NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

FS implementation

LAB 9

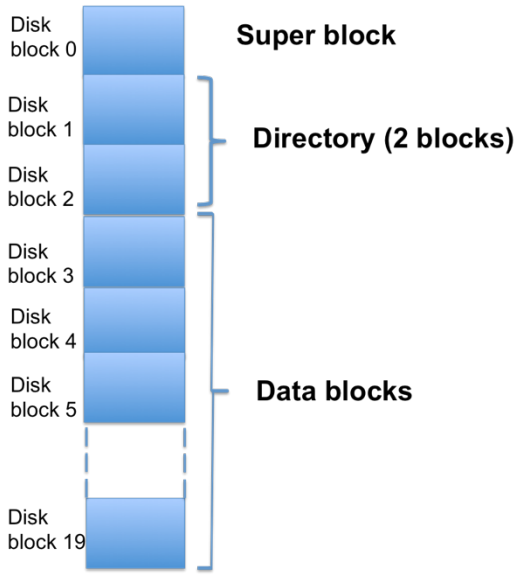
Lab objectives

To consolidate the knowledge on how a *file system* (*FS*) works via the implementation of the listing of a directory's contents of a very simple file system.

File system format

This lab uses a very simple FS that is stored in a *volume,* like a disk, but simulated by a *file* where reading or writing *disk blocks* corresponds to reading or writing *fixed sized data chunks* from or to that file. Therefore, all read or write operations start at file's offset that are multiple of the disk block's size.

The *FS format* contains a *super-block* describing the disk's organization, one directory that is also equivalent to the inodes table containing meta-data of the files and the occupied blocks. The remaining blocks are used for the data blocks (either in use or free). Figure 1 shows an overview of a particular disk with *20 blocks* and two blocks for the directory table, after being initialized with the *format* command.



The file system layout

For this *FS* the disk is organized in 4K blocks. A block may either contain the superblock (block 0), the directory/inodes table (block 1 and the next ones accordingly to information in superblock), or data:

- Block 0 is the *super-block* and describes the disk organization (each field is 32bits wide):

```
struct fs_superblock {
    uint32_t magic;      // magic number is there when formated
    uint32_t nblocks;    // fs size in number of blocks
    uint32_t ndirblocks; // number of blocks for directory
    uint32_t ndirents;   // max number of dir entries
};
```

  - The first field must be initialized with a "magic number" (**0x00f00baa**). It is used to verify if the disk contains a valid *FS*
  - The second value is also an unsigned integer defining the size of the disk in blocks (*nblocks*)
  - The third value defines the number of blocks for the directory/inodes table.

- The fourth value defines the maximum number of directory entries (if using all ndirblocks).

- Block 1 and possibly the following ones have the directory/inodes table. Each directory entry is also like an inode and uses 128 bytes containing the following information:

```
struct fs_dirent {
    uint8_t isvalid;
    char name[MAXFILENAME];
    uint32_t size;
    uint32_t blk[POINTERS_PER_ENT];
};
```

  - A validation byte defining if the inode contains valid information (1) or not (0)
  - Name, a C string (63 bytes maximum) with the file's name
  - Size, the current file size
  - File blocks, (unsigned int) pointers to the file's blocks (15 max). Those blocks are used for storing the contents of the file

- The used/free block bit map or similar data structure, to register the used blocks, is not in disk. It could be built in memory when mounting the FS from the directory entries. For this exercise we will not need it.

File system operations

The file *fs.c* implements the operations that manipulate the file system (notice that the dirent number is used like a file descriptor):

```
void fs_debug();

int  fs_format();

int  fs_mount();

int  fs_open( char *name);

int  fs_read( int inode, char *data, int length, int offset );
```

Work to do – showing the contents of the directory

Download the Zip archive from CLIP with the source code.

Complete the implementation of the `fs_debug` function, the listing of all files in the FS. Implement as well, the `fs_open` and the `dirent_load` operations, so that the contents of a file can be read:

void fs_debug()

Displays information about the current active disk. The expected output is something like:

```
superblock:
    20 blocks
    2 dir/inode blocks
    64 inodes/dirents
*************************************
inode size  name  blocks
0     146   foo   3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*************************************
```

This should work either the disk is mounted or not. If the disk does not contain a valid file system, i.e. the magic number is not present at the beginning of the super block, the command should print *"Non-valid filesystem"* and return immediately.

int fs_open(char *name)

Searches an entry on the directory for the named file. Returns its entry number on success or, if not found, -1.

void dirent_load(int numde, struct fs_dirent *ino)

Fill ino struct with information in the dirent number 'numde'. Must calculate the block number where this dirent number is, and its position in that block.

The functions above are in file *fs.c*. ***It is the only file that you need to modify.***

A Shell to test the FS

A shell to manipulate the file system is available and can be used as in the examples below:

```
$ ./fslab9 image.20
```

where the argument is the name of the file (disk) with the file system. Note: a second numeric argument can be given for erasing and creating the file that simulates the disk, with that number of blocks.

One of the commands is *help*:

```
>> help
Commands are:
    format
    mount
    debug
    cat <name>
    help
    exit
```

The commands *format*, *mount*, *debug* correspond to the functions with the similar name previously described or that you can see in the source code. Do not forget that a file system must be formatted before being mounted and used, but an example (image.20) is already provided.

The *cat* command shows the content of a file and will work as soon as you implement the open and dirent_load operations. Example with the provided disk:

```
>> cat foo
 It works!


 It works!
========================


 It works!


================
END
146 bytes copied
```

Bibliography

[1] Sections about persistence (chapters 36 and 39) of the recommended book, “Operating Systems: Three Easy Pieces” Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau”

[2] http://pages.cs.wisc.edu/~remzi/OSTEP/file-intro.pdf

[3] Slides from FSO classes