Systems Programming Assessed Component 3

Percentage of final grade: 50%

Date set: 24th November 2016

Submission date and time deadline: 9pm on Thursday 22nd December 2016

Introduction

You should use your solution for the exercises from week 9 as your starting point for this assignment.

So far, we have been adding all our code to our kernel, but we are getting to the point where we need to start to separate out the kernel from other parts of the operating system. This means that we need to be able to load other components from the disk, as well as being able to load user applications.

You saw the implementation of the floppy disk driver in week 9. Now we need code to read and navigate the file system.

Task

There are two parts to this task:

a) Implement routines to open, read and close files using the FAT 12 file system.
b) Implement some basic utilities to navigate around a disk formatted using the FAT 12 file system.

Part A

A commercial operating system will usually support multiple file systems and multiple devices. However, for UODos, you only need to support the boot device that is formatted using the FAT 12 file system (i.e. the .img file you have been using so far). Of course, in a real operating system, you would also need to implement routines to write to the disk. However, since we are only interested in loading files at this stage, we will ignore the functionality needed to write to the disk.

Full details of the functions you need to write are provided in Appendix 1.

Part B

When you have implemented Part A, you need to implement the following utility commands that will use the functions from Part A and demonstrate that your implementation of Part A works. These should all be implemented as part of the command processor you have been working on in weeks 8 and 9.

READ filename

This function should read a text file from disk and display the contents of the file on the screen. You must be able to specify a normal DOS-style file path for the filename, including sub-directories, e.g.

```
READ \subdir\testdoc.txt
```

would display the contents of testdoc.txt that resides in the sub-directory subdir.

CD dir

Change to a particular directory on the disk image and make it the current directory for future disk operations. The CD command should accept any valid path, including . and ..

PWD

Display the current directory (as navigated to using CD). If the current directory is the root directory, then PWD should just display '\'.

DIR

Display the names of all of the files and directories in the current directory.

Testing

With your submission, you should submit a document in .doc, .docx or .pdf format that details the testing that you have done on your solutions to Parts A and B. You should clearly state your rationale behind the approach you have taken to your testing. In particular, if part of your submission is not completely functional, you should detail the problems you have encountered and the progress you have made towards solving them.

In order to test your submission, you will need to write test files to the disk image. A batch file (copytestfiles.bat) has been provided to you that will mount and unmount the disk image. You can edit this file to copy test files to the disk image.

Submission

You must submit a zip file containing your submission to submission point "Assessment 3 Submission Point" by the due date and time. Any submission that Course Resources indicates as being submitted late will not be marked and a grade of 0 given for this component.

The zip file must contain:

- The complete folder for the submission, including the files you used as a starting point.
- The folder must be cleaned before you create the zip file. This means that it should be as if you had run 'make clean' on the folder. Your submission will be tested by using make to build the disk image containing your code and then it will be executed in Bochs (using the .img file)
- Your testing document.

The name of the zip file should be Assessment3_xxxxxxxxx.zip where xxxxxxxxx is your student number. For example, if your student number was 123456789, your zip file would be called Assessment3_123456789.zip. Any file that is not named using this convention or if something other than a zip file is submitted will result in the assessment not being marked and a grade of 0 for this assessed component.

Hints

You might find the following hints useful while working on this assessment:

1. The keyboard driver you were previously given does not completely handle UK keyboards correctly. In particular, the backslash (\) key does not work correctly. When entering path names in your command processor, press the hash (#) key to enter a backslash.
2. Remember that sub-directories on the FAT file system are implemented as files of directory entries, 16 entries in each sector.
3. The first two entries in each sub-directory are the special entries '.' and '..'. These contain references to the current directory and the parent directory respectively.
4. Path-names are case-insensitive.
5. On FAT 12 systems, a cluster is one sector in size (i.e. 512 bytes).
6. In week 4, you were provided with assembler code that read a FAT 12 formatted disk. That code only read the root directory, not sub-directories, but rereading this code (and the notes from weeks 3 and 4) will be useful to you in doing this assessment.

Appendix 1 – The Functions Required for Part A

Your code for Part A should implement the following functions:

void FsFat12_Initialise()

This function could be called from the kernel's Initialise function, after the floppy disk driver has been initialised. It should be used for any initialisation required for your file system. Suggested functionality could be:

- Read the BIOS Parameter Block from the boot sector
- Calculate the offsets (i.e. the number of sectors) to the File Allocation Table (FAT), the offset to the root directory and the offset to the data sectors
- Read the File Allocation Table into memory

FILE FsFat12_Open(const char* filename)

Search the directory for the specified filename (which could be a complete path, including sub-directories) and return a populated FILE structure (see below for details of FILE). If the file does not exist on the disk, return a FILE structure with the Flags fields set to FS_INVALID.

If the file is found, the FILE structure should be populated as follows:

CurrentCluster	Set to the number of the first cluster for the file
Position	Set to 0
Eof	Set to 0
FileLength	Set to the length of the file.
Flags	Set to FS_FILE or FS_DIRECTORY

unsigned int FsFat12_Read(PFILE file, unsigned char* buffer, unsigned int length)

Read the next block of data from the file specified by the FILE structure specified by the file parameter. The position to read should be obtained using the CurrentCluster and Position fields.

The amount to read is specified in the length parameter. The data should be copied to the buffer specified by the buffer parameter.

If end-of-file is reached during the read, then the EoF field should be set to 1.

The function should return the actual number of bytes read from the disk.

Note. If you are unable to implement the full functionality for this function, it will be acceptable for you to implement the function so that it ignores the length parameter and just reads 512 bytes at a time (i.e. one sector). You can then ignore the Position field in the FILE structure. This implementation will result in a lower grade, but it will enable you to implement the functionality required for Part B.

void FsFat12_Close(PFILE file)

Set the Eof field in the FILE structure to 1.

In order to help you, two C header files have been provided – filesystem.h and bpb.h.

filesystem.h contains definitions of the FILE structure as well as the structure for a directory entry (DirectoryEntry).

bpb.h contains definitions for the boot sector and the BIOS parameter block.