**CPEG308/CPSC503 F2021 – Simple File System Design**

A disk management system is made of 3 layers:

1) The first component is a simple shell application that allows the user to perform operations on the **Simple File System** such as printing debugging information about the file system, formatting a new file system, mounting a file system, creating files, and copying data in or out of the file system. It is a user interface (i.e., API provided by the file system and the programs that use them)
2) The second component takes the operations specified by the user through the command line and performs them on the **Simple File System** disk image. This component is charged with organizing the on-disk data structures and performing all the bookkeeping necessary to allow for persistent storage of data (i.e., internal structure, free block management and communication with device drivers)
3) The third component emulates a disk by dividing a normal file (called a **disk image**) into `4096-byte` blocks and only allows the **File System** to read and write in terms of blocks. This emulator will persistently store the data to the disk image using the normal open, read, and write system calls. See Fig 1.

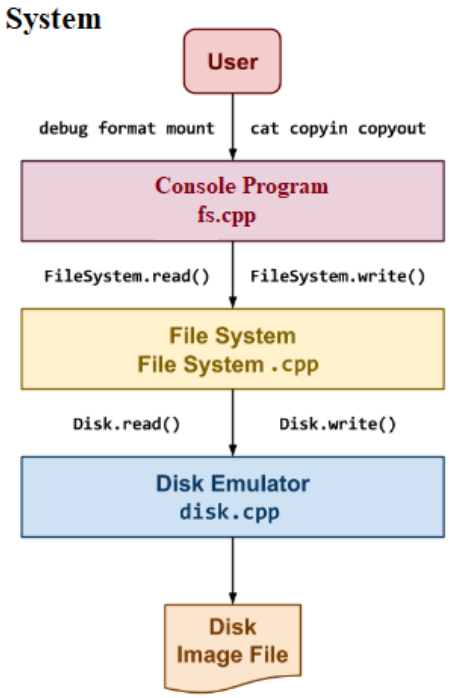

Fig 1. Disk Management System

The file system has a logical view of the disk that may be different than the one use by the disk controller. Assume a simple **file system** with disk of the size S, divided int blocks of size of `4KB (4096 bytes)`. So, the number of blocks on the virtual disk is #Blocks = S/`4KB`.

The first block of the disk is the **superblock** that describes the layout of the rest of the filesystem. Is has ten fields (Fig 2.):

1) **Magic**: Not used.
2) **Blocks**: The second field is the total number of blocks, which should be the same as the number of blocks on the disk.
3) **InodeBlocks**: The third field is the number of blocks set aside for storing **inodes**. The format routine is responsible for choosing this value, which should always be `10%` of the **Blocks**, rounding up.
4) **Inodes**: The fourth field is the total number of **inodes** in those **inode blocks**.
5) **Meta1 Bitmap Block#**: Is the block number for the bitmap the keep free status for Meta1 blocks or 1
6) **Meta2 Bitmap Block#**: Is the block number for the bitmap the keep free status for Meta2 blocks or 2
7) **Data Bitmap Block#**: Is the block number for the bitmap the keep free status for Data blocks or 3
8) **Meta1InodeBlockStart:** Is the location of the first Meta1 block or block# 4
9) **Meta2InodeBlockStart:** Is the location of the first Meta2 block (based on disk size)
10) **DataBlockStart:** Is the location of the first data block (based on disk size)
11) The rest of the space in the superblock block, is not used

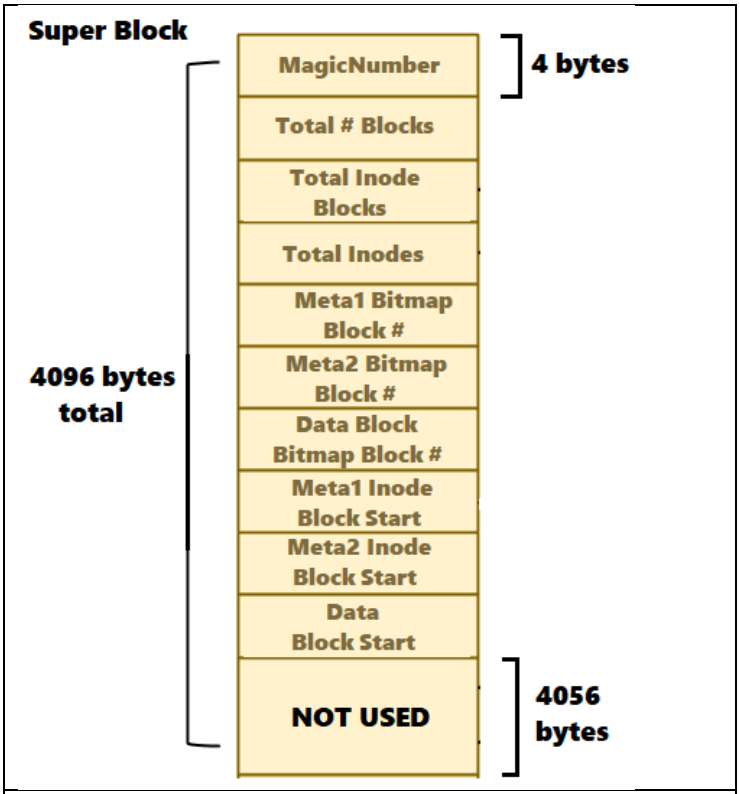

Fig. 2. Superblock data structure

A. Three blocks are reserved for free space managements in bitmap format. Each block can encode the free/not free status of 32768 items (or 4096 * 8). Items are inodes for Meta 1 & 2 and blocks for data blocks.
    1. The first bitmap is keeping track of free file control block FCB nodes or **free inodes** of type META Data1 (the FCBs are inode use to save filename and pointer to another inode).
    2. The second bitmap is keeping track of **free inodes** data of type META Data2. META Data2 inodes keep track of id of data blocks link to a particular file.
    3. The third bitmap is keeping track of **free data blocks.**
B. Following the superblock and the 3 bitmaps blocks are inode blocks. As stated above there are 2 types of inode blocks, one that contains inodes of Meta1 data type and another that contains inodes of Meta2 data type (Fig 3.). Typically, **ten percent** of the total number of disk blocks is used as **inode** blocks (The first 5% for Meta1 and the second 5% for Meta2 type). Meta1 and Meta2 are each 32 bytes, therefore each iblock can hold 128 of them.

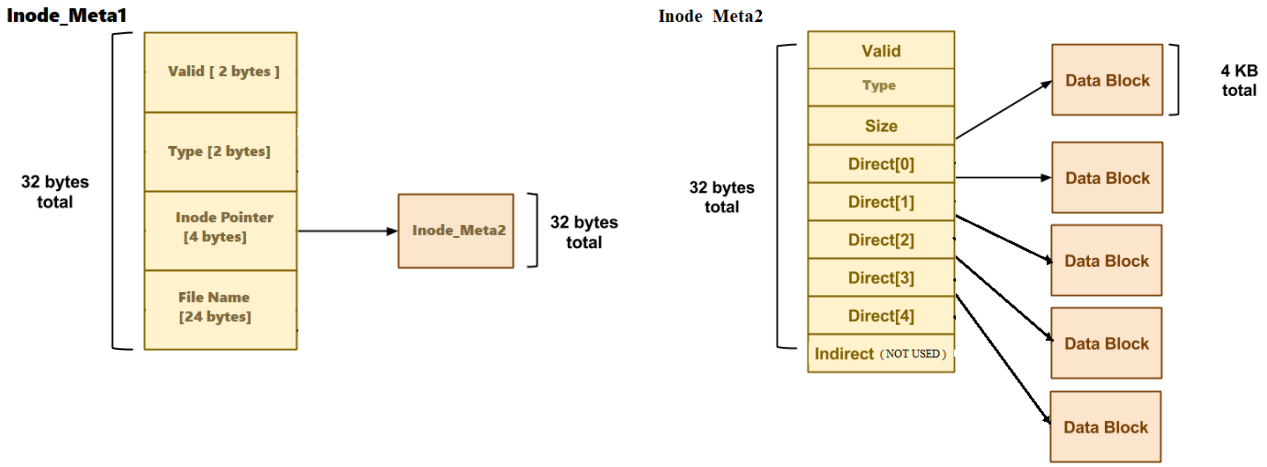


Fig. 3. Meta1 inode on the left and Meta2 inode on the right.

C. The remaining blocks in the filesystem are used as plain **data** blocks of 4KB each. (See Fig 4.)

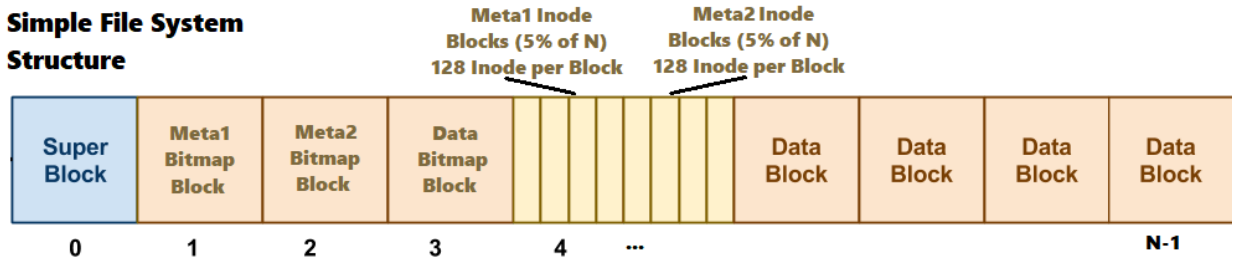


Fig. 4 Blocks Layout in SFS

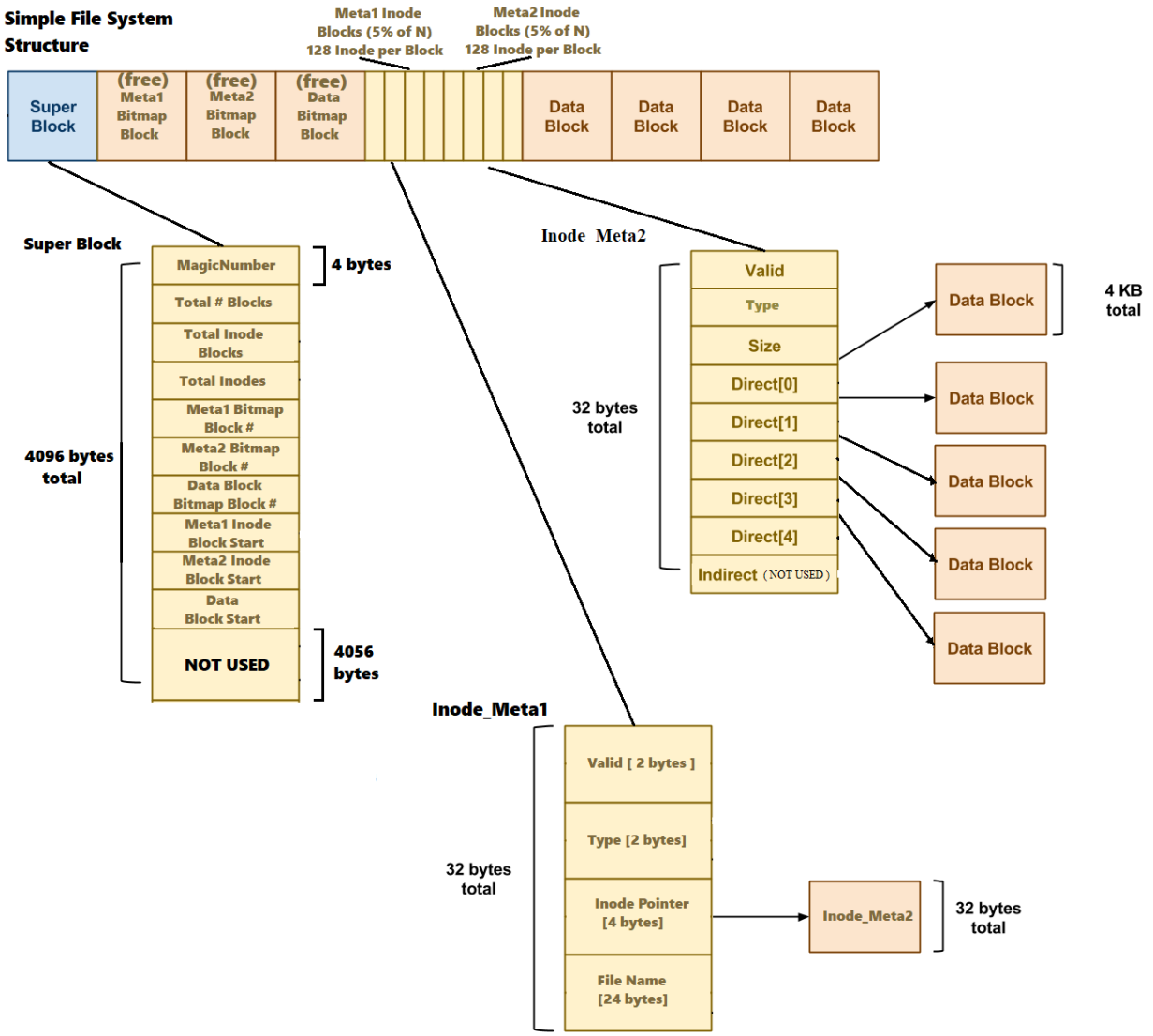


Fig. 5. Simple File System Interaction

Assignment:
Your assignment is to modify the starter code provided to implement the following two functions in FileSystem.cpp: read and remove

The implementation of these two functions will allow you to execute the copyout and the remove commands. You should only need to modify FileSystem.cpp and only read and remove functions.

This is a group project (2 -3 students)
For submission, you only need to upload FileSystem.cpp