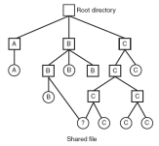
Preview

- Shared File in multiuser system
  - Save i-node index
  - Symbolic link
- Log-Structured File System (extension of i-node + contiguous)
- Journaling File System
- Disk Space Management
  - Block size
  - Free block management
    - Linked List
    - Bit Map

Shared Files

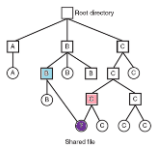
- To share a file, it is convenient to appear simultaneously in different directories belongs to different users.



- Assume user B and C are same group member working on same project.
- User C create a file which is shared with user B.
- Assume a file information is saved in two directories

Shared Files

- Shared file idea is convenient but it produce problems when a shared user modifies the shared file.



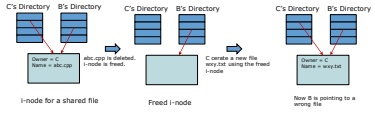
• Lets assume a shared file (purple) shared with user B and user C.
• Then file information must be saved in both directory B and directory C.
• If user B modify shared file, only the shared file information in directory B will be modified.

Shared Files

- There are two solutions
  - Disk block information for a file are saved not in the directory but in the i-node.
  - A system create a symbolic link to a shared file.
    - **The symbolic link is a file** which contains the path name of the shared file.
- Both solutions still have problems

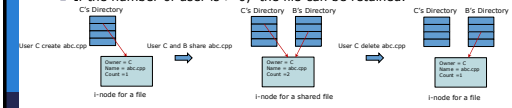
Shared Files
(with i-node)

- Lets assume user C create a file and it is shred with user B.
- i-node number must be saved in both directory.
- If user C remove the shared file then i-node used for shared file can be used for other new file.
  - Ex) If a user create a new file and system assign the same i-node for new file, B is pointing to a wrong file.



Shared Files
(with i-node)

- Solution
  - System check the number of user for a file in i-node
  - If the number of user is > 0, the file can be retained.

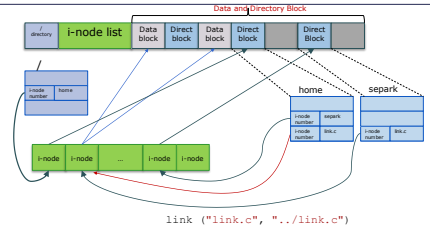


- Still there is a problem.
  - Even though C remove a shared file (from previous example), that space is count as C's space since it is owned by C.
  - It is used for counting the quota for C.

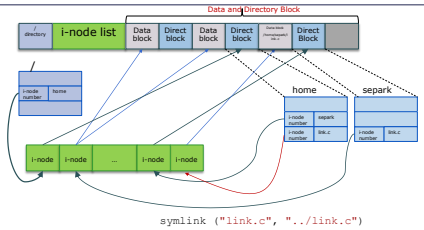
Shared Files
(with Symbolic Link)

- With symbolic link, the same problem with i-node does not arise since only real owner has pointer to the i-node.
- With Symbolic Link, extra overhead is required
  - To read or write a file
    1. Get a path
    2. Parse the path from the root
  - Extra block for a i-node is required for each Symbolic link to store only path – wasting disk space!

The `link()` System Call



The `symlink()` System Call



Log-structured file systems

- CPU speeds have increased dramatically while disk access times (seek time) have only improved slowly.
- This will cause more and more applications to become disk-bound.
- Log-Structured file system is designed to reduce the impact of this problem.
- (This idea is for HDD but this idea is used in SSD!)

Log-structured file systems

- Log-structured file systems are **based on the assumption that files are cached in main memory.**
- As a result, disk traffic will become dominated by writes.
- A log-structured file system writes all new information to disk in a sequential structure called **the log**.(contiguous blocks)
- This approach increases write performance speed dramatically by eliminating almost all seeks.

Log-structured file systems

- The sequential nature of the log also permits much faster crash recovery:
  - **UNIX** : scan the entire disk to restore consistency after a crash.
  - **LSF** (Log-Structured File): examine the most recent portion of the log.
- Some file system use a log as a temporary storage to speed up write and crash recovery.
- **LSF** system stores data permanently in the log: there is no other structure on disk.

Log-structured file systems

- For a log-structured file system to operate efficiently, it must ensure that there are always large extents of free space available for writing new data.
- A **segment cleaner thread** continually regenerates empty segments.
  - Start out by reading the summary of the first segment.
  - Check i-node map to find out it is currently used segment.
  - If not, it set as a available segment for write operation

Log-Structured File System

- The basic idea is to **structure the entire disk as a log**.
- All writes are initially buffered in memory.
- Periodically, all buffered writes are written to the disk in a single segment at the end of log.
- A single segment contains i-nodes, directory blocks, and data blocks, all mixed together

Log-Structured File System

- In Unix file system, each i-node is at a fixed location on disk.
- In LSF, each i-node is not at a fixed location; they are written to the log.
- LFS uses a data structure called an **i-node map** to maintain the current location of each i-node for each file.
- Opening a file consists of using the map to locate the i-node for the file.

Log-Structured File System

- Once a existing file is open for update, a segment of the file is load to the memory (file cache).
- It always write back to the disk at the end of log.
- Which means that updated file will not copy back to the previous location.

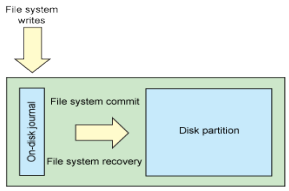
Journaling File System

- When a system using non-journaling file system is improperly shut down, the operating system detects this and performs a consistency check using the **fsck** utility.
- The **fsck** scans the file system and fixes any issues that can be safely corrected.
- In some cases, the file system can be in such bad shape that the operating system boots into single user mode to allow the user to further the repair process.

Journaling File System

- To see the nature of problem with non-journaling file system in Unix: removing a file need three steps.
  1. Remove the file from its directory
  2. Release the i-node
  3. Return all disk block to the free block list
- If there is a crash after any of these step, system need scan entire file system to recover!!!

Journaling File System

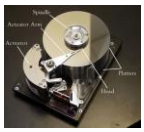


Journaling File System

- In general, journaling file systems avoid file system corruption by maintaining a **journal**.
- The **journal** is a special file that logs the changes destined for the file system in a circular buffer.
- At periodic intervals, the journal is committed to the file system.
- If a crash occurs, the journal can be used as a checkpoint to recover unsaved information and avoid corrupting file system metadata.

Disk Space Management

- Disk is divided into same size blocks.
- We need to decide size of a block .
  - Larger block size, internal fragmentation!!
  - Smaller block size, need more seek and rotational delay!!



Disk Space Management

- Larger block size, internal fragmentation!!
  - Lets consider disk space efficiency.
  - With 4 KB files with 1KB, 2KB and 4KB per block- no wasting space
  - With 4 KB file with a 8 KB block – 50% wasting space.
  - With 5 KB file with 1 KB, 2 KB, 4KB, 8KB block
    - With 1 KB – need 5 blocks : no wasting space.
    - With 2 KB – need 3 blocks : 1KB wasting space
    - With 4 KB – need 2 blocks : 3kB wasting space
    - With 8 KB – need 1 block: 3 KB wasting space

Disk Space Management

- Using a small block means that each file will consist of many blocks.
- Reading each block requires more seek and rotation delay in HDD.

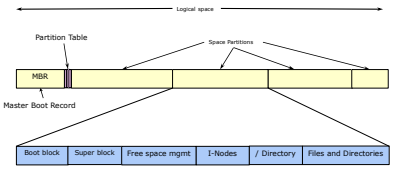
ex) a disk with 1MB per track, a rotation time of 8.33 msec and average seek time of 5 msec. The time in msec to read a block of k bytes is sum of seek rotation and transfer time:

$$\underbrace{5}_{\text{Seek time}} + \underbrace{(2^{20}/k)}_{\substack{\text{number of block} \\ \text{per track}}} \times \underbrace{8.33}_{\text{Rotation time}} + \underbrace{4.165}_{\text{initial Rotation time}} \text{(half of 8.33)}$$

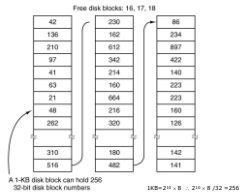
Free Block Management

- System reserve spaces (blocks in secondary memory) to save free block information.
  - Linked-List – free block information is saved in the blocks
  - Bitmap – system keep Bitmap for saving free block information (1 for used block 0 for available block)

File System Layout



Free Block Management
(Linked List)



Free Block Management
(Linked List)

- Each block for free blocks list holding as many free disk block numbers as will fit.

Ex)

- With a 1 KB block size and a 32 bit disk block number.
- A block can hold $8 \times 2^{10} / 32 = 256 - 1 = 255$ free blocks numbers (1 slot for next free block pointer)!!!
- 16 GB disk has $2^{34}/2^{10} = 2^{24}$ blocks
- Needs $2^{24}/255 = 65794$ blocks to hold free block numbers
- Need 65794 KB

Free Block Management
(Linked List Implementation)

- When the free list method is used, only one block of pointers need be kept in main memory.
- When a file is created, the needed block blocks are taken from the block of pointers.
- When it runs out, a new block of pointer is read in from the disk.
- When a file is deleted, its blocks are freed and added to the block of pointer in the main memory.

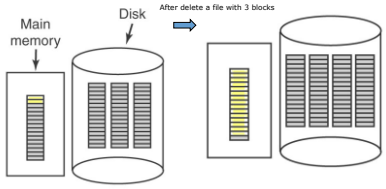
Free Block Management
(Bit Map)

Ex)

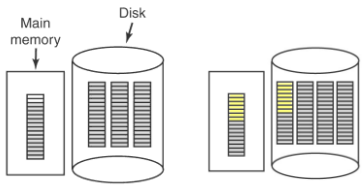
- 16 GB disk
- Size of block = 1KB
- There are $2^{34}/2^{10} = 2^{24}$ KB blocks
- Need 2^{24} bit (for bit map) = $2^{24} / 2^3$ Byte
  $= 2^{11} \times 2^{10}$ Byte $= 2^{11}$ KB
  Need 2048 blocks for the bitmap.

Free Block Management
(Linked List Implementation)

Under certain circumstances, linked method leads to unnecessary disk I/O

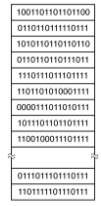


Free Block Management
(Linked List Implementation)



Free Block Management
(Bit Map)

A disk with n block requires a bitmap with n bits. a free blocks are represented by 1s in the map, allocated blocks by 0 (or vice versa).



A bitmap