Design considerations

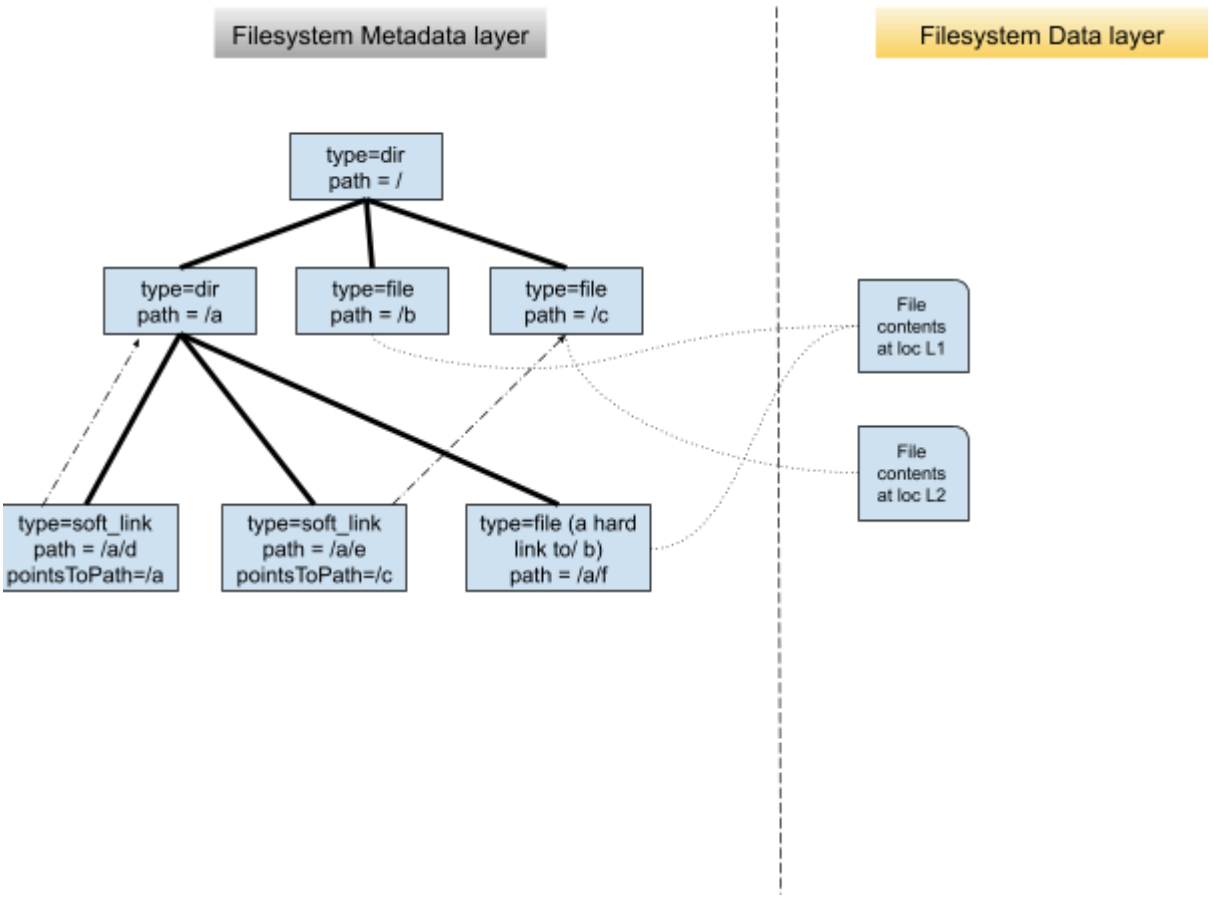
- A metadata layer for maintaining the directory tree.
- Would be nice to build in a layered fashion separating the metadata and the data layer, so that both the metadata and data layer could be made pluggable (eg say a persisted file system instead of in-memory
- Should scale with more memory available (ie ability to create more files and directories).
- Cleanup up files/directories should be garbage collected and avoid memory leaks.
- Avoid redundant memory allocations during hard linking.
- The problem asks to support writing data to file, and not appending data to existing files. However, let's design in a way that it can be extended to support appending as well.

Restrictions

- Only support file system operations within a process. That is, across processes there will not be any shared view of the filesystem. It will also be single-threaded, to keep it simple and avoid contention across threads and the need for locking.
- Hard linking a directory will not be supported to avoid infinite loop.
- Filenames will be limited to 100 chars.

Design

- Language of choice is Java
- Use a D-array structure to represent the directory tree hierarchy starting from the root node.
- Have references from parent/child in both directions to easily support relative path accesses.
- Soft-links will be lazily evaluated since the file they point to could appear at a later time.
- Since the language of choice is Java,
    - Hard links are easy to support, since java uses references.
    - Java GC collection automatically handles cleaning up of unused file-contents automatically, as long as the references to them are deleted. Hence hard links are automatically handled


Filesystem Metadata layer
Filesystem Data layer
type=dir
path = /
type=dir
path = /a
type=file
path = /b
type=file
path = /c
File
contents
at loc L1
File
contents
at loc L2
type=soft_link
path = /a/d
pointsToPath=/a
type=soft_link
path = /a/e
pointsToPath=/c
type=file (a hard
link to/ b)
path = /a/f