CSCI 460 Operating Systems
Fall 2015
Assignment 3

Due Date:

The assignment must be submitted by 8am Friday November 20, 2015. Students must submit their programs via the course canvas site. If multiple files are submitted, you are to pack them into a single zip file named `Wnnnnnnnn.zip` or `Wnnnnnnnn.tar.gip`; if just a single C file is submitted, it should be named `Wnnnnnnnn.c` (where `Wnnnnnnnn` is your student W-number).

The programs must be able to compile, link and run in with gcc under Linux.

Assessment

Student assignments will be assessed on correct functionality, algorithm design, data structure design and readability of the programs.

Requirements

You are to write a C Application Program Interface (API) for a file subsystem to work with a provided device driver.

Provided Files

The following files have been provided for your use in the assignment.

Driver.h	Header file specifying the interface to the device driver.
Driver.o	Object file for the device driver, 64-bit Linux version.
FileSysAPI.h	Header file for your file system, specifying the functions that you must provide
EasyTest.c	An easy test program for your file system. Note: your assignment will be tested with a far more stringent test than this.

The File System API

Your API must have the following functions, as specified in the provided file `FileSysAPI.h`:

```
int CSCI460_Format ( )
```

Requests the creation of a file system on the device.
Return value: 0 = failure, 1 = success.

This function must fail if the device driver returns an error code.

```
int CSCI460_Write ( char *FileName, int Size, char *Data)
```

Writes *Data*, an array of *Size* characters to the file *FileName*. If the file already exists in the file system, the existing file is to be deleted and a new one written with the specified data. Return value: 0 = failure, 1 = success.

This function must fail if there is no file system, if the file system is full, or if the device driver returns an error code.

Note that, for simplicity, this function writes the entire file in one function call.

```
int CSCI460_Read (  char *FileName, int MaxSize, char *Data)
```

Reads characters from the file *FileName* into the array *Data*. At most *MaxSize-1* characters will be read and a null character put at the end of the string.
Return value: 0 = failure. A non-zero return value indicates the number of characters read from file.

This function must fail if there is no file system, if the file does not exist, or if the device driver returns an error code.

Note that, for simplicity, this function reads the entire file in one function call.

```
int CSCI460_Delete(  char *Filename)
```

Deletes the specified file. Return value: 0 = failure, 1 = success.

This function must fail if there is no file system, or if the file does not exist in the file system.

The Device Driver

A device driver is supplied as `Driver.o` to provide the low-level functionality needed to support your API.

The device uses logical block addressing, with each block containing 64 bytes, as defined in the header file Driver.h :

```
#define      BYTES_PER_SECTOR        64
```

The functions in the device driver are declared in the header file `Driver.h` and are shown below:

```
int DevFormat ( )
```

Formats the device, erasing any data previously stored there.
Return value: 0 = failure, 1 = success.

This function will fail if the device driver cannot write to the device.

```
int DevWrite (    int BlockNumber, char* Data)
```

Writes the contents of *Data* to one block in the file system. The block to be written is specified by its logical block number *BlockNumber*.
Return values: 0 = failure, 1 = success.

```
int DevRead (     int BlockNumber, char* Data)
```

Reads one block of data from the file system into *Data*. The block to be read is specified by its logical block number *BlockNumber*.
Return values: 0 = failure, 1 = success.

The provided file `DriverTest.c` is used to test the device driver and to demonstrate its use.

Assumptions, Requirements and Simplifications

1. Your API will need to create and manage a directory of files in the file system. This can be a simple flat directory, rather than a tree structure or acyclic directed graph.

2. Your program will be tested with a large number of files. Therefore your directory must be represented by a dynamic data structure which enables efficient access to the file data.
3. You are to use a modified i-node approach to file block allocation. Each file starts with an i-node block which contains sixteen 32-bit numbers:
    - 13 direct block numbers
    - 1 single indirect index block containing block numbers of 16 data blocks
    - 1 double indirect index block containing block numbers of 16 index blocks
    - 1 triple indirect index block containing block numbers of 16 double indirect index blocks
4. Free space management is to be done by maintaining a linked list of free-block spans, where a “span” is a sequence of free blocks. Each entry in the linked list is to contain the logical block number of the first free block in the span and the number of (successive) free blocks in the span.
5. For the purpose of this exercise, the file directory and free-block list is not written onto the device. Instead it should exist only in the memory of your API. Please note that this implies that the life of the file system does not extend beyond the execution of an application program which uses your API.