Funny File System Specification

Version 1.0, 1 March 2024

The Funny File System (FFS) is a file system created for the Funny Operating System (FunnyOS). FFS was designed with three goals in mind:

- Efficient use of disk space
- Ease of implementation
- Performance

You can see these goals reflected in the structure of FFS. It is just useful enough to be fully functional without requiring significant overhead.

The Funny Operating System is seen as the reference implementation of FFS. The source code for it is open, and can be found at https://github.com/Finxx1/funnyos.

Contents

Notation and Conventions

This document uses several notational conventions in order to ease reading and prevent confusion when implementing FFS.

Punctuation is intentionally placed outside of quotes in order to prevent ambiguity. If there is punctuation inside the quote, it is part of whatever it is referring to.

Integer types in this document are referred to by their bit count, using the format “intN_t”, where ‘N’ is the amount of bits in the integer. The type may also be prefixed with a ‘u’ to represent an unsigned integer. These type conventions can be used in C source code by including the “stdint.h” standard header.

All integer values are stored in “little endian”. This means that on “big endian” systems, you will need to rearrange the byte order. Below is an example function that does such.

```
uint64_t swap64(uint64_t x) {
    return
    ((x << 56) & 0xff00000000000000UL) |
    ((x << 40) & 0x00ff000000000000UL) |
    ((x << 24) & 0x0000ff0000000000UL) |
    ((x <<  8) & 0x000000ff00000000UL) |
    ((x >>  8) & 0x00000000ff000000UL) |
    ((x >> 24) & 0x0000000000ff0000UL) |
    ((x >> 40) & 0x000000000000ff00UL) |
    ((x >> 56) & 0x00000000000000ffUL);
}
```

Strings in FFS are a sequence of characters, ended by a NULL termination. The size is calculated by finding the position of the NULL terminator. In practice, this looks like below.

Hex	0x68	0x65	0x6c	0x6c	0x6f	0x2e	0x0a	0x00
ASCII	h	e	l	l	o	.	\n	\0

An absolute path as referred to in this document is the full path of a file. Directories are delimited by a forward slash (‘/’). The way you address a particular partition root is implementation defined. For example, a file called “secret.txt” put inside a folder called “do not touch” would lead to an absolute path of “/do not touch/secret.txt”. On a DOS-like system, it might be “D:\do not touch\secret.txt”, and on a Unix-like system, it might be “/mnt/do not touch/secret.txt”.

A timestamp is a 64-bit unsigned value that counts the number of seconds since the Unix Epoch (1 January 1970). Higher-precision timing was deemed unnecessary.

Structure

A key part of FFS is the way files are listed. FFS creates two stacks; one at the start of the partition, and one at the end. The one at the start has file data, and the one at the end has file entries. This allows for maximum use of disk space, and addresses short comings of other file systems that have a limited number of file entries.

An FFS partition begins with a header that describes important info.

Type and Name	Description
char signature[8]	Must be “funny :)”
uint64_t partitionsize	Size of the partition, in bytes
uint64_t datasize	Size of data stack, in bytes
uint64_t entrysize	Size of file entry stack, in bytes

Every file entry follows the same format.

Type and Name	Description
string path	The file’s absolute path
uint64_t data	The offset of this file’s data in the data stack
uint64_t size	The size of this file's data in the data stack
timestamp modified	Date and time of the file’s last data modification

FFS has no directory structure. Rather, every file is scanned in order to find what is in a directory. This is expected to happen just once at the start of an operating system’s initialization and keep a copy in memory, as to not cripple performance. If a directory listing is too large, another solution may be employed.

In Practice

The following is an example hex listing of a 64-byte long FFS partition.

66	75	6e	6e	79	20	3a	29	f	u	n	n	y		:	)
40	00	00	00	00	00	00	00	(	.	.	.	.	.	.	.
01	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.
1f	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.
4c	6d	79	2e	74	78	74	00	L	m	y	.	t	x	t	.
00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.
01	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.
00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.

Legend	🟧	🟥	🟦	🟩	🟫	🟩	⬜	🟪	🟪
	sig.	size	datsize	entsize	filedat	name	datpos	datsize	time

This partition should have a single file, called “my.txt”, with its data being a single letter ‘L’. Partitions clearly should be larger than this, but this should be enough to get off the ground.