Dynamically Updating Virtual Search Folders on Unix System

Elizabeth Dethy

February 28, 2014

1 Overview

This design proposal provides an overall system design for implementing searchmount. The primary advantage of this feature is dynamic updating which provides the user with an efficient tool for creating and maintaining virtual directories.

This design approach stores the results of different search queries in databases on−disk. A notification system, inotify, is used to alert the file system that files have been changed and to update the corresponding database. Updating the databases will not cause significant performance delays on either the existing file system or in accessing files from the virtual directories.

2 Design Description

2.1 Databases

The key data structure used in this design is a collection of databases that store the results of each search query. Two different formats will be used. The databases will be maintained on disk and be persistent across restarts.

2.1.1 The Virtual Directory Database

In this implementation a virtual directory database ("VDD") is created when a user mounts a searchmount call. The keys in the database correspond to the filenames returned by the search query. Three pieces of meta-data are stored as values: a symlink to the file, the previous key, and the next key. Maintaining pointers implements a modified doubly linked list datastructure within the database. This is illustrated in the figure below. The doubly linked list datastructure makes updating the database efficient and makes it easy to find the next file in a virtual folder.

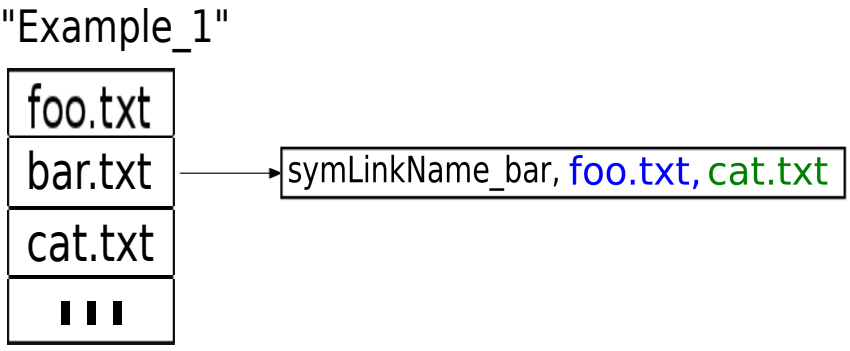

Figure 1 A diagram of a VDD named Example_1. Each filename is associated with a symlink, previous key ("foo.txt") and a next key ("cat.txt"). The first filename is foo.txt.

2.1.2 The Master Database

The master database is created when the user mounts their first searchmount call. The master database stores the information needed to query the VDDs and store general information about each virtual directory. The keys in the master database are the names of the virtual directories the user has created. The metadata associated with each key is the filename of the first file in the VDD, a record of inotify notifications, the search expression and the search path. The format of the master database is illustrated in the diagram below.

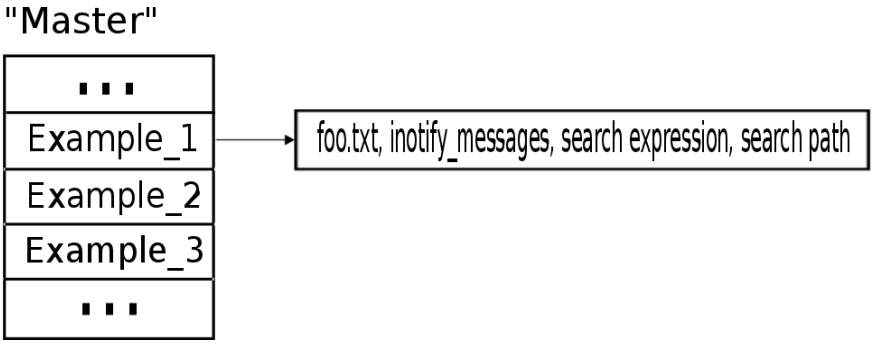

Figure 2 The Example_1 database key points to the first file in Example_1, foo.txt, and the associated inotify messages.

2.2 Creating a New Virtual Directory

When a user mounts a new virtual directory the UNIX find command is run using the search path and search expression parameters specified in searchmount. The resulting filenames are stored in a corresponding VDD and the search parameters and first file name are stored as values in the master database. Additionally, inotify watches are set on the root search directory and all subdirectories.

2.3 User and Application Programming Interface

This implementation will support the following objects and methods for users and applications to interface with the virtual file system.

- **DirEntryPointer:** specifies a filename, a symbolic link to the file, and the virtual directory containing the symbolic link to the file. If multiple virtual directories contain symbolic links to the same directory, a DirEntryPointer object is created to represent each symbolic link. All database queries return a DirEntryPointer object.
- **getFirstDirectoryEntry(name):** Queries the master database using the name as the key. Uses the value _key value to query the corresponding VDD database.

- **readNextDirectoryEntry(DirEntryPointer previousFile):** reads the virtual directory and filename from the previousFile object. Queries the database for the previousFile. The value next_key is used to query the database again for the desired file.
- **fileName(DirEntryPointer entry):** reads the filename from the entry.
- **readSymbolicLink(DirEntryPointer entry):** reads the symbolic link from the entry.

2.4 Updating the Virtual Directories

Virtual directories update when a user make an API call to the virtual directory. Inotify updates are retrieved from the master database. The search query is run on all changed folders but is not applied recursively. If sub−directories have been changed the inotify watches on those folders will have alerted the file system and they will be queried independently. The database is updated with the results of the searches (either adding or removing files). The diagram below illustrates removing the file bar.txt from figure 1.

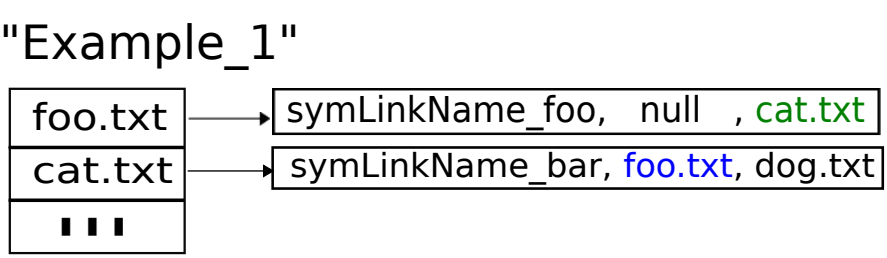

Figure 3 The next_key value of foo.txt is updated to cat.txt (replacing the old bar.txt with bar.txt's next_key value. Similarly, the prev_key value of cat.txt is updated to foo.txt (replacing the old bar.txt with bar.txt's prev_key value).
On a system restart, because inotify watches are not persistent across restarts, all inotify watches will be started again.

3 Conclusion

The proposed design of the searchmount feature supports all methods necessary to interface with either a user or an application. Future optimizations could include caching recently used virtual directories in memory for faster access and optimizing database updates.