BSCS-8C

File System Design

Operating Systems

Abdullah Hassan Chaudhry 266639, Ahmad Jarrar 249423
3-1-2021

Directory Structure

Our File System operates on an emulated hard disk of size 16 kilobytes. The hard disk is emulated by a single file on disk, “data.dat”.

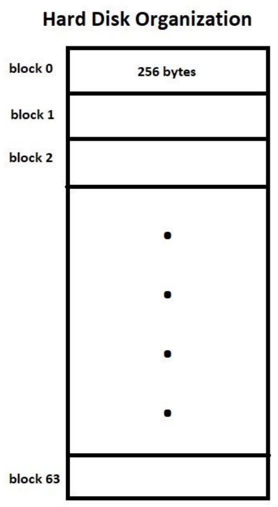
We have designed a hierarchical storage structure in which every file is stored in a directory which are also contained in some directories. A default “root” directory is created, and everything is a successor of this root directory.

To understand how the files are stored we need to understand these concepts.

Key Concepts

Blocks

We have divided our disk space into fixed sized blocks simply referred as blocks. These blocks are 256 Bytes each. This is the minimum allocation size, meaning any file (or directory) created on the hard disk is allocated an integral number of blocks.



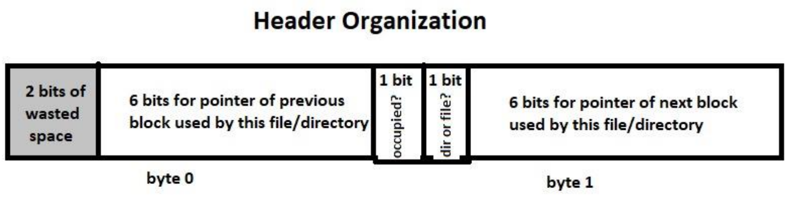
Headers

Each block has a header which takes up first 2 bytes of each block. It contains key attributes of the block.

It contains:

- Address of next block
- Address of previous block
- Is part of directory
- Is occupied

As we have fixed our disk space to 16Kb, we have 64 blocks, so the addresses (next and previous block) take 6 bits each. Is_dir and Is_occupied are Boolean flags and take up 1 bit each. These 4 values are stored in header as follows:


Header Organization
2 bits of wasted space
6 bits for pointer of previous block used by this file/directory
1 bit
occupied?
1 bit
dir or file?
6 bits for pointer of next block used by this file/directory
byte 0
byte 1

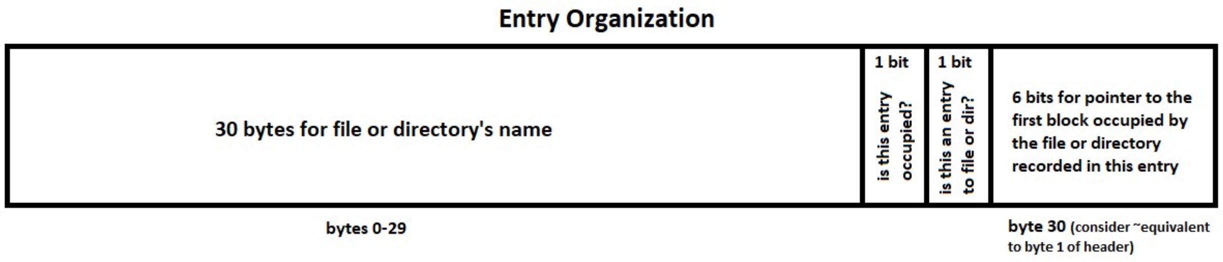
Entries

Entries are what define a file in directory. They contain important information to search for a file in directory. An entry takes up 31 bytes to store.

Entry contains:

- File name - upto 30 characters long
- Address of the first block of file
- Is a directory
- Is occupied

Entry is stored as follows:

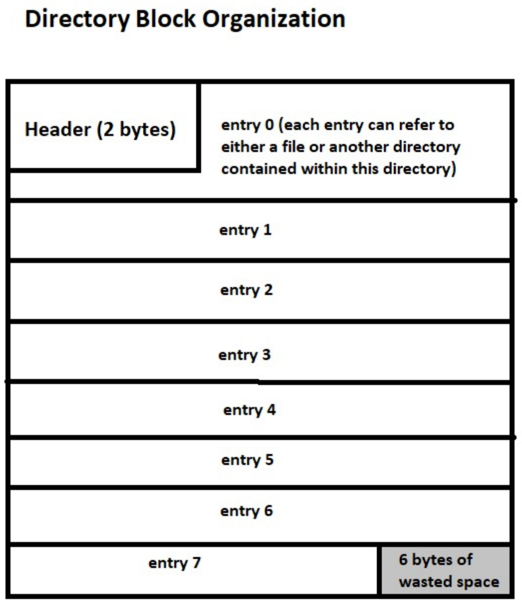

Entry Organization
30 bytes for file or directory's name
1 bit
is this entry occupied?
1 bit
is this an entry to file or dir?
6 bits for pointer to the first block occupied by the file or directory recorded in this entry
bytes 0-29
byte 30 (consider ~equivalent to byte 1 of header)

Directory

This is a special type of file which contains information of where other files are stored. It consists of Entries which point to files or directories that are part of this directory.

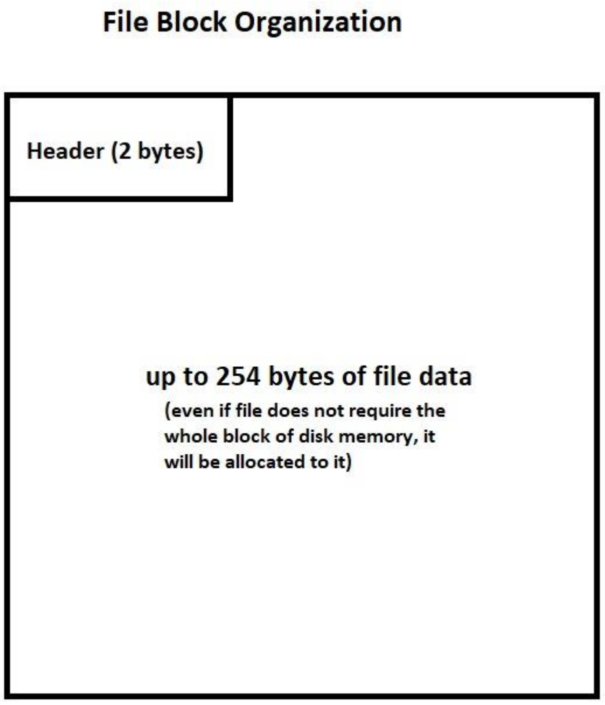
First Entry of a directory always points to its parent directory.

Below is an example of a block of memory representing a directory.


Directory Block Organization
Header (2 bytes)
entry 0 (each entry can refer to either a file or another directory contained within this directory)
entry 1
entry 2
entry 3
entry 4
entry 5
entry 6
entry 7
6 bytes of wasted space

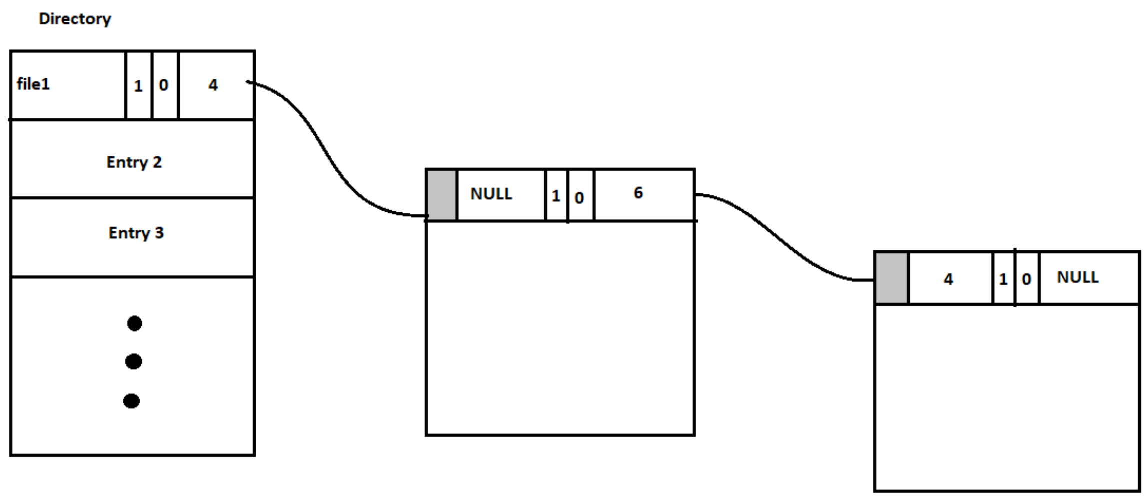
Files

Files are basic storage structure provided to the user. It can store any type of information.


File Block Organization
Header (2 bytes)
up to 254 bytes of file data
(even if file does not require the whole block of disk memory, it will be allocated to it)

Storage Strategy

As mentioned earlier a hierarchical model is used for the storage in which we maintain a record of the first blocks of files in the directory forming a tree. In our model any file can take up as many blocks of memory as it requires so size of file is not limited except for the memory itself. In case a file spans over multiple blocks, we contain the address of next and previous block of that file in the header of blocks forming a doubly linked list.



Build and Run instructions

A make file is included in the program which is used to build the program or clean the folder.

Use **make clean** to clean the folder of all binary and data files.

Use **make** to build the binaries and executable.

Use **./filesystem** to run the command line interface of the program.

Files system can also be accessed through other programs by including 'fileSystem.h' and creating a FileSystem object.

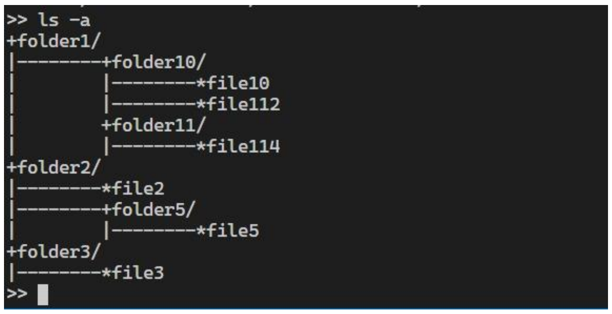
Interface

Command	Optional flags	Usage	Description
ls	-a	Ls [flags]	Lists The files and folders in current directory. Use '-a' to view all sub dirs.
mkdir		mkdir [name]	Creates a folder in the current directory.
mkfile		mkfile [name]	Creates a file in the current directory.
view		view	View Stats of the File System.
cd		cd [path]	Change current directory.
rm	-r	rm [flags] [name]	Remove File/folder from current directory. Use '-r' to delete recursively.
mv		mv [source] [dest]	Move File/Folder from source to destination.
pwd		pwd	Print present working directory.
map		map [name]	Show which blocks in memory a file/folder occupies.
open		open [file_name]	Open file to read/write.
close		close	Close currently open file.
read	[start] [size]	read [flags]	Read entire opened file. Use options to read from an offset or limit file size.
write	[start] [size] -s [content]	write [flags]	Write to opened file. It overwrites existing file.
Append	-s [contents]	append [flags]	Write to opened file. It writes to the end of existing file.
trunc		trunc [size]	Truncate opened file to size.

* **Note:** Manual can also be accessed from within the program using ‘man’ command

Example

An example directory structure



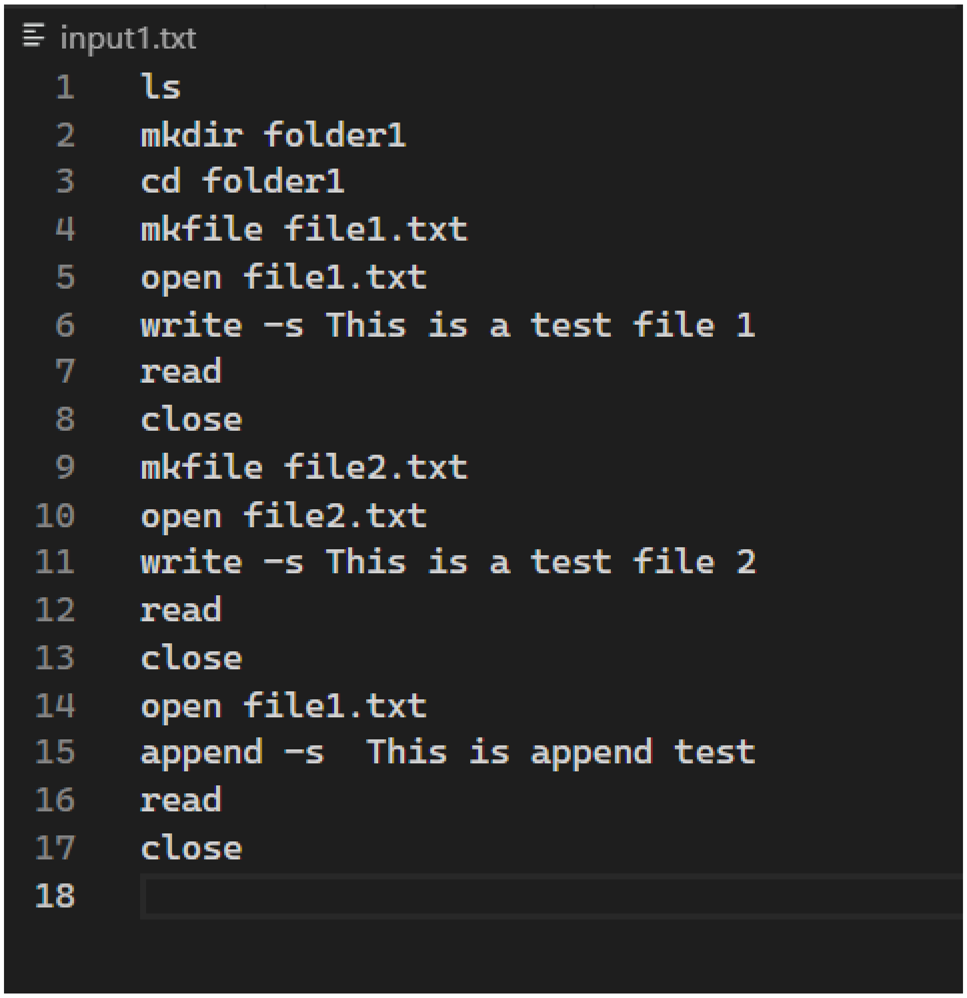
```
>> ls -a
+folder1/
|--------+folder10/
|        |--------*file10
|        |--------*file112
|        +folder11/
|        |--------*file114
+folder2/
|--------*file2
|--------+folder5/
|        |--------*file5
+folder3/
|--------*file3
>> 
```

Multiple Threads

We have added support for multiple users in our operating system.

To use our program in multithreaded mode users need to provide scripts in text files with one command in one line.

E.g.



input1.txt

```
ls
mkdir folder1
cd folder1
mkfile file1.txt
open file1.txt
write -s This is a test file 1
read
close
mkfile file2.txt
open file2.txt
write -s This is a test file 2
read
close
open file1.txt
append -s  This is append test
read
close
```

Up to 10 script files can be provided at a time.

For multithreaded mode pass the script files as command line arguments.

To run scripts in 3 files script1.txt, script2.txt and script3.txt

Use: **./filesystem script1.txt script2.txt script3.txt**

Output of the scripts will be written in new text files generated by the program which will be named as out_script_name.txt.

**7 sample scripts have been provided with the program.*

File System Over Network

We have provided server and client applications to emulate many users interacting with a centralized filesystem.

Protocol

We have designed a simple protocol which sends the input and outputs in packets of 250 bytes. These messages end with special End of Transmission (EOT) character. Server and clients consider the messages as one until this special character is received which allows for variable length of transmissions.

In addition, client sends End of Medium (EOM) character when exiting which terminates its thread on server side.