Group 5 WOFS
Nick Lockett (nsl8), Matthew Wu(mlw55)
George Bernard (ghb5), Ryan St. Pierre(ras70)

On-disk structure

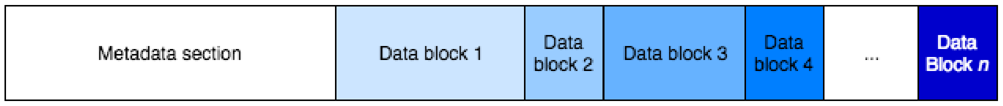

Figure 1- High-level overview of on-disk structure

Our on disk structure will consist of two main components - the metadata section, which contains the headers to directories, files, and the relationship between them, and the data block section, which contains the raw file data. The metadata section will precede the data blocks in the on-disk structure.

The metadata section represents the tree structure of the file system. On every read only one data block is accessed, while multiple portions of the metadata section are accessed to locate this one data block . We hope to leverage spatial locality and increase (cache read) performance by placing the metadata headers together at the start of the image file, rather than placing metadata directly in front of their respective files.

Data blocks will be of variable sizes each with size just large enough to accommodate its file content. Of course data block sizes will be multiples of the *word* size (8 bytes) to stay aligned. This may lead to *small amounts* of fragmentation for file structures of many files with each below 8 bytes, we welcome advice for better word or alignment sizes.

Metadata On-disk structure

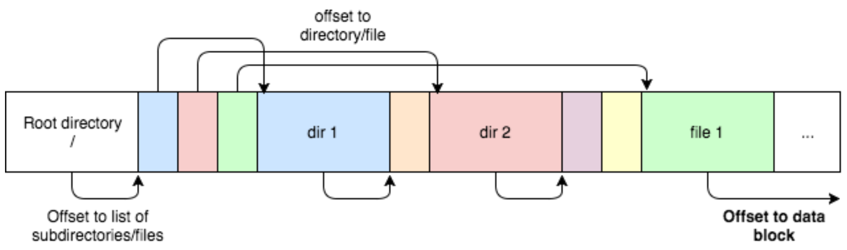

Figure 2- Metadata section on-disk structure

Our metadata headers will have fixed size and include the following information:

- *Name (256 bytes/characters):* Name of the file or directory. This is a common maximum (at least for linux files/systems).
- *Type (unsigned int):* Directory or file. This is currently an *unsigned int* for development flexibility, in case we need more types in the future. This will likely be implemented as an enumeration in the plural case. If not this might be scaled back to an *bit*.

- *Data block/Subdirectory offset (unsigned long):* In case of a directory, an offset to the list that containers offsets to its subdirectories. In case of a file, an offset to its data block.
- *Length (unsigned long):* In the case of a directory, the number of subdirectories and files it has. This will allow us to know the size of the subdirectory list mentioned above. In the case of a file, the size of the file (bytes).
- *Time (unsigned long):* Time in UNIX time of last access.

Note: The above list is not necessarily complete. Other metadata information maintained by the linux file system can be added to our header representation as desired (such as date created, time accessed, etc.). However, we believe the above includes at least enough to represent the file system properly.

Although our metadata header structures are of fixed size we need to accommodate a variable number of subdirectories and files under a given directory. To accomplish this each metadata header structure has an offset value that corresponds to a list of offsets of subdirectories and files. In Figure 2, this is represented by the blue, red, and green blocks after the root directory metadata, which give the offset to directory 1, directory 2, and file 1 respectively. Given our on-file layout this would mean directory 1, directory 2, and file 1 are subdirectories and subfiles of the root.

Questions/Concerns

- Variable size data blocks: Given this is a WOFS we see no need to have a fixed size for our data blocks, as long as we maintain alignment. Is 8 bytes a “good” choice for word size, especially for common unix loads of many very small files? Will this cause any performance problems that we are not considering?
- Fixed size metadata header structures with subdirectory list: The subdirectory list is core to our on-file disk structure. We wanted to make sure we our not neglecting any cases in which this implementation would be problematic.
- Search: Upon read we image traversing metadata structure section, starting at the root. In our current implementation this would force us to first check all subdirectories directories/files of the root for a name match and then recurse this process with the next directory in the path. To speed this up we considered a hashing scheme based on the name of the file/directory, but though this was not worth the space tradeoff. Additionally, the traversal should be quite quick given read caching on the metadata section block. Do you have any additional thoughts on this?
- Forgetting something: Lastly, we just wanted to ensure we are not forgetting anything in the metadata structure header that is crucial to the implementation of the file system.