Section 12: File Systems, Journaling

CS 162

August 3, 2020

Contents

1 Vocabulary **2**

2 Inode-Based File System (7/29) **3**

3 Comparison of File Allocation Strategies **5**

4 Logs and Journaling **6**

1 Vocabulary

- **Transaction** - A transaction is a unit of work within a database management system. Each transaction is treated as an indivisible unit which executes independently from other transactions. The ACID properties are usually used to describe reliable transactions.
- **ACID** - An acronym standing for the four key properties of a reliable transaction.
  - *Atomicity* - The transaction must either occur in its entirety, or not at all.
  - *Consistency* - Transactions must take data from one consistent state to another, and cannot compromise data integrity or leave data in an intermediate state.
  - *Isolation* - Concurent transactions should not interfere with each other; it should appear as if all transactions are serialized.
  - *Durability* - The effect of a committed transaction should persist despite crashes.
- **Idempotent** - An idempotent operation can be repeated withiout an effect after the first iteration.
- **Logging File System** - A logging file system (or journaling file system) is a file system in which all updates are performed via a transaction log ("journal") to ensure consistency in case the system crashes or loses power. Each file system transaction is written to an append-only redo log. Then, the transaction can be committed to disk. In the event of a crash, a file system recovery program can scan the journal and re-apply any transactions that may not have completed successfully. Each transaction must be idempotent, so the recovery program can safely re-apply them.

2 Inode-Based File System (7/29)

1. What are the advantages of an inode-based file system design compared to FAT?

2. What is the difference between a hard link and a soft link?

3. Why do we have direct blocks? Why not just have indirect blocks?

4. Consider a file system with 2048 byte blocks and 32-bit disk and file block pointers. Each file has 12 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer.

   (a) How large of a disk can this file system support?

   (b) What is the maximum file size?

5. Rather than writing updated files to disk immediately, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every x seconds. List two advantages and one disadvantage of such a scheme.

6. List the set of disk blocks that must be read into memory in order to read the file `/home/cs162/test.txt` in its entirety from a UNIX BSD 4.2 file system (10 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer). Assume the file is 15,234 bytes long and that disk blocks are 1024 bytes long. Assume that the directories in question all fit into a single disk block each. Note that this is not always true in reality.

3 Comparison of File Allocation Strategies

In lecture three file allocation strategies were discussed: (a) Indexed files. (b) Linked files. (c) Contiguous (extent-based) allocation.

Each of these strategies has advantages and disadvantages, which depend on the goals of the file system and the expected file access patterns. For each of the following situations, rank the three designs in order of best to worst. Give a reason for your ranking.

1. You have a file system where the most important criteria is the performance of sequential access to very large files.

2. You have a file system where the most important criteria is the performance of random access to very large files.

3. You have a file system where the most important criteria is the utilization of the disk capacity (i.e. getting the most file bytes on the disk).

4 Logs and Journaling

You create two new files, F_1 and F_2, right before your laptop's battery dies. You plug in and reboot your computer, and the operating system finds the following sequence of log entries in the file system's journal.

1. Find free blocks x_1, x_2, ..., x_n to store the contents of F_1, and update the free map to mark these blocks as used.
2. Allocate a new inode for the file F_1, pointing to its data blocks.
3. Add a directory entry to F_1's parent directory referring to this inode.
4. *Commit*
5. Find free blocks y_1, y_2, ..., y_n to store the contents of F_2, and update the free map to mark these blocks as used.
6. Allocate a new inode for the file F_2, pointing to its data blocks.

What are the possible states of files F_1 and F_2 *on disk* at boot time?

Say the following entries are also found at the end of the log:

7. Add a directory entry to F_2's parent directory referring to F_2's inode.
8. *Commit*

How does this change the possible states of file F_2 on disk at boot time?

Say the log contained only entries (5) through (8) shown above. What are the possible states of file F_1 on disk at the time of the reboot?

What is the purpose of the *Commit* entries in the log?

When recovering from a system crash and applying the updates recorded in the journal, does the OS need to check if these updates were partially applied before the failure?