BSCS-8C

File System Design

Operating Systems

Abdullah Hassan Chaudhry 266639, Ahmad Jarrar 249423
1-19-2021

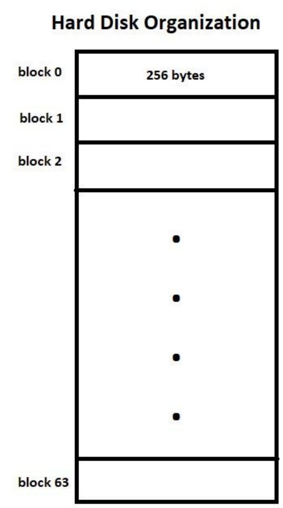
Directory Structure

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

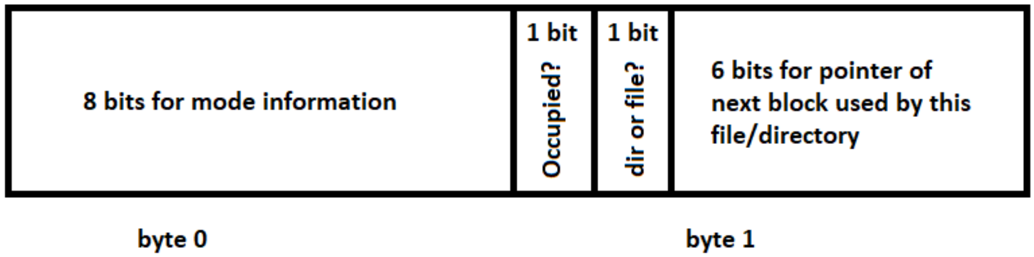
Each block has a header which takes up first 2 bytes of each block. It contains key attributes of the block.

It contains:

- Address of next block
- Mode information
- Is part of directory?
- Is occupied?

As we have fixed our disk space to 16Kb, we have 64 blocks, so the addresses take 6 bits. Is_dir and Is_occupied are Boolean flags and take up 1 bit each. Mode is a 8 bit number contains information of whether file is opened and if opened, in which mode. These 4 values are stored in header as follows:

Header Organization



Entries

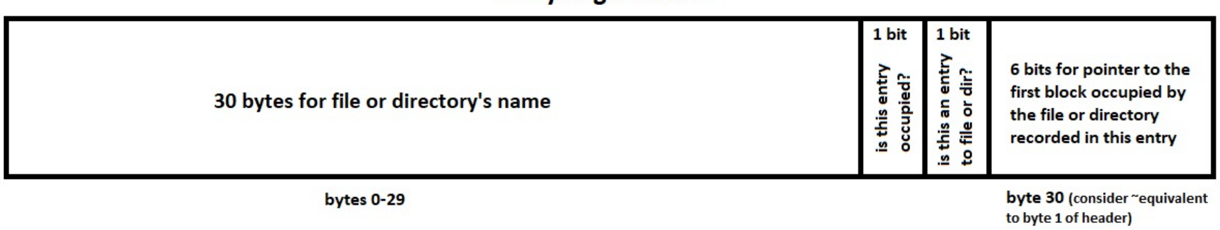
Entries are what define a file in directory. They contain important information to search for a file in directory. An entry takes up 31 bytes to store.

Entry contains:

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

Entry is stored as follows:

Entry Organization

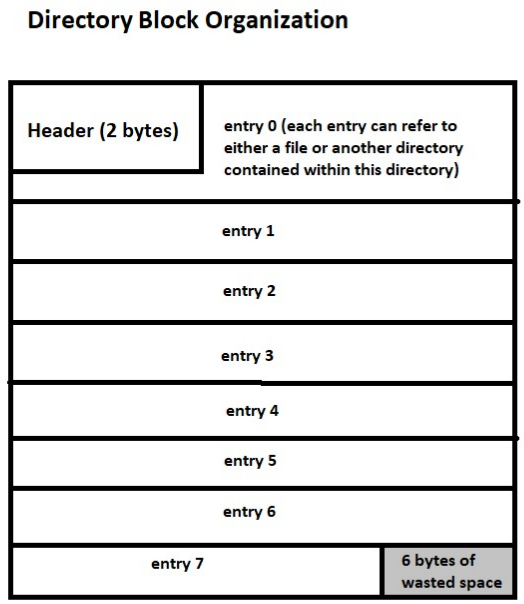


Directory

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.

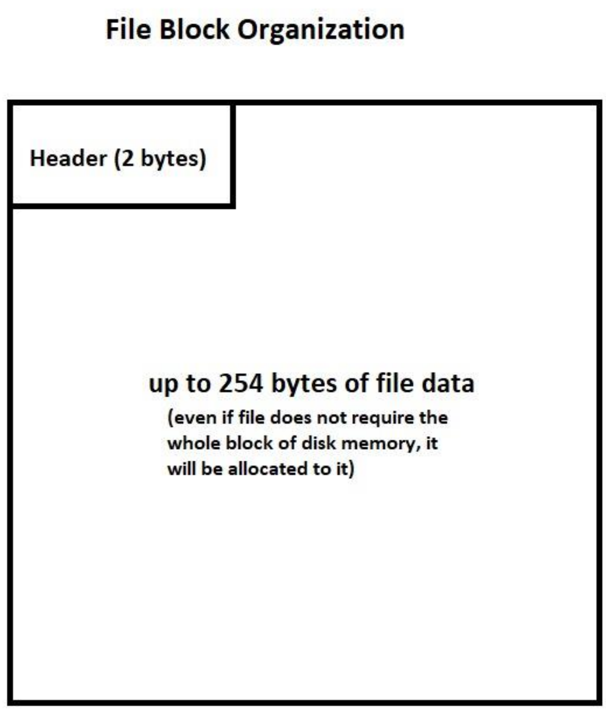
First Entry of a directory always points to its parent directory.

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



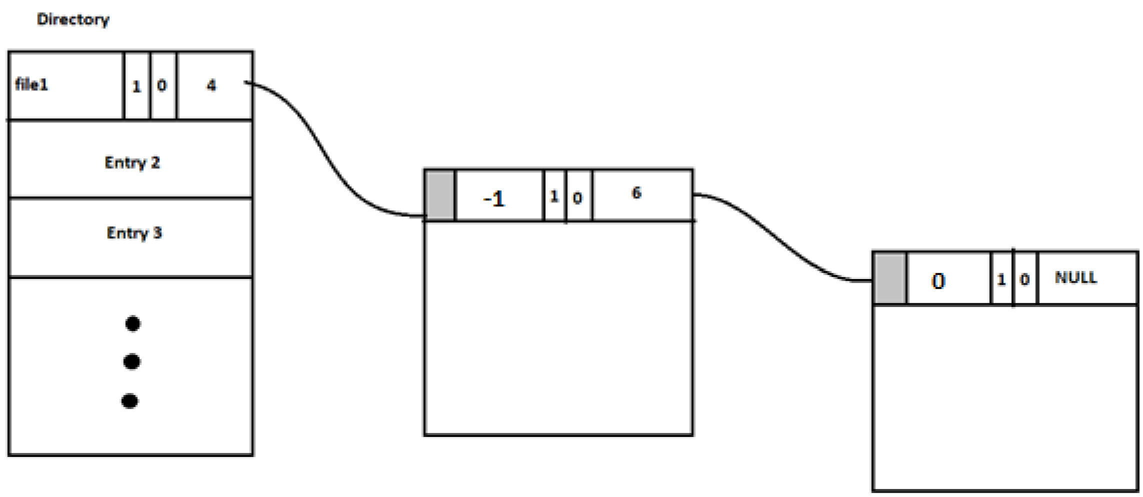
Files

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



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 block of that file in the header of blocks forming a 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	Optionas/flags	Usage	Description
ls	-a	ls [options]	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	-r -w	open [file_name]	Open file to read/write.
close		close	Close currently open file.
read	[start] [size]	read [options]	Read entire opened file. Use options to read from an offset or limit file size.
write	[start] [size]	write [options] -s [content]	Write to opened file. It overwrites existing file.
mvwf		mvwf [start] [size] [target]	Copies the specified bytes to the target.
Append		append [options] -s [content]	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.

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.

Thread Security

As our server is multithread capable, we need to make our operations thread safe to avoid corruption of file system.

We achieve this using two strategies.

File Modes

Files can be opened in two levels of permissions. Files can be opened in read only mode in which user cannot modify the content of the file. This mode allows multiple users to read the file at the same time. We maintain the number of users reading the file in the header of the first block of the file.

Write mode allows both read and write operations. File needs to be opened in this mode to be modified. This mode restricts the access to the file to one user only. In addition, user cannot open file in write mode if someone else has already opened it.

Directories only have a single mode unused/used. When some file is open in any mode all the folders in path from file to root directory are marked as used. This prevents deleting or moving any of these folders.

Similarly delete and move operations are disabled for the opened file.

Mode information is stored in the first header of the file/directory and takes up 1 byte. It is a signed number in which -1 represents that file has been opened in write mode. When opened in read mode the number is incremented which means 127 users can open a file in read mode at a time. Conversely when it is closed the number is decremented.

When the file is not used it has mode value 0.

Locks

We have used 3 mutex locks to synchronize entire file system.

Fist one is used to make the access to the physical file atomic operations. This makes sure that the there are no errors when OS tries to seek to different parts of the file as we need random I/O. Our file operations are designed in a way that we need access to physical file for very small proportion of the time needed by the functions so locking does not affect performance for small number of users at a time. This feature essentially emulates single read/write head on disk.

Second lock is used to make our block allocation atomic. It is the biggest potential failure point where data in our file system can corrupt, if 2 or more files are allocated the same block in memory.

Last lock is used to make the change of file mode operations (get_mode, set_mode, clear_mode) atomic. This lock ensures that information about a file's mode is up to date and stable while the mode values are being changed.