Separating Directory Structures from Physical File Systems

Dan Muntz
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2001-174
July 12th , 2001*

E-mail: dmuntz@hpl.hp.com

file systems, distributed, naming, directories

Traditionally, file systems contain directory structures that are tightly bound to a particular file system implementation. These structures may be embedded, both logically and physically, in the file system and contain data that are specific to the file system implementation. Changing the directory structure of a file system can be extremely tedious: the file system code must be changed and rebuilt, new file system initialization code (mkfs) is needed, and new recovery code (fsck) is also likely to be necessary. Several areas of file system research could benefit from a generic directory structure that is implemented *above* the physical file system layer, allowing experimentation with directory contents and possibly alternative naming schemes.

* Internal Accession Date Only
Approved for External Publication

Separating Directory Structures from Physical File Systems

Dan Muntz

Hewlett-Packard Labs
1501 Page Mill Rd, Palo Alto, CA 94304, USA
dmuntz@hpl.hp.com

Abstract

Traditionally, file systems contain directory structures that are tightly bound to a particular file system implementation. These structures may be embedded, both logically and physically, in the file system and contain data that are specific to the file system implementation. Changing the directory structure of a file system can be extremely tedious: the file system code must be changed and rebuilt, new file system initialization code (`mkfs`) is needed, and new recovery code (`fsck`) is also likely to be necessary. Several areas of file system research could benefit from a generic directory structure that is implemented *above* the physical file system layer, allowing experimentation with directory contents and possibly alternative naming schemes.

Introduction

Each underlying file system has its own definition for the structure of a directory. *Directory files* provide a directory structure that is independent of the underlying file systems and allow great flexibility in customizing directory structures to particular tasks. A *directory file* is simply a file in the underlying file system that contains "pieces" of the namespace for a file system built on one or many underlying file systems. Depending on implementation, directory files may be interpreted by file system clients, servers, or both. Directory files serve the same purpose as traditional directories—they completely describe the namespace of their file system, and they are used very similarly to existing directory structures. Their advantage lies in their independence from underlying file systems, and the flexibility this allows.

An example

Directory files play an important role in the implementation of a new file system architecture being developed at HP Labs. A primary goal for this architecture is to unite heterogeneous file systems into a single namespace, where any object in the namespace can reside in any underlying file system, on any file server. For example, if a user has large multimedia files, small text files, and some files that are relatively static over time, all of these files may appear in a single directory, but may be served transparently from different file systems and/or file servers particularly suited to each group.

To achieve this goal, the directory structure is augmented so that a name in the namespace may refer to an object on another server; potentially in a different type of file system. Implementing this change to the directory structure would normally require changes to the code and tools of each underlying file system for which support is needed. However, by implementing a namespace above the underlying file systems, it is possible to change the directory structure for the experimental file system without changes to each individual file system.

In this case, there is a directory file for the root of the experimental file system. This directory file is just a regular data file in some underlying physical file system. It is initialized with entries for "." and ".." and namespace construction may then proceed from this point. For example, if the root of the file system resides on a server, S1, and a new file, /F1, is created on another server, S2, an entry is made in the root directory file, containing information typically found in a directory entry, plus any additional information

needed for the experimental file system, e.g., the information that F1 is located on S2. Currently, there is a directory file implementation on a Linux 2.4.2 NFSv2 server that functions transparently with existing NFS clients.

Implementation

An existing implementation of directory files was completed on Linux 2.2.14 and subsequently on Linux 2.4.2. This implementation consists of modifications to the Linux kernel NFS server code, but works with any NFS client. A directory-file-based file system is created by initializing a root directory file, called `ROOT`. The directory containing this file is then exported. The `ROOT` file is initialized with entries for "." and ".." that both refer to `ROOT`. An entry in a directory file consists of:

- The user's name for the object (e.g., `passwd`)
- The system's name for the object in the underlying physical file system (e.g., `file.001`; see below)
- The type of the object (e.g., directory or file)
- Any other information that may be necessary (e.g., server where the object is located)

For this implementation, file system objects are created in a flat namespace in the underlying physical file system(s), and are given unique object identifiers (a name in the namespace of the underlying file system). Thus, the directory entry associates the users's name for a file with a unique object id used by the system to retrieve object contents.

In the NFS server code, an exported file system is flagged as a "directory-file file system" if it contains a ROOT directory file (this will change in a future implementation, and is just a temporary hack). All directory operations for this file system are intercepted and interpreted in the directory file context. For example, if a `readdir` request is received, the corresponding directory file is opened, its contents are read, and the appropriate `readdir` response is constructed and sent to the client. Reading and writing of the directory files is handled through the vnode operations (dirops and fileops) for the underlying file system.

Future Work

There are at least two uses for directory files currently being investigated, or considered for work in the near future. First is the concept of using different data structures in directory files to improve performance (even for standard NFS file servers). There is currently an implementation of directory files using hashing that provides improved performance for very large directories.

Another potential use for directory files is the exploration of alternative naming schemes. As the contents may be changed easily, it is possible to explore associating various keys or properties with objects in the namespace. Files may then be located by searches on these keys.

Preliminary testing has shown some cost for using directory files, obviously, but the flexibility of directory files has also resulted in cases where performance exceeds that of using the physical file system's directory structure. More performance work is necessary, both for evaluation and optimization.

Related work

ReiserFS [1] uses an alternative directory structure to increase the performance of a particular physical file system. By using a B-tree structure, RiserFS increases performance for operations on large directories. Directory files could be used to implement a B-tree or any other data structure on top of any underlying file system. This concept is also useful for experimenting with various data structures before committing to the larger task of developing a physical file system committed to a specific directory structure.

Conclusions

Directory files provide a very flexible framework for experimenting with changes to directory structures and alternative namespaces. There is a working

implementation that is providing the namespace for ongoing file system experimentation. The separation of the namespace from the underlying physical file systems greatly simplifies the task of unifying heterogeneous file systems. While there is an obvious cost to using a namespace implemented above existing file systems, there are cases where this is not only useful for experimentation, but may also improve performance.

References

1. Reiser, H., *ReiserFS*, 2001. http://www.namesys.com/res_whol.shtml.