Design and Implementation of a Trivial Filesystem on a Regular File of the Underlying Operating System

Opened: Monday, 1 April 2024, 12:00 AM
Due: Sunday, 7 April 2024, 11:00 PM

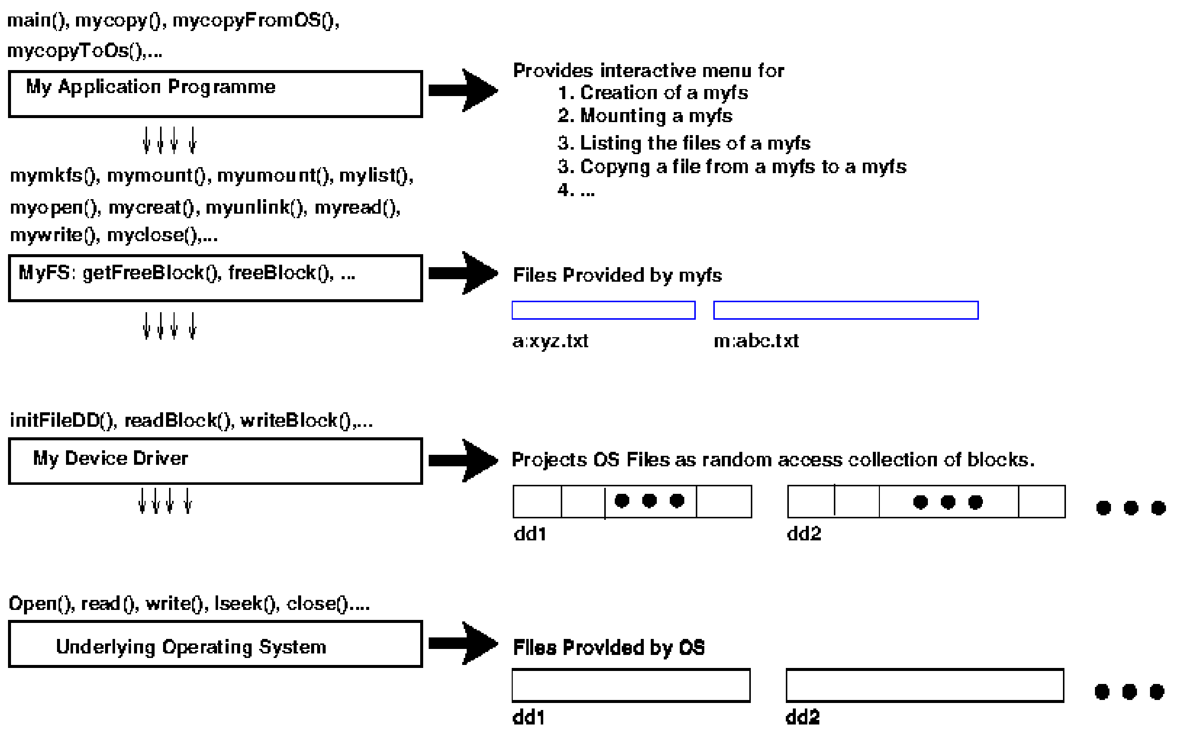
Mark as done

The present assignment pertains to "design" and "implementation" of a "filesystem" (henceforth, referred as "**myfs**") on files provided by the underlying operating system (which can be accessed by application programs through system calls like (i) open(), (ii) read(), (iii) write, (iv) lseek, (v) close(), etc.)

Operating system allows creation of supported filesystems (fat16, fat32, vfat, ntfs, ext4, etc.) on **block devices**, that is, devices which are projected as "**random access collection of blocks**" by their device drivers. The study material in the previous page titled "**Filesystem Fundamentals**" highlights the layered architecture for implementation of filesystems.

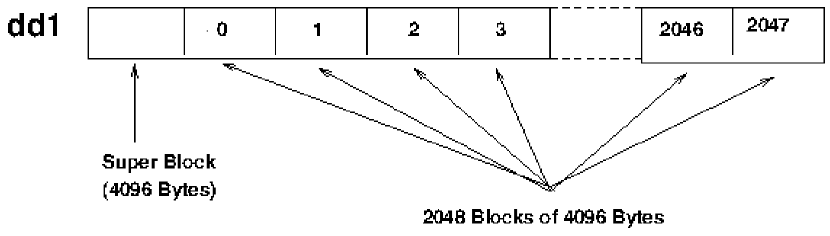
In a nutshell, under the present assignment, you are supposed to write an application program, which, at one end, makes calls to (i) open(), (ii) read(), (iii) write, (iv) lseek, (v) close(), etc. to operate on OS files, and on the other hand, provide functions like (i) myopen(), (ii) myread(), (iii) mywrite, (iv) mylseek, (v) myclose(), etc., using which other programs can operate on the files of the filesystem created on **myfs**. Henceforth, these types of files will be called as "**myfile**".

Following figure explains the idea in more detail.

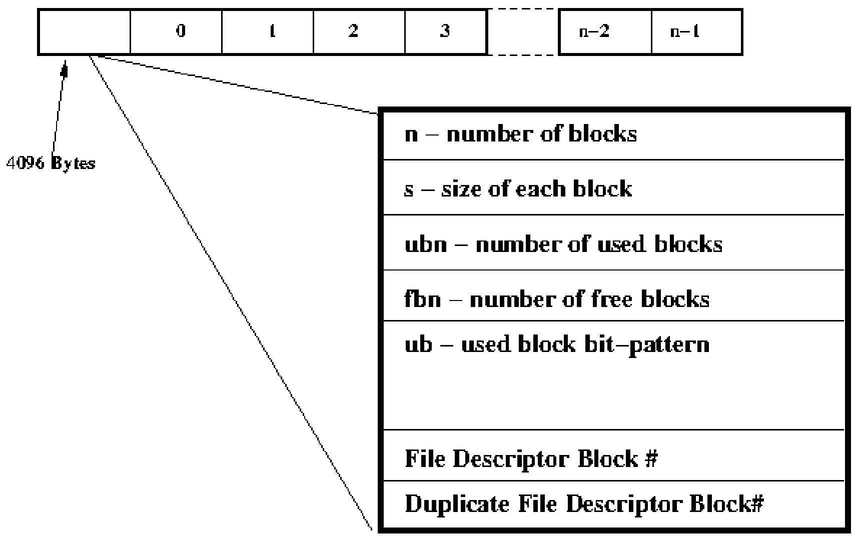

main(), mycopy(), mycopyFromOS(), mycopyToOs(),...
My Application Programme
Provides interactive menu for
1. Creation of a myfs
2. Mounting a myfs
3. Listing the files of a myfs
3. Copyng a file from a myfs to a myfs
4. ...
mymkfs(), mymount(), myumount(), mylist(), myopen(), mycreat(), myunlink(), myread(), mywrite(), myclose(),...
MyFS: getFreeBlock(), freeBlock(), ...
Files Provided by myfs
a:xyz.txt
m:abc.txt
initFileDD(), readBlock(), writeBlock(),...
My Device Driver
Projects OS Files as random access collection of blocks.
dd1
dd2
Open(), read(), write(), lseek(), close()....
Underlying Operating System
Files Provided by OS
dd1
dd2

Please note that we can write functions, through which, an OS file can be conceived as a collection of random access blocks of fixed size. For example, a file named "**dd1**" can be conceived as a collection of **2048** (**n** = 2048) blocks, where each block is of **4096** (**s** = 4096) bytes. Let, the **n** blocks of the file **dd1** are numbered as **0** through **n-1**.

The file, then, can be conceived as shown in the following figure.



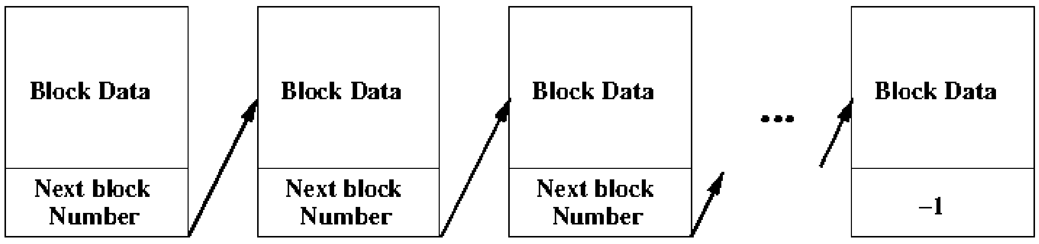
Note that the first 4096 Bytes (Labelled as Super Block) is kept for storing information like **n** (total number of blocks), **s** (size of each block), etc.


0
1
2
3
n−2
n−1
4096 Bytes
n – number of blocks
s – size of each block
ubn – number of used blocks
fbn – number of free blocks
ub – used block bit–pattern
File Descriptor Block #
Duplicate File Descriptor Block#

Subsequently, for **myfs** layer, we would also store **ubn** (number of used blocks), **fbn** (number of free blocks), **ub** (a bitmap showing which blocks are used), etc. in this block. Please note that, in **ub** there should be **n** number of bits, one bit for each block in the file (eg, **dd1**). An 1 (one) in a bit of **ub** reflects that the corresponding block is in used state. Whereas, a 0 (zero) in a bit of **ub** reflects that the corresponding block is in free state. The superblock also stores the block number of the block where file descriptors are saved.

Now, the contents of a **myfile** can spread across multiple blocks and those blocks can be "organized" in the "disk" (here the OS file, **dd1**, say) in various ways.

For the present assignment you adopt the organization as shown in the following figure.

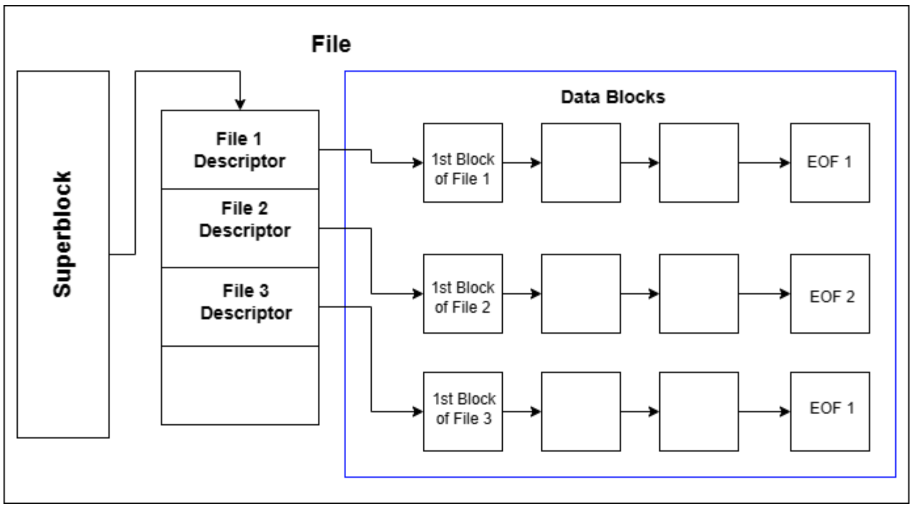


Please note that at the end of each block of the file, the block number of the next block is stored with the exception that at the of the last block of the chain -1 is stored (to mark that there is no next block).

The file descriptor of a **myfile** stores the block number of the 1st block of the file.

Let myfs be a flat filesystem, i.e., it does not support the idea of foler/directory.

The overall organization of myfs is shown in the following figure.



A File Descriptor would contain:

Content type	Minimum Size	Purpose
File name	20 Characters	Stores the name of the respective **myfile**
First Block number	3 Bytes	Stores the block number of the 1st block of the respective **myfile**
Size	4 Bytes	Stores the size of the respective **myfile**
Modification Time	32 Bytes	Stores the time of last modification in terms of epoch time

On the basis of the above model, you first write the following functions.

Please note that the parameters and return types of these functions are representative only. Make your own assumptions wrt their signature (name, parameters, return type) of these functions and how exactly they should perform their tasks.

1. "***int init_File_dd(const char *fname, int bsize, int bno)***" that creates a file (if it does not exist) called **fname** of appropriate size (**4096 + bsize*****bno** Bytes) in your folder. The function initializes the first 4096 Byes (the Super Block) putting proper values for **n**, **s**, **ubn**, **fbn**, and **ub**. If for some reason this function fails, it returns -1. Otherwise it returns 0.
2. "***int read_block(const char *fname, int bno, char *buffer)***" - reads **bno** block number of the file ***fname*** and stores in *buffer.*
3. "***int write_block(const char *fname, int bno, char *buffer)***" - writes **bno** block number of the file ***fname*** from ***buffer***.
4. "***int get_freeblock(const char *fname***" that reads the 1st 4096 Bytes (the Super Block, containing **n, s, ubn, fbn, ub**) from the file ***fname*** and finds and returns the block-number of the 1st free block (starting from the 0th bloc) in that file. This function sets the corresponding bit in **ub** and fills up the free block with 1's. **ubn** and **fbn** too are modified appropriately. On failure this function returns -1. Please note that all the modifications done by this function have to be written back to the file ***fname***.
5. "***int free_block(const char *fname, int bno)***" that reads the 1st 4096 Bytes (the Super Block, containing **n, s, ubn, fbn, ub**) from the file ***fname*** and frees the block ***bno***, that is, resets the bit corresponding to the block ***bno*** in **ub**. This function fills up the block ***bno*** with 0's. **ubn** and **fbn** too are modified appropriately. On success this function returns 1. It returns 0 otherwise. Please note that all the modifications done by this function have to be written back to the file ***fname***.
6. ***int readblock(const char *fname, char *myfile_name, int bno, char *buffer)*** - reads the ***bno***th block of ***myfile_name*** in ***buffer*** from the file ***fname***. This function returns 1 if successful, 0 otherwise.
7. ***int writeblock(const char *fname, char *myfile_name, int bno, char *buffer)*** - writes one block of data, taking it from buffer and putting it as the data for the ***bno***th block of ***myfile_name*** in the file ***fname***. If, at present, those many blocks are not present in the chain, this function will add required number of blocks in **myfile_name** filling them up with 0. If the given ***myfile_name*** does not exist, this function creates **myfile_name**. . This function returns 1 if successful, 0 otherwise.
8. Write a function "***int check_fs(const char *fname)***" that checks the integrity of the file ***fname*** with respect to n, s, ubn, hbn, ub and the contents of the blocks. In case of any inconsistency (say, **ubn**+**fbn** ≠ **n, ub** does not contain **ubn** number of 1's, etc.), this function returns 1. It returns 0 otherwise.

The above functions are finally used by ***myfs*** while inplementing the myfs calls shown earlier in the 1st figure of this assignment.Please note that you have to write all the functions of that figure,

Submission status

Attempt number	This is attempt 1.
Submission status	No attempt
Grading status	Not graded
Time remaining	6 days 7 hours
Last modified	-
Submission comments	▸ Comments (0)

Add submission

You have not made a submission yet.

◄ Filesystem Fundamentals

Jump to...