Thomas Nguyen

Professor Amer

COEN 177

December 10th, 2016

Assignment #4

In this file systems report, I'll be writing about HFS (Hierarchal File System), which serves as the file system for Macs. While researching the details of the file system, it was surprising to find that HFS is very unconventional in comparison to most standard file systems. This was primarily because HFS was designed to support a graphical user interface.

I'll first begin introducing by comparing HFS to most traditional file systems. In a traditional file system there usually exists an i-node table and explicit directories, however HFS utilizes B*-trees for its file system structures. One similarity between traditional file systems and HFS is its block bitmap which is utilized to represent file allocation.

In HFS, there are five types of "logical" blocks. Each of these blocks are inherently 512 bytes, however these blocks can be grouped together to form "allocation blocks". The first type of these blocks is called the Boot Block. The Boot Block is primarily used to store startup information. The next block is the Master Directory Block which stores a large amount of metadata about the entire volume. After that is the Volume Bitmap block which keeps track of the allocation of all our logical blocks. The size of the Volume Bitmap is determined by how big the size of the volume is since each bit in the bitmap needs to represent one logical block. The following logical block is the Extent Overflow File block which stores extra extents once all other extents have been used. The next block, the Catalog File Block, stores records of all files & directories on the volume.

To briefly go a little more in depth about both the Catalog File block, which is represented using a B*-tree, the Catalog File can consist of any of the following: a File Thread Record, a File Record, a Directory Thread Record, and a Directory Record. A Thread Record represents the name of the file and the Catalog Node ID of its parent directory. Whereas a Record keeps all the metadata of the file or directory. The metadata contains usual information such as the size of the file, the timestamps, and the pointers to the file's first data block. However the Record also stores additional information specific to HFS such as the information necessary for the GUI to know when opening the directory/file in the Finder.

Now that we have an understanding on some of the blocks that are used to construct the foundation of the file system, it's important for us to understand how the B*-tree is implemented in the Catalog File for us to access a Record – the logical block that contains all metadata and access to the physical data block for our file. The Catalog File is itself a B*-tree which begins with a header node that describes the entire B*-tree structure. Some of the information the header node obtains includes the depth of the tree, the number of the root node, and everything else the search algorithm needs to find a record. In the Catalog File's B*-tree, the most important thing to know about its structure is that it is a version of a binary search tree, however it can have multiple children for each node, each sub-tree's root node can must have the minimum node identification number, and only leaf nodes contain the actual data we want.

Therefore to access the leaf nodes, we can use binary search [O(log(n))] to traverse to the leaf nodes by continually asking whether the node id number is less than or greater than the current node's identification number. Based on this binary decision, you will traverse a certain direction in the tree. Once you have reached the parents of the leaf nodes, you will need to sequentially search [O(n)] through that parent's children since on parent can have multiple children. (Although a linear runtime would not be an ideal part of the search algorithm for a record, there will be minimal searching necessary, as the binary search will have covered a majority of the record organization.) Once you have found the correct node, you can check the node descriptor to locate your record within the node. Once you have located your record – which contains the access path (pointer) to the data of your file, you can sequentially search through the data blocks of the file to locate any arbitrary byte of that file. And there you have it!

I’ve attached some figures below to aid in the visualization of HFS.

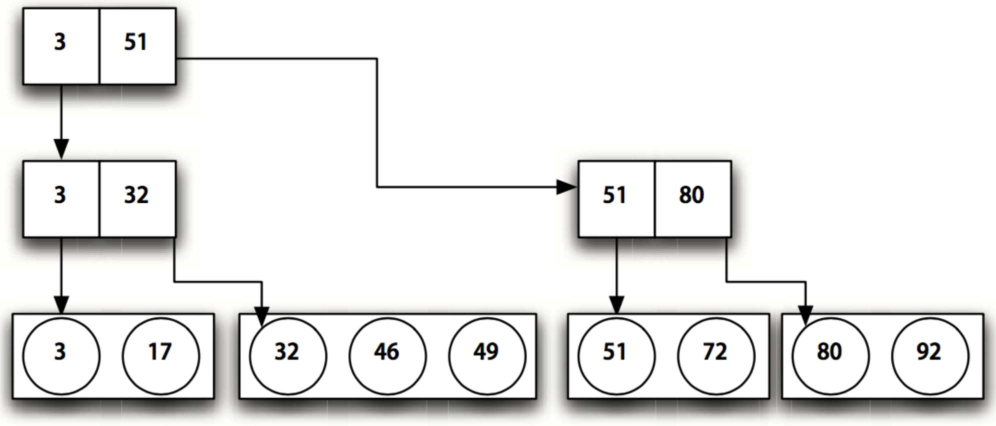

FIGURE 4 – AN HFS-STYLE B-TREE*

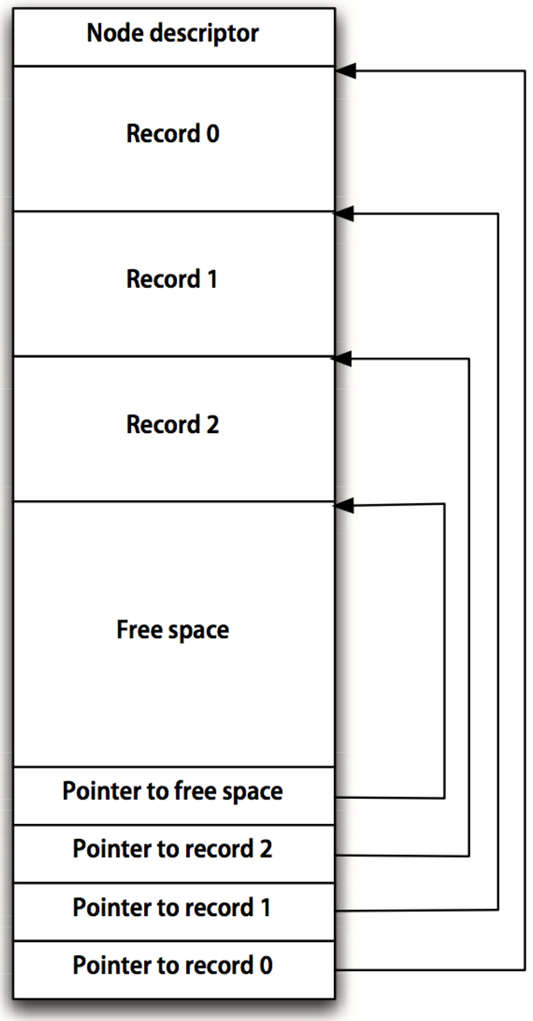

FIGURE 5 – NODE STRUCTURE

Citations

Deatherage, Matt, Justin Seal, Nathanial Irons, Jerry Kindall, John Welch, and John Gruber. "HFS and HFS Plus Complete." *MWJ* (2003): n. pg. Web. 10 Dec. 2016