UNIVERSITY OF AUCKLAND
Waipapa Taumata Rau
NEW ZEALAND

ENGINEERING
DEPARTMENT OF ELECTRICAL, COMPUTER, AND SOFTWARE ENGINEERING

SOFTENG 370 2023
Operating Systems
Assignment 2 – A file system
Due Monday May 8th, 2023 11:59pm

Introduction

In this assignment you have to design your own file system. You are supplied with an interface to a device that provides very little functionality. You can read and write blocks using a block number. So the device looks like an array of blocks. You have to write information about files and directories to the device as well as the file data itself.

Objective

- To design and implement a file system on top of a simple block device.

The Device

The virtual disk device that you use to maintain your file system is really just a single file that you have access to via the `device.h` file. The device provides the bare basics on which you implement your file system. The methods you can call are:

```
int blockRead(int blockNumber, unsigned char *data);
int blockWrite(int blockNumber, unsigned char *data);
int numBlocks();
```

There are also some other definitions in that file; a constant, data types and some convenience functions. In particular the `displayBlock` function is useful for seeing what you have actually stored in a block. Read the comments in the `device.h` file for further information. Block numbers go from 0 to the number of blocks minus 1. In this assignment the block size will always be 64 bytes; you can rely on this. So you should pass unsigned char arrays of length 64 as parameters to the block read and write functions.

What you have to do

Design a file system that allows a hierarchy of directories and files in those directories. Basically the file system should operate in a similar way to a Unix (or Linux) or Windows file system.

A directory contains information about the files stored in it. File names (and directory names) can consist of between 1 and 7 printable ASCII characters including the space character, excluding "/".

Directories are separated by the "/" character.

In theory there is no restriction on fully qualified file name lengths. In reality it will depend on the size of the device and how much device space is currently being use. In particular a name could be longer than 64 bytes. e.g.

```
/direct1/direct2/direct3/direct4/direct5/direct6/direct7/direct8/file
```

Some of the space on the virtual disk device (referred to as the device from now on) must be allocated to information about the file system and what blocks are currently being used etc. You have complete freedom as to what you include in this information and the way it is to be stored on the device. The blocks holding this information on the device will be referred to as the system area of the device.

Your file system will be tested by running at least twice, once to set up the device and put some directories and files on it. The second time your file system will have to access the device with the information placed there on its previous run. Because of this, all file system information must be written back to the device. You cannot maintain file system information only in the memory of your program because the program will be shut down and the information has to be used again on its next run. Your file system is not allowed to create any other files apart from the virtual device file. All necessary information must be written to the device.

Complete the `fileSystem.c` file that implements the file system interface as defined in `fileSystem.h`. You can add as much as you want to `fileSystem.c`, you can also add extra files, but the test programs will only call the interface defined in `fileSystem.h`.

You should write some test programs which test your file system. In the next section I specify the types of tests which will be used in the marking process.

The display program (in `display.c`) can be compiled and run (read the comments).

```
gcc -Wall display.c device.c -o display
./display
```

This will work without you adding any more code. You can use it to see what the device initially holds.

Testing

In order to test your program I have supplied three test files.

The first “`test.c`” uses the CuTest unit testing framework cutest.sourceforge.net (a really small and simple library to unit test C code). You need to have the `main.c`, `test.c`, `CuTest.c` and `CuTest.h` files in the same directory as your assignment files.

If you want to add your own tests in the file `test.c` or in another file you need to recreate the `main.c` file which runs them. There is a simple shell script which generates this file for you. Execute `make-tests.sh` and redirect the output into `main.c`. Then compile and run `main.c`.

```
gcc -Wall main.c CuTest.c test.c fileSystem.c device.c -o test
./test
```

The other two test files do not use the unit testing framework but are designed to be run one after the other.

`beforeTest.c` writes some files and directories to the device.

```
gcc -Wall beforeTest.c fileSystem.c device.c -o before
./before
```

This should produce the output:

```
/:
fileA:     6
fileB:     6
dir1:      xx     (xx can be whatever size your directory is)


/dir1:
fileA:     7
```

`afterTest.c` reads the data to see if it was put there correctly

```
gcc -Wall afterTest.c fileSystem.c device.c -o after
./after
```

This should produce the output:

```
Data from /fileA: aaaaa

Data from /dir1/fileA: 1a1a1a

/:

fileA:     6

fileB:     6

dir1:      xx     (xx can be whatever size your directory is)


/dir1:

fileA:     7
```

Limits

You may assume the following limits on your file system:

Volume name maximum size - 63 single byte characters

File or directory name maximum size - 7 single byte characters

File or directory size - limited by available blocks

File pathname size - limited by available blocks

Number of blocks on the device - between 8 and 1024 (you may change this in `device.c` during your testing)

Environment

You should be able to develop your solution on most versions of Linux. The markers will use the Ubuntu 20.04 image on flexit.auckland.ac.nz so you must test on this system before submitting.

The given tests (and all of the unseen tests) check for correct output given correct input except as specified below. Concentrate on correct functionality without worrying about checking for bad input data. But it is possible that the correct result from a function call should be -1 with the file or device error number set.

Do worry about

The value for `NUMBLOCKS` hidden in the `device.c` file will be changed for some of the marking runs to be bigger than 16. Make sure your solution works with different numbers of blocks on the device.

Don't worry about

All file and directory names will be valid names (but the files may not exist yet).

There are lots of possible errors which can occur in file systems. The tests used by the markers will not test all possible errors, see the top of `device.h` and `fileSystem.h` for a list of the errors which may be checked for. In general the tests will be looking for correct behaviour for correct input. If you are worried about particular errors feel free to ask me about them.

Hints

Do not start coding your file system until you have the design worked out carefully.

There may be two different versions of file system information, those on disk and those in memory. Of course the two have to be consistent but some care in designing these will make your life much easier.

When designing your file system you may want to consider having a linked list of blocks for each file, directory and even the system area. This can be done using an integer in each block representing the next block in the list.

Questions

1. (5 marks)

Precisely describe the format of the system area of the device in your file system. Detail what each field is and how long it is. If your system area grows as the number of files or directories increases describe how and when it grows. Show output from the `display` program to explain your answer.

e.g.

```
* The first block of the system area (referred to as MFT_FIRST_BLOCK)
has the following format:
* ---------- mft block -------------
* system ID                (8 bytes)
* mft size in blocks       (4 bytes)
* free blocks vector       (64 bytes)
* first mft record …
```

2. (3 marks)

Precisely describe a directory in your file system. Detail what each field is and how long it is. If a directory grows as the number of files or directories it contains increases describe how and when it grows and how the directory blocks can be found. Show output from the `display` program to explain your answer.

3. (2 marks)

Some implementations will store information about each file solely in the directory it is stored in, others will use the system area as well? Explain which of these approaches you took and why you did it that way.

4. (2 marks)

Explain what would happen in your file system after a number of files have been created by a program and then the program crashes before executing the `atexit` handler `removeDevice` in `device.c`. In particular explain what would happen when the file system was reconnected.

5. (2 marks)

The block size for the assignment was set at 64. Explain what changes you would have to make to *your solution* in order to use a different block size, such as 32 or 4096. If your solution wouldn't need any changes then explain how it copes with different block sizes.

Helpful hints

For those of you who are still becoming C programmers:

```
man memcpy
man strchr
man 3 string
man 3 dirname
man 3 basename
man memset
man bcopy
man sprintf
```

Submission

There are two submissions. Submit `fileSystem.c` (and any other NEW files you have created) in "Assignment 2". Enter your answers to the questions in "Assignment 2 Questions".