Operating Systems - Laboration 3

File Systems

Daniel Lagula

DV1628

Design Decisions

- A decision was made to instruct the compiler to pack the struct dir_entry. The reason is to avoid padding of int8 and int16 members, otherwise the struct may require 68 bytes on disk instead of 64, and we will in that case lose 4 file entries per directory.

This can be accomplished by using a #pragma pack directive:

```
#pragma pack(push, 1)
struct dir_entry {
    char file_name[MAX_FILE_NAME_LENGTH + 1]; // name of the file / sub-directory
    uint16_t first_blk; // index in the FAT for the first block of the file
    uint32_t size; // size of the file in bytes
    uint8_t type; // directory (1) or file (0)
    uint8_t access_rights; // read (0x04), write (0x02), execute (0x01)
};
#pragma pack(pop)
```

This means that we can store a maximum of 64 files in one directory, for example in the root directory.

- The file_name field in dir_entry is 56 characters long. In order to avoid storing another field to keep track of the name length a decision was made to store the name as a null-terminated string. This means that the maximum length is 55.

- A constant was used to indicate the maximum length of a file and directory name. And a constant variable defines how many files and directories each directory can contain:

```
#define MAX_FILE_NAME_LENGTH 55

const unsigned MAX_DIR_ENTRIES = (BLOCK_SIZE / sizeof(dir_entry));
```

- Another design decision was to use helper methods to read the FAT into memory, and to write the FAT to disk. Read FAT will also assure that the disk is already formatted (has already been formatted by calling the method `format()`).

```
// read FAT into memory
int readFAT(bool assumeFormatted);
// write FAT from memory to disk
int writeFAT();
```

- To keep track of the current directory, a variable called current_block had to be stored in memory. It can be changed by using the command `cd(dirpath)`.

```
uint16_t current_block = ROOT_BLOCK;
```

- Most file operations include reading a directory and to either find an existing file entry or a free entry for a new file.
  Because of this, a helper struct was defined in fs.h ("dir_info"), which will store all directory entries in an array. It will also store the index in the array to the new/existing file entry, and finally the actual disk block where the file resides:

```
// Helper struct to keep info about a file in a directory
struct dir_info {
    int block;  // disk block number where the dir 'entries' are stored
    int index;  // index to actual dir_entry in the 'entries' array
    dir_entry entries[MAX_DIR_ENTRIES]; // all directory entries in a block
};
```

- The commonly used struct dir_info will be filled with data by the helper method called '`findFileEntry()`'.

```
// find a file entry in the directory. Either find an existing file,
// or a free entry for a new file (note: check for duplicate filenames).
int findFileEntry(std::string filepath,
                  uint8_t newOrExisting,
                  dir_info &dir,
                  uint8_t access_rights);
```

- In order to handle absolute and relative file paths, another helper method ('getDirBlock') was used to traverse the directory tree in order to find the correct directory block.

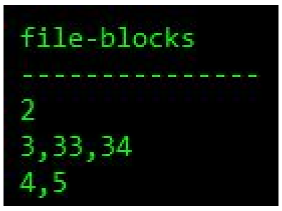
```
// find the directory block where the given file resides
// (note: filepath may contain a relative or absolute path)
int getDirBlock(std::string filepath, int &dir_block);
```

- When using mv or cp to move/rename/copy a file, a decision was made that the user must either give the destination name of the file, or simply use the Linux standard "." in order to use the existing name of the source file. For instance - the following commands will give the same results:

```
$ mv file1 dir1/.
$ mv file1 dir1/file1
```

```
std::string dest_name = getFileName(destpath);
if (dest_name == ".") {
    dest_name = getFileName(sourcepath);
    destpath = destpath.substr(0,destpath.length()-1) + dest_name;
}
```

- In order to make it easier to test the program a decision was made to print another column in the "ls" output which will show the used FAT blocks for each file/dir.



```
file-blocks
---------------
2
3,33,34
4,5
```