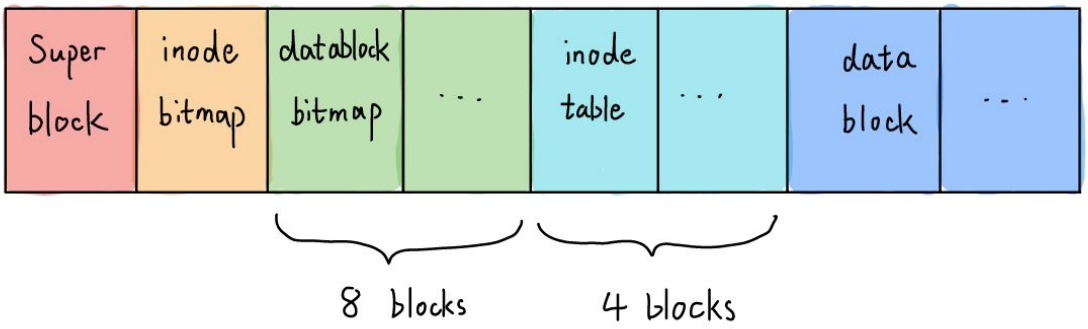
File System Proposal



A description of how you are partitioning space. This will include how you are keeping track of free inodes and data blocks, and where those data structures are stored, how you determine which blocks are used for inodes. Note that the size of the disk image and the number of inodes are parameters to mkfs.

The first block of our file system will be the super block. The second block will be the inode bitmap block. From the 3rd to 10th block are the data bitmap block. The 11th to 14th blocks are the inode table. The rest of the disk consists of data blocks. Each block is 4KB which is defined in the header file. We choose the structure bitmap to keep track whether the inode or datablock is in use. Each bit is used to indicate whether the corresponding block is free (0) or in-use (1). Each inode has a size of 256B and there are 64 inodes for our file system, therefore we only need to keep track of the first 64 bits in the inode bitmap block. All bits in the data bitmap block are used to keep track of the data block. 4096 x 8 x 8 = 262144, therefore we could keep track of up to 262144 data blocks in total. 262144 x 4KB = 1G, 1 gigabyte is the maximum of data our file system could store. 1 + 1 + 8 + 4 + 262144 = 262158, 262158 is the number of blocks in our disk. Moreover each block is 4KB, thus the size of our disk image is 4194304 KB.

A description of how you will store the information about the extents that belong to a file. For example if the data blocks for file A are blocks 2,3,4 and 8,9,10, then the extents are described as (2,3) and (8,3). Where is this information about the extents stored?

Each inode contains an array of 13 direct pointers and 1 single indirect pointer and 1 double indirect pointer. The direct pointer points to a struct that has 2 attributes which are the starting block of the extent and the length of this extent. For instance, starting block:10, length:3, then the data block index will be 10,11,12. The single indirect pointer points to an extent of pointers and the double indirect pointer points to an extent of single indirect pointer.

Describe how to allocate disk blocks to a file when it is extended. In other words, how do you identify available extents and allocate it to a file? What are the steps involved?

First we will have to access the number of free data bitmap and inode bitmap from the super block. If there is not enough space, out of size error will be indicated. Next, find a free inode for the file and change the corresponding inode bitmap to 1 and the attributes in the superblock. Then iterate through the data bitmap and find the number of consecutive 0s. Whenever the number matches the number of disk blocks that need to be allocated, we set the pointer of the inode to that part of the data block. Next, the data bitmap block and the superblock attributes will then be updated. If there is not enough consecutive free space for the extent, we could break the data of the file into smaller pieces and then allocate them. The way to break the file could be keeping track of the longest consecutive free data block and then allocate the data into those blocks. By doing this repeatedly, eventually the entire file will be allocated.

Describe how to free disk blocks when a file is truncated (reduced in size) or deleted.

To delete a file, we first look at the inode of the file. The inode contains the size of the file and the pointer to the start of the file data. We will derive how many data blocks are allocated by taking the integer division of the file size by 4096B ($size \div 4096$) - this will be rounded up to obtain the exact number of data blocks to be freed. Let's say the exact number is n, then we could just iterate through n data blocks in the inode pointer. The way to iterate is by going through the inode pointer in order. For instance, the first pointer starts from block 2 and has length 2, the second pointer starts block 7. The specific size is 10000B, then we know the third data block will have to be truncated, which is block 7 in the example. The remainder of $(size \div 4096)$ is the offset to get to the last byte of the file and then put EOF after the last byte. Next, iterate through the rest of the inode pointer and set the corresponding data block bitmap to 0 to free the data block.

Describe the algorithm to seek to a specific byte in a file.

First we need to locate the exact data block for that byte. That specific byte divided by 4KB gives a quotient and a remainder. The quotient is the number of data blocks we have to iterate in order to find the specific data block. The way to iterate is by going through the inode pointer in order. For instance, the first pointer starts from block 2 and has length 2, the second pointer starts block 7. The specific byte is 10000B, then we know the third data block will have the specific bytes, which is block 7 in the example. Next, we can simply add the offset which is the remainder to get the specific byte. The data are contiguous in the data block. Thus, Simply adding will provide us with the specific byte-position of the file.

Describe how to allocate and free inodes.

The inode bitmap will indicate which inodes are free or allocated within the inode table. The file system will be able to write on inodes where its corresponding bit in the inode bitmap is 0 (this indicates the section is free). So to allocate free inodes, the file system first checks which bits in the inode bitmap are 0. Then, it will write on that section of the inode table indicated by the bit and turn the bit into 1. The correspondence between the bit in the inode bitmap and the section in the inode table is made by their respective position in order. For example, if i th bit in the inode bitmap is 0, the file system will write on i th section of the inode table (address indicating start of the inode table + (256 * i) to get the address of the section). Before freeing the inodes itself, the system has to free the data blocks pointed by the inodes. This is accomplished through the same procedure stated above - setting the bits in the data block bitmap corresponding to the data blocks to 0. Once this initial step is done, the inodes can be freed by setting the bits designating the inodes in the inode bitmap to 0.

Describe how to allocate and free directory entries within the data block(s) that represent the directory.

A directory is a collection of entries and each entry costs 256Bytes of space. Therefore, the file system will iterate through the data block of the directory by 256 Bytes and check the entry's inode number and file name at each iteration. It will then allocate the first entry that has an inode number of 0. Here, inode "0" is reserved to indicate that the entry is free to use. To free an entry, the file system will simply set the inode number of that entry in the directory to be 0.

Describe how to lookup a file given a full path to it, starting from the root directory.

The inode and data block corresponding to the root will be reserved. From the inode of the root, the system will find the data block of the root directory. It will then browse through the data block to spot the entry that matches the "name" of the next file or directory in the path. The entry will present the inode number which will guide the system to the inode of the file or directory as indicated. If the system is guided to the inode of a directory it will repeat the same process as the above but looking for the next "name" in the path. Whenever a file's name is not found, an error will be indicated. If the system is led to a file, this indicates the end of the path has been reached. It then opens the content of the data blocks and its extent.