Monster Mash

A UNIX-like shell and filesystem

Alok Hota & Mohammad Raji

March 7, 2015

1 About

Monster Mash is a UNIX-like shell and virtual filesystem. Its name was originally MASH, which stood for **M**ohammad and **A**lok's **SH**ell. This name was taken however, so we added *monster*, referencing the hit single by Bobby Pickett. It supports various simple commands similar to basic UNIX commands. The underlying filesystem is based on UNIX's inode and data block structure.

2 Installation

The repository is available on Github (`https://github.com/ahota/monstermash`). To build Monster Mash, simply run `make`. This will create two executables, `mmash` and `client`.

3 Usage

The `mmash` executable will start the shell and filesystem in local mode. In this mode, the user can interact with the filesystem using the commands below.

Alternatively, Monster Mash can be started in server mode by running: `mmash -s [port]`. This will start the server and wait for a client to connect. The `port` argument is optional. If it is provided, the Monster Mash server will communicate with the client on that port. If it is not provided, the server will use the default port, 1962, which is the year *Monster Mash* was released as a single.

The Monster Mash client is used by running: `client <hostname> <port>`. Both the server `hostname` and `port` are required. Once the client has connected, usage is identical to running Monster Mash in local mode.

The following commands are supported:

- `mkfs`: Formats and (re)generates the filesystem. This is run automatically when starting Monster Mash locally or when a client connects. This creates a root directory (`/`) and sets it as the current directory
- `open <filename> <r|w|rw>`: Opens a file for reading (`r`), writing (`w`) or both (`rw`) depending on the flag given. If a file is opened with the `w` or `rw` flag and does not exist, it will be created. After a file is opened, a file descriptor is printed for later access
- `read <fd> <size>`: Reads `size` bytes from an open file with file descriptor `fd` if the file was opened for reading
- `write <fd> <string>`: Writes `string` to an open file with file descriptor `fd` if the file was opened for writing
- `seek <fd> <offset>`: Seeks to `offset` (bytes) in file with file descriptor `fd`
- `close <fd>`: Closes file with file descriptor `fd` if it was open
- `mkdir <name>`: Creates a directory `name` in the current directory. `name` can have spaces if it is surrounded by quotes
- `rmdir <name>`: Removes the directory `name` from the current directory. The directory must be empty
- `cd <path>`: Changes the current directory to `path`. `path` must be relative to the current directory. It can have spaces if surrounded by quotes
- `link <target> <linkname>`: Creates a link `linkname` to file or link `target`. Paths to `link` and `target` can be relative

- `unlink <name>`: Unlinks a link `name` or deletes a file `name`. If the target would have zero links to it after this element is unlinked/deleted, the data is erased from the disk
- `stat [name]`: Provides the following information to the element `name` (or the current directory if `name` is not provided):
  - name on disk
  - inode ID
  - element type
  - link count
  - number of blocks allocated
  - size on disk
- `ls`: Lists all elements in the current directory
- `cat <name>`: Prints the contents of the file `name`
- `cp <src> <dest>`: Copies the file or link `src` to a new file or link `dest`. `src` and `dest` can be relative paths with spaces if surrounded by quotes
- `tree`: Lists the contents of all directories hierarchically from the current directory along with their total file size
- `import <src> <dest>`: Imports a file `src` from the host to a file `dest` on the Monster Mash filesystem
- `export <src> <dest>`: Exports a file `src` from the Monster Mash filesystem to a file `dest` on the host
- `exit`: In local mode, this shuts down the shell and returns the user to the host. In server mode, this severs the connection between server and client and shuts both processes down. The Monster Mash disk is still available on the host

4 Architecture

The Monster Mash filesystem is based on concepts from a typical UNIX filesystem.

4.1 inodes

The filesystem uses inodes to keep a record of elements. Each element, whether it is a file, directory, or link, has one inode. Each inode is 8 bytes in size and has the following structure:

Type	Name	Size	Purpose
`char`	`type`	1	Type of element: 'f', 'd', or 'l'
`short`	`id`	2	Unique ID for this inode
`int`	`first_block`	4	Offset to the first block allocated for this element
`char`	`pad`	1	Padding to reach 8 bytes
	Total	8	bytes

The inode table supports up to 1024 inodes. Since each inode is 8 bytes, the inode table occupies 8 kB of the disk.

Type	Name	Size	Purpose
char *	name	32	The name of this element or 'CONTINUED'
int	next_block	32	Byte offset to next block or -1
short	link_count	2	Number of links to this element
char *	data	4058	Contents of this element
	Total	4096	bytes

4.2 Data Blocks

The first data block begins immediately after the inode table. Blocks are 4096 bytes each and have the following structure:

The first data block for an element (ie the block pointed to by `first_block` in this element's inode) contains the correct name in its `name` field. Names for elements are limited to 32 characters. When a block is created, its `next_block` pointer is set to -1 and `link_count` is set to 1. `link_count` is incremented and decremented as links to this element are created or unlinked, respectively. If a file is created and linked to, its `link_count` would be 2. If the *file* is then deleted, the data blocks associated with that file are still kept, as `link_count` is still 1 due to the link. If that link is later unlinked, the data blocks are erased.

Directories hold a table of their contents in the `data` section of their block(s). Each entry in the table contains two values: inode ID and name. inode IDs in the table are used for unique referencing in commands such as `cd`. Names in the table are used for printing and user-reference in commands such as `ls` and `tree`. Files use the `data` section to hold their contents in plain text. Links use this section simply to hold a reference of the inode ID to which they link.

If the data in a directory or file grows beyond a single block, a new block is automatically allocated and linked to in the current block's `next_block` pointer. This new block's `name` field contains 'CONTINUED' to distinguish it as a continuation block. Block allocation and traversal is completely abstracted from the user during normal use. The number of blocks allocated for an element can be seen with the `stat` command.

5 Limitations

Currently `mmash` supports 1024 inodes as a maximum. This means there cannot be more than 1024 elements on the filesystem, including the root directory. The total inode table size is 8KB. The disk itself is 100MB total, less the inode table and block meta-data size, leaves 99.06% of the disk for data.

In the current version, the filesystem is not persistent across runs and is erased every time `mmash` is started.