  - where data can be allocated **only in a single contiguous area**
- solution (inefficient, i.e. time consuming)
  - **defragmentation**: move all allocated space at one end of the HDD
  - $\Rightarrow$ free space in a single contiguous area at the other end of HDD

4.3.8

Fragmentation Types: Internal Fragmentation

- context
  - any free block could be allocated when new space needed
  - BUT . . . **allocation is done in terms of predefined units**, i.e. blocks
  - AND . . . **needed space not a multiple of block size**
- problem
  - some (internal) “free” space cannot be used
  - **unused space in blocks allocated to files** is lost
- specific to
  - allocation strategies that allocate data in terms of fixed-size blocks
- solutions
  - **smaller block size** to reduce internal fragmentation
  - **fragment blocks**, i.e. share the same block for tails of multiple files

4.3.9

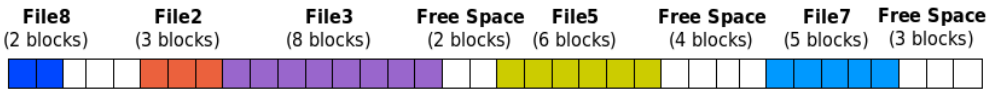


Figure 6: HDD Partition Structure

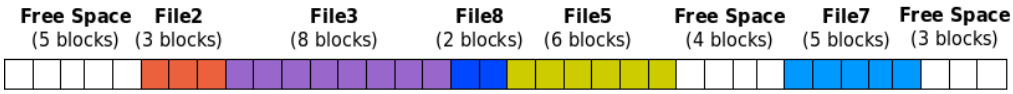


Figure 7: HDD Partition Structure

Fragmentation Types: Data Fragmentation

- context
  - the allocated space of a file is distributed in different non-consecutive blocks
  - i.e. more contiguous areas
- problem
  - **file access could suffer performance penalties**
- specific to
  - **any-free-block allocation strategy**
  - that do not impose the file to be in a single contiguous area
- solution
  - **defragmentation**: reallocate file's blocks in consecutive ones

4.3.10

File System Block Size

- The problem: How large the block should be?
- Alternatives
  - Larger
    - efficient read of file data
    - increase internal fragmentation (waste HDD space)
  - Smaller
    - reduce internal fragmentation
    - increase data fragmentation and data access time (many disk accesses)
- **no good-for-all solution**
  - *performance* and *space utilization* are inherently in conflict
  - the block size should be chosen knowing the way and for what the HDD partition will be used
  - a compromise should be chosen in a general usage case

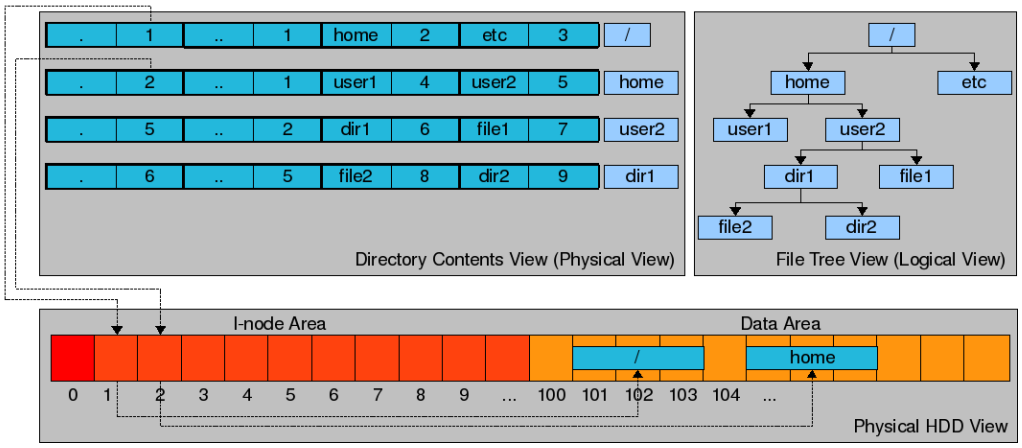
4.3.11

2 Directory Implementation

Directory Contents

- a system "file"
- stored as a stream of bytes, but interpreted by the OS
- organized as a collection of records (elements), called **directory entries**
- a directory entry contains
  - the (file, directory etc.) name
  - file's metadata or a reference to them

4.3.12



File "I-node" (Record)

- a physical space (element) and a corresponding data structure used to store information about a FS element (file, directory etc.)
- stores file meta-data, like
  - file type
  - size
  - owner
  - permission rights
  - time stamps
  - the BAT (Block Addresses Table)
  - etc.

4.3.13

The Relationship Between The Directory Entry and The I-node: Illustration

4.3.14

3 Hard and Symbolic Links

Sharing Data Between Directories

- make a file (directory) appear in different directories
- the operation is called linking files
- two kinds of links
  - hard (physical)
  - soft (symbolic)

4.3.15

Hard Link: Creation and Usage

```
create("/home/os/dir1/f1", 0600);    // only allocate i-node; no data
link("/home/os/dir1/f1", "/home/os/dir2/f2"); // hard link
link("/home/os/dir1/f1", "/home/os/dir2/f1"); // hard link
open("/home/os/dir1/f1", ...);  // open file with i-node 5
open("/home/os/dir2/f2", ...);  // open file with i-node 5
open("/home/os/dir2/f1", ...);  // open file with i-node 5
stat("/home/os/dir1/f1", ...);  // read i-node 5 contents
stat("/home/os/dir2/f1", ...);  // read i-node 5 contents
stat("/home/os/dir2/f2", ...);  // read i-node 5 contents
```

4.3.16

Hard Link: Illustration

4.3.17

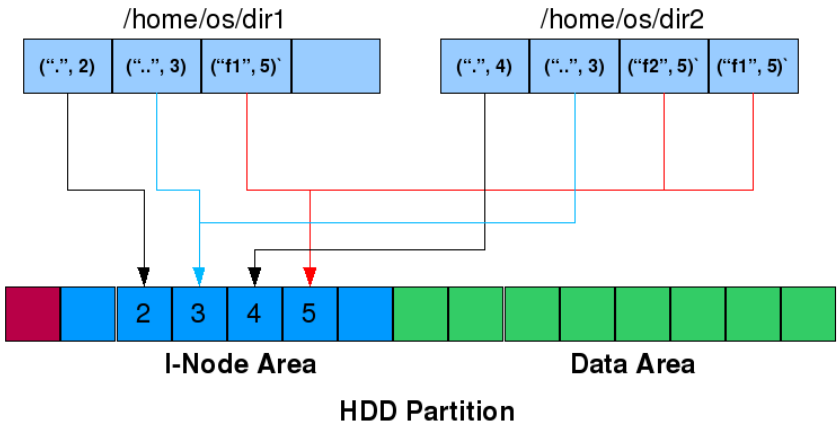


Figure 8: Hard Link Implementation

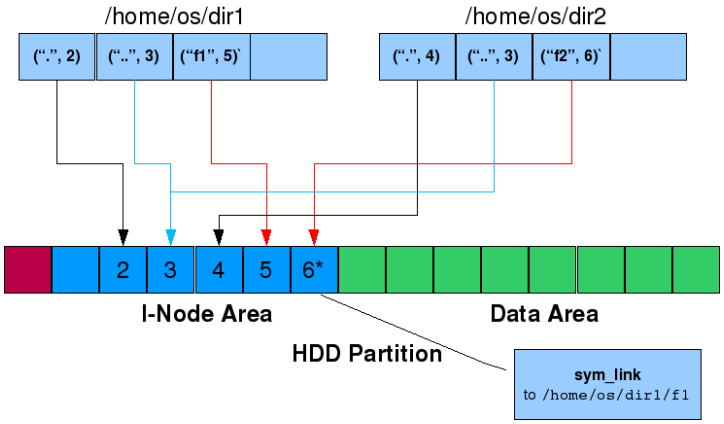


Figure 9: Symbolic Link Implementation

Hard Link: Discussion

- Advantages
  - a file really belong to the two or more directories, when a path is removed the physical file (space) is not removed until all hard links are removed
  - very transparent; there is no difference and distinction between different hard links to the same file
- Disadvantages
  - cannot be established between different partitions

4.3.18

Symbolic Link: Creation and Usage

```
create("/home/os/dir1/f1", 0600);    // only allocate i-node; no data
symlink("/home/os/dir1/f1", "/home/os/dir2/f2"); // hard link
open("/home/os/dir1/f1", ...);  // open file with i-node 5
open("/home/os/dir2/f2", ...);  // open file with i-node 5
stat("/home/os/dir1/f1", ...);  // read i-node 5 contents
stat("/home/os/dir2/f2", ...);  // read i-node 5 contents
lstat("/home/os/dir2/f2", ...);  // read i-node 6 contents
```

4.3.19

Symbolic Link: Illustration

4.3.20

Symbolic Link: Discussion

- Advantages
  - can be created between different partitions
- Disadvantages
  - once the referenced path is removed, the link become invalid

4.3.21