CS 333 :: Meeting Notes :: Fast File System (FFS)

Learning Objectives

- Be able to describe the FFS on-disk format
- Be able to describe the allocation and placement heuristics
- Be able to articulate the reasoning behind the FFS allocation and placement heuristics
- Given your knowledge of the FFS heuristics, be able to identify the best and worst case FFS inputs/behaviors

Overview of Concepts

The Fast File System (FFS) was developed as an alternative to the poorly performing default Unix file system. FFS employed a number of heuristics that were designed to take advantage of the performance characteristics of hard disk drives. We should know these well:

- I/Os require a *setup* cost to move the disk head and locate it above the first meaningful byte on the platter, followed by
- a *transfer*, where useful bytes are read from the platter Since setup (seeking + rotational delay) is wasted work, FFS tries to minimize the required work to setup an I/O.

FFS focuses on allocation and placement policies that try to minimize *fragmentation*.

The text identifies at least 3 problems that FFS attempts to solve

1. Data and metadata separation. Dependent reads require seeks
2. Fragmentation of free space
3. Small block sizes (transfer units)

cylinder/block groups

- a **cylinder** is a set of tracks that are the same distance from the center of the disk
- a **cylinder group** is a set of contiguous cylinders. You can think of cylinder groups as concentric vertical slices through the disk that are thicker than one track.

In the old days, disks were addressed using cylinder-head-sector addressing. This is no longer the case. Now disks are addressed using LBAs.

- A **block group** is a set of contiguous LBAs.

(You will find nothing in disk specs that say block groups correspond to cylinder groups, but this is likely the case.)

Questions:

1. What metadata structures does FFS use?
2. How does FFS use cylinder groups to organize it's on-disk data? (What data structures are placed where?)

FFS Policies

The FFS policies are a series of heuristics that aspire to, as often as possible, place related objects together. They are heuristics because they make no guarantees. We must hope that they continue to perform well over time.

Placing Files and Directories

To allocate a new directory:

- Find some cylinder group, *c*, with a low number of allocated directories inside it, as well as a high number of free inodes.
- Allocate the new directory inside *c*.

To allocate a new file:

- When allocating the inode for a file, try to place it in the same cylinder group as its directory.
- When allocating blocks for a file, try to place those blocks in the same cylinder group as its inode.

Questions:

1. How does the directory allocation heuristic affect the relative placement of two directories with a parent child relationship?
2. How does the file allocation heuristic affect the relative placement of a directory and a regular file with parent child relationship?
3. How does the file allocation heuristic affect the relative placement of a two files or directories with sibling relationships?
4. How do the file and directory placement heuristics (as described above) relate to file system growth (creating new files and directories).
5. What happens to locality when a file is moved from one directory to another?
6. What locality guarantees are provided when a hard link is created?

Large Things

FFS breaks ranges of file blocks into chunks, and spreads those chunks among different block groups. Thus, if the FFS chunk size is 16, then FFS will attempt to allocate the first 16 blocks of a file `f` in the same block group. When the `f` grows beyond 16 blocks, FFS will try to allocate blocks 16-31 in the same block group (but a different block group than blocks 0-15). `f` 's blocks are spread among block groups, but chunks of consecutive blocks are kept together.

Small Things

FFS has the ability to split data into units that are smaller than a physical disk block.

- *Sub-blocks* in FFS are small blocks (512 bytes) that FFS can allocate to files
- When enough sub-blocks are collected, those sub-blocks are copied into an actual physical block.

Thoughts

The FFS goal is to keep related things together. Yet only certain notions of related are prioritized.

1. When you perform a recursive directory traversal (perhaps `grep -r "somestring"`), what order do you traverse the file system namespace?
2. What seek behavior does that traversal order result in on FFS?

FFS does a lot of work to place data/metadata according to some heuristics.

1. Does FFS do any work to maintain its locality goals after initial allocation?
2. What happens if you rename a file? Create a hard link? Delete a file?