Assignment Project Exam Help

Operating Systems and Concurrency
Lecture 21: File Systems IV
G52OSC

https://powcoder.com

Add WeChat powcoder
Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

- **Contiguous implementations** are easy, fast, but result in external fragmentation
- **Linked lists** are sequential and have block sizes $\neq 2^n$ (page sizes are 2^n)
- **FAT** have block sizes $= 2^n$, but the table becomes prohibitively large
- **I-nodes** are only loaded when the file is open, contain attributes and multiple block levels

Assignment Project Exam Help

- Hard and soft links with i-nodes
- File system **paradigms** (which use an underlying file system)
  - Log structured file systems
  - Journaling file systems
  - The Unix/Linux virtual file system

https://powcoder.com

Add WeChat powcoder

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. (In Unix: ln /C/file1 /B/file2)
        - 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. (In Unix: ln -s /C/file1 /B/file2)

- 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)

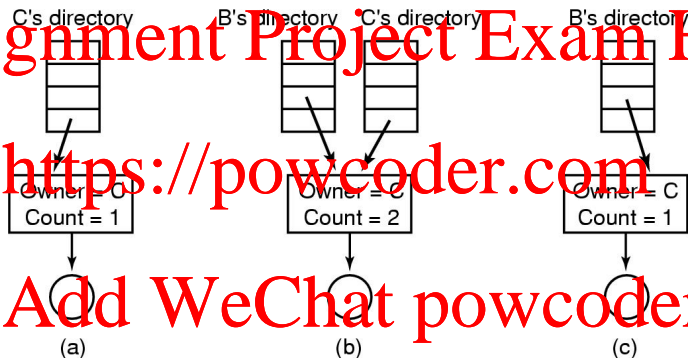# I-nodes: Sharing files between directories
Hard and Soft Links

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

I-nodes
Soft Links

Assignment Project Exam Help

- 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

```
[pszit@luthien ~]$ pwd
/home/pszit
[pszit@luthien ~]$ ls -i labs/
3250013 req1b.c

[pszit@luthien ~]$ ln labs/req1b.c hardLink

[pszit@luthien ~]$ ln -s labs/req1b.c softLink

[pszit@luthien ~]$ ls -ali hardLink softLink
3250013 hardLink
3250021 softLink -> labs/req1b.c

[pszit@luthien ~]$ rm labs/req1b.c
```

Log Structured File System
Context

- Consider the **creation of a new file** on a Unix system:
  1. Allocate, initialise and write the i-node for the file
     - i-nodes are usually located at the start of the disk
  2. Update and write the directory entry for the file (directories are tables/files that map names onto i-nodes in Unix)
  3. Write the data to the disk

- The corresponding blocks are not necessarily in **adjacent locations**!
- Also in linked lists/FAT file systems blocks can be **distributed all over the disk**

Log Structured File System
Context

- Due to seek and rotational delays, **hard disks are slow** compared to other components in a computer (e.g. CPU, main memory)
  - Can we develop a file system that copes better with the inherent delays of traditional disks?
- A **log structured file system** aims to improve speed of a file system on a traditional hard disk by **minimising head movements** and **rotational delays** using the entire disk as a great big log
- A log is a data structure that is written only at the end

Log Structured File System
Concept

- Log structured file systems **buffer read and write operations** (i-nodes, data, etc.) in memory, enabling us to **write "larger volumes"** in one go
- Once the buffer is full it is "flushed" to the disk and written as **one contiguous segment** at the end of "a log"
  - I-nodes and data are all written to the same segment
  - Finding i-nodes (traditionally located at the start of the partition) becomes more difficult
- An **i-node map** is maintained in memory to quickly find the address of i-nodes on the disk
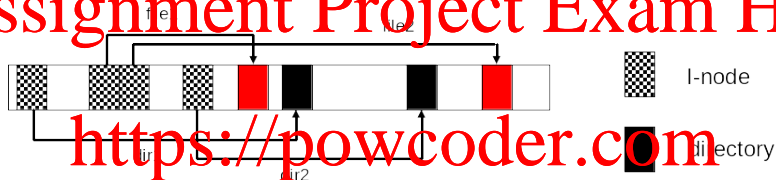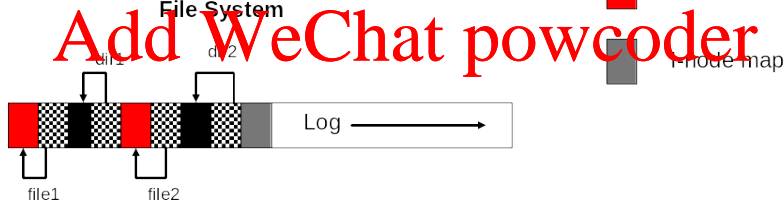
Log Structured File System
Structure

Figure: Blocks written to create two 1-block files: dir1/file1 and dir2/file2

Log Structured File System
Concept

- How do we deal with deleted files?
- A **cleaner thread** is running in the background and spends its time scanning the log circularly and compacting it.
- A hard drive is treated as a **circular buffer**
- It removes deleted files and files being used right now are marked as free segments as they will be later written at the end.

Figure: Log Structured File System

Log Structured File System
Advantages and Disadvantages

- It greatly **increases disk performance** on writes, file creates, deletes
- Writes are **more robust** as they are done as a single operation (Multiple small writes are more likely to expose the file system to serious inconsistency).
- However, it has not been widely used because it **is highly incompatible** with existing file systems.
- In addition, the cleaner thread takes additional CPU time

File System Implementations
Journaling File Systems: Example

Assignment Project Exam Help

- **Deleting a file** consists of the following actions:
  1. Remove the file's directory entry
  2. Add the file's i-node to the pool of free i-nodes
  3. Add the file's disk blocks to the free list

https://powcoder.com

- Where can it **go wrong**?, e.g.:
  - Directory entry has been deleted and a crash occurs $\Rightarrow$ i-nodes and disk blocks become inaccessible
  - Add WeChat powcoder
  - The directory entry and i-nodes have been released and a crash occurs $\Rightarrow$ disk blocks become inaccessible

File System Implementations
Journaling File Systems: Example

Assignment Project Exam Help

- **Changing the order** of the events does not necessarily resolve the issues

https://powcoder.com

- Journaling file systems aim at **increasing the resilience** of file systems against **crashes** by recording each update to the file system as a transaction

Add WeChat powcoder

File System Implementations
Journaling File Systems: Concept

- The key idea behind a journaling file system is to **log all events** (transactions) before they take place
  - Write the actions that should be undertaken to a log file
  - Carry them out
  - Remove/commit the entries once completed
- If a crash happens in the middle of an action (e.g., deleting a file) the **entry in the log file will remain present after the crash**
- The log can be **examined after the crash** and used to restore the consistency of the file system
- **NTFS** and **EXT3-4** are examples of journaling file systems

File System Implementations
Virtual File Systems: Concept

- **Multiple file systems** usually coexist on the same computer (e.g., NTFS and ISO9660 for a CD-ROM, NFS)
- These file systems can be **seamlessly integrated** by the operating system (e.g. Unix / Linux)
- This is usually achieved by using **virtual file systems** (VFS)
- VFS relies on **standard object oriented** principles (or manual implementations thereof), e.g. polymorphism

File System Implementations
Virtual File Systems: Concept

Assignment Project Exam Help

- Consider some code that you are writing, **reading "data records"** (DataObject) from a file

https://powcoder.com

- These records can be stored in **CSV file**, or **XML File**
- How would you make your code resilient against **changes in the underlying data structure**?

Add WeChat powcoder

File System Implementations
Virtual File Systems: Concept

- Consider some code that you are writing, **reading "data records"** (DataObject) from a file
- These records can be stored in **CSV file**, or **XML File**
- How would you make your code resilient against **changes in the underlying data structure**?
  - You would hide the **implementation** behind **interfaces** using **Data Access Objects** (DAOs)

File System Implementations
Virtual File Systems: Concept

- We can define a **generic interface**, e.g. `DataReader`, containing a method `public DataObject readData();`
  - In the case of file systems, this would be the **POSIX interface** containing reads, writes, close etc.
- You would hide the CSV and XML code in **specific implementations** of the `DataReader` interface. e.g. `CSVDataReader` and `XMLDataReader`
  - In the case of file systems this would be the file system implementations

File System Implementations
Virtual File Systems: Concept (Cont'ed)

- You would rely on **polymorphism** to call the correct method
  - DataReader dr = new CSVDataReader()
    dr.readData() // reads data from CSV
  - DataReader dr = new XMLDataReader()
    dr.readData() // reads data from XML

File System Implementations
Virtual File Systems: Concept

```java
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4
5  public class ClientApplication {
6    public static void main(String[] args) throws IOException {
7      System.out.println("Choose CSV or XML?");
8      BufferedReader br
9        = new BufferedReader(new InputStreamReader(System.in));
10     String type = br.readLine();
11     br.close();
12
13     DataReader reader = null;
14     if (type.equals("CSV")) {
15       reader = new CSVDataReader();
16     } else if (type.equals("XML")) {
17       reader = new XMLDataReader();
18     }
19
20     if (reader != null)
21       System.out.println(reader.readData());
22   }
23 }
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

File System Implementations
Virtual File Systems: Concept

- In a similar way, Unix and Linux **unify different file systems** and present them as a single hierarchy and hides away / abstracts the implementation specific details for the user.
- The VFS presents a unified interface to the "outside"
- File system specific code is dealt with in an **implementation layer** that is **clearly separated from the interface**

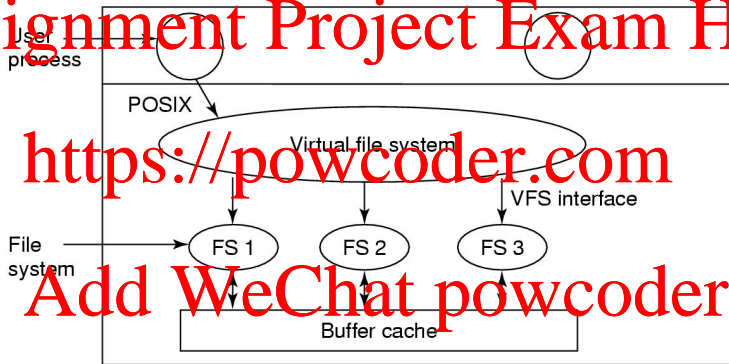# File System Implementations
Virtual File Systems: Concept

Figure: Virtual File System (Tanenbaum)

File System Implementations
Virtual File Systems: Concept

- The VFS interface commonly contains the **POSIX system calls** (`open`, `close`, `read`, `write`, ...)
- Each file system that meets the VFS requirements **provides an implementation** for the system calls contained in the interface
- Note that implementations can be for **remote file systems** (e.g. sshfs), i.e. the file can be stored on a different machine

File System Implementations
Virtual File Systems: In practice

- Every file system, including the root file system, is **registered with the VFS**
    - A list / table of addresses to the **VFS function calls** (i.e. function pointers) for the specific file system is provided
    - **Every VFS function call** corresponds to a specific **entry in the VFS function table** for the given file system
        - The **VFS maps / translates** the POSIX call onto the "native file system call"
- A virtual file system is essentially **good programming practice**

File System Implementations
Virtual File Systems: Example

```
$ df -Th
Filesystem      Type        Size   Used  Avail  Use%  Mounted on
devtmpfs        devtmpfs     32G      0    32G    0%  /dev
/dev/sda3       btrfs       215G    82G   132G   39%  /
tmpfs           tmpfs        32G   8.6M    32G    1%  /tmp
/dev/sda3       btrfs       215G    82G   132G   39%  /home
/dev/sda1       ext4        477M   166M   282M   38%  /boot
/dev/sdb        ext4        917G   165G   706G   19%  /data
pszit@bann:     fuse.sshfs  700M   282M   418M   41%  /mnt/bann
```

Summary
Take-Home Message[1]

Assignment Project Exam Help

https://powcoder.com

- Hard and Soft Links
- File system **paradigms:**
  - Logs: store everything as close as possible
  - Journaling: apply the transaction principle
  - VFS: apply good software design (polymorphism)

Add WeChat powcoder

---

[1]Tanenbaum Chapter 4, Sections 4.3.5, 4.3.6, 4.3.7

File systems
Food for thought

Following up the previous question on FAT-32, we mentioned that FAT-32 has severe limitations (e.g. the file allocation table could be too big).

- Why do you think that this file system is still in use in most of flash drives, cameras or MP3 players?
- When would you consider formatting your flash drive with NTFS or ext3?
- If you format your flash drive with (Windows) NTFS, will it work (directly) in Unix systems?

Submit your answer at:

`https://b.socrative.com/login/student/`

Room name: G52OSC