A Search-Folder System for Unix

Jason Paller-Rzepka

6.033 Design Project 1

Proposal

Overview

Unix users often repeat certain `find` commands to locate files of interest. One way for them to eliminate this repetitive task from their workflow is to create Search Folders. A Search Folder is a virtual directory that contains the results of a file search query. Search Folders automatically update their contents as the results of the file search change, so they are always up-to-date. By using Search Folders, Unix users can significantly the number of repetitive file searches they perform.

This proposal discusses the design of a Search-Folder system for Unix. The system maintains iterable databases of search results, and listens for file-system updates to detect changes and update those search results. The design allows the system to operate quickly: requests to iterate through directory contents are fast, and the search results update very soon after the underlying data is modified.

Design Description

The design consists of two components: *databases* that describe the properties and contents of Search Folders, and *file-update queues* for updating those changes when underlying data is modified. The public interface to those components lets clients create Search Folders, and get their contents.

Databases

The system stores one Contents Database per Search Folder. It also stores several databases that contain metadata for all the Search Folders.

Contents Database

A Search Folder's Contents Database stores the full path of each match, where a "match" is a file that satisfies that Search Folder's search criteria. It stores the data in key-value pairs: each key is the full path of a match, and each associated value contains data that allows the Contents Database to imitate a doubly linked list. Specifically, the value is a pair that contains the full paths of a previous match and a subsequent match. This database also contains a *null* path, the empty string, which represents the beginning and the end of the list. See Figure 1 for an example Contents Database.

The Contents Database has two key properties. First, it allows for the quick lookup, addition, and deletion of individual matches, because the rows are keyed by match paths. This

allows the system to add and remove matches quickly. Second, it is quick and easy to iterate over matches: given a file path, finding the subsequent match is as easy as looking up that file's row in the Contents Database and reading the subsequent match's path from that row. This makes the system easy to implement.

Metadata Databases

The system also maintains three metadata databases. Each contains one row per Search Folder. The first maps a Search Folder path to a Contents Database filename. The second maps a Search Folder path to a file-update queue filename. The third maps a Search Folder to its query parameter string.

File-Update Queues

The system creates an `inotify` queue for each Search Folder. This queue receives updates from the searched directory, and all its subdirectories, recursively. A background process listens to these queues. When a file is added or modified, the process adds it to the correct Contents Database if it matches the search criteria. When a directory is added or modified, the process makes them publish their own `inotify` events to the queue, and adds the correct files to the Contents Database. When a file or directory is deleted, the process removes any deleted paths from the Contents Database.

Using file-update queues makes the system fast. As soon as a searched directory is modified, the system updates the Contents Database to reflect the modification. Importantly, updating the Contents Database does *not* require the entire searched directory to be examined—the system only examines the files or directories that changed. So, the system works quickly, and remains responsive.

API

The system's interface consists of methods to do the following:

- **Create a Search Folder** for a directory, with some search parameters.
- **Read the entries** from a Search Folder (with a `getFirst` method and a `getNext` method).
- **Produce a human-readable name** for an entry in a Search Folder.
- **Get the target** of an entry in a Search Folder—that is, the full path of the file that this entry represents.

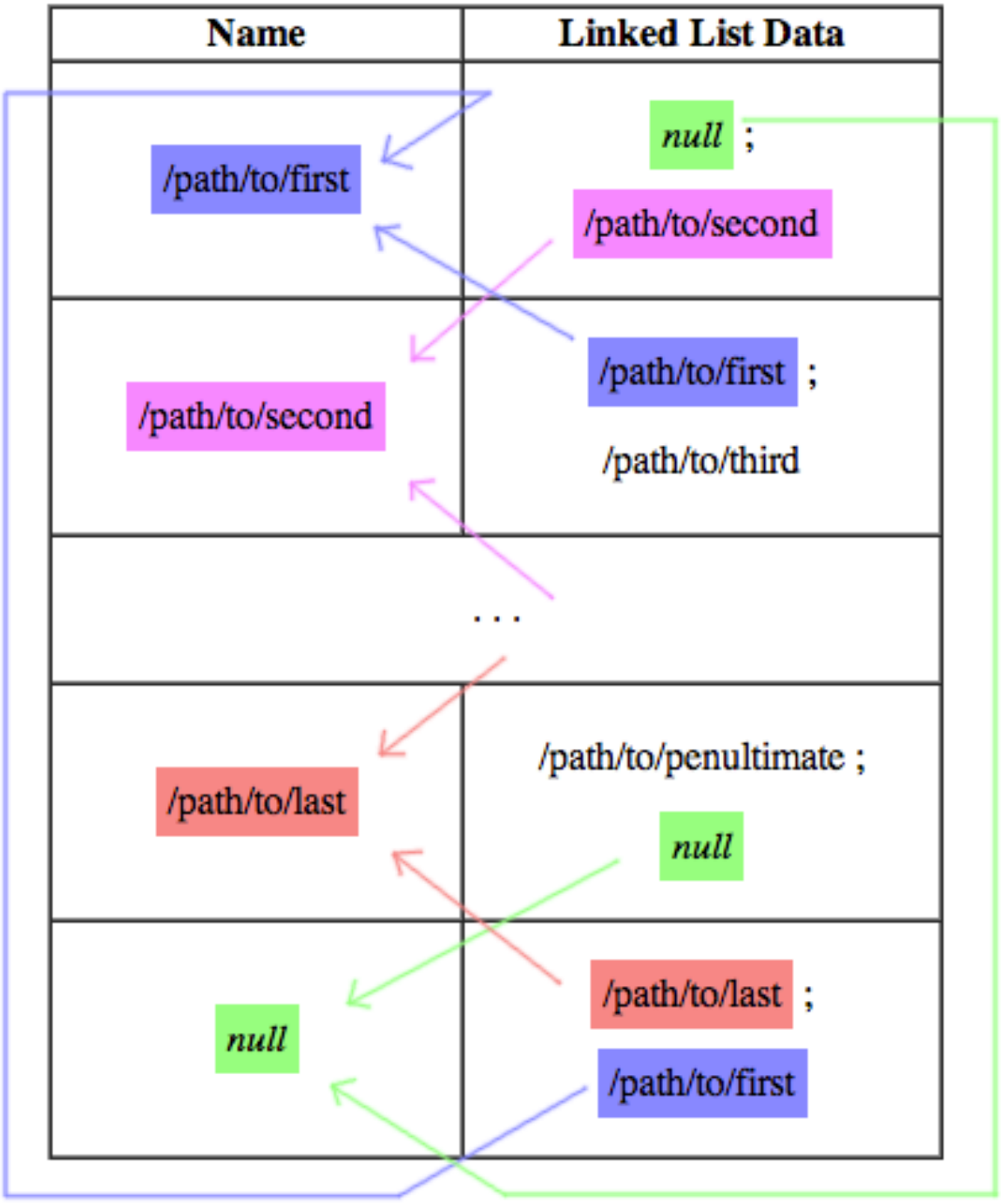

Figure 1: The Doubly Linked-List Structure of the Contents Database
Each row of the table is a row of the Contents Database. Each key is a filename. Each value contains two filenames: one pointing to the previous row, one pointing to the next row. The *null* key represents the edge of the list—it comes before the first key, and it comes after the last key.

Conclusion

The above design describes an efficient system for implementing Search Folders in Unix. The Contents Database can be modified quickly, and the doubly linked-list structure allows for quick, easy iteration over its contents. The file-update queue system keeps the Search Folders up-to-date without delays, both because the listener responds to changes immediately, and because it works just with changed files instead of re-executing entire directory searches. The question of how to synchronize the actions of the listener and any processes that iterate over a Contents Database remains to be explored.