EXTENDED MODEL ANSWER

Operating Systems II 2003 Paper 3 Question 5 (SMH)

Log-structured file systems

Note: do not require potential solutions to the problems with log-structured filing systems; however the answer gives some for clarity.

A log-structured file system (LSFS) works by treating the disk as a non-volatile append-only infinite buffer. Every data and meta data update (write operation, directory modification, etc) is performed by appending an update-record to the end of the log. This avoids costly seeks (and makes disk head scheduling trivial). Reads logically proceed by finding the most recent version of all relevant blocks (directory entries, inodes and data blocks). The key potential benefit is increased performance and throughput on write operations.

The problems are multiple. Firstly, locating things (inodes, data blocks, etc) on the disk is tricky since no-idea where to begin. This can be 'solved' by occasionally writing checkpoint records at fixed (well-known) locations on disk. Checkpoints can contain pointers to *inode maps* which themselves contain pointers to inodes. Once an inode is located, the relevant file or directory blocks are located from there.

The second and more important problem is that the disk is not actually infinite. Hence the operating system must periodically compact the log by removing redundant (old) blocks. This is a major pain, since naive compacting of entire log means lots of random access reads and writes. Threading through log gives fragmentation issues and really just defers the random access death to later. 'Best' solution probably segmented logs, but even then choosing correct segments to clean is tricky.

Journalling

A journal is essentially a non-volatile append-only infinite buffer as above. However unlike LSFS, this buffer takes only a small part of the disk and contains only *metadata* update records; the standard filesystem structure (ext2 or ntfs or whatever) is used in the remainder of the disk. Journalling is used, then, not as zero-seek optimization but rather in a way akin to a transactional log (unfortunate name overlap here).

The point of the journal, then, is to allow the efficient return of a file-system to a consistent state after a crash. This is particularly useful on modern large disks and filesystems since traditional 'scavenging' (e.g. via fsck or scandisk) typically takes time proportional to the size of the filesystem. Journal recovery on the other hand takes time proportional to the number of uncommitted metadata operations.

Tape backup strategy

A good way of achieving this is to use the *ruler function*. In this scheme we number our k tapes 0, 1, 2, ..., $(k-1)$, and number the days 1, 2, 3, ... Then on day d we use tape t such that 2^t is the highest power of two which divides d. Whenever we use tape t we dump all files modified since we last used that tape, or any tape with a higher number. With this scheme we keep files from the time they were last modified until at least 2^{k-1} days later (and at most $2^k - 1$ days later).

To achieve this we need the filesystem to allow us to obtain a list of files modified in the past 2^n days for various values of n. At the most basic level the filesystem need only maintain the last modification time (`mtime`) for all files. To avoid the `find`, however, a more sophisticated scheme (such as maintaining an index of files last modified on a certain day) would be required.