Assignment 3: File Systems

TINBES02-2

Bastiaan Teeuwen

Hogeschool Rotterdam

April 16, 2019

File systems are the format in which data is formatted on disk. Some examples of file systems you may know are NTFS (the default file system Windows installations use), FAT16, FAT32, exFAT (which are mostly used for USB drives nowadays), ext2, ext4 (which are used on Linux) and so forth.

1 Linked lists

A linked list is commonly used in C to bind together non-adjacent memory locations. They essentially provide the same functionality as an array (in which every entry *is* guaranteerd to be more or less adjacent). The big difference however is that entries can be added dynamically to the linked list, which cannot be done with a simple array.

You'll see an example of a linked list in the next section.

I won't be giving much more information away here as linked list are pretty self-explanatory if you understand how pointers in C work.

2 ramfs

The file system we'll be writing today will only support one directory to keep things simple. We also won't be reading and writing to disk, but to RAM. This means that all saved files will be lost on a system reset. Hence we'll call it ramfs, sounds good right?

Go ahead and open up "`ramfs.c`" and "`include/ramfs.h`". Let's look at the structure defined in the header file first:

```
struct file {
        char            name[NAME_MAX + 1];
        char            *data;
        size_t          size;
        struct file     *prev, *next;
};
```

This structure represents a file in our file system. The first file is pointed to by `head` in `ramfs.c`.

`name` is the name of the file, which can be `NAME_MAX` bytes + 1 (for the null character) long.

`data` points to a memory location where the contents of the file are stored.

`size` is the length of the data memory location in bytes.

`prev` and `next` are used to point to the preceding and following file respectively. This is called a *linked list*.

Some things you should keep in mind before you start:

- File names are unique, no two files with the same name can exist
- Use `malloc()`, `realloc()` and `free()` from `sys/alloc.h` to allocate, reallocate and free memory dynamically respectively.
- Use the functions you have implemented in the memory assignment.
- Handle `NULL`, names longer than `NAME_MAX` and other mistakes a user or user of ramfs may make correctly to avoid undefined behavior.
- If you're having trouble understanding how everything fits together, get out some pen and paper and draw out the relations between all functions, `struct file` and the data location of files.

The empty functions in '`ramfs.c`' have comments above them describing the action they should perform and what value should be returned. The rest is just common sense and knowledge of C.

3 Commands

To make our file system more useful, we'll add an interface on the command line. Add the following commands to the command line interface you wrote in the previous chapter:

`ls`	lists all files in the directory
`cat [file name]`	reads the contents a file and prints them to the screen
`touch [file name]`	create a new file
`rm [file name]`	remove a file

Be sure to also handle the exit codes some functions return and display a fitting message on the screen to notify the user when he/she's done something stupid.

To test the cat command, call `ramfs_write()` and write to a file first. You'll also be using this function in your final assignment together with `ramfs_read()` to write and read to and from files respectively.

4 Bonus points

Implement a directory structure and corresponding commands like `cd` and `mkdir`.