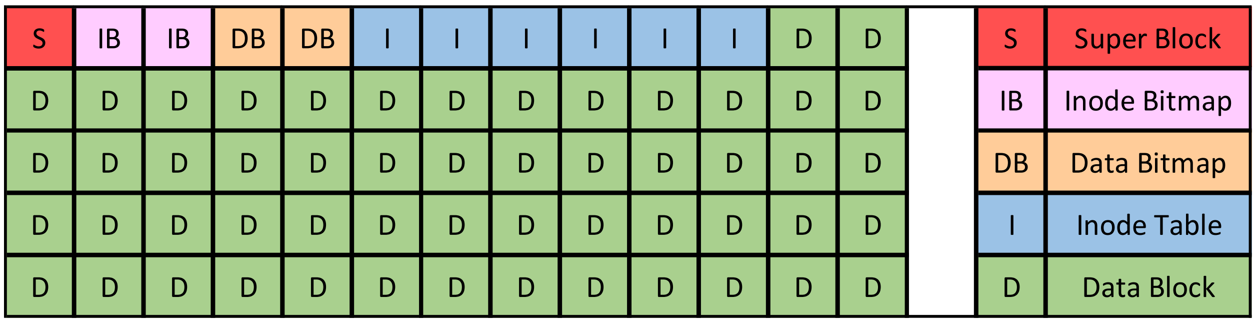
Yuchen Tong
1003534669

Junming Zhang
1003988982

# Assignment 1 Proposal

## 1. Disk Image Layout:



## 2. How to partition image (given the size of disk image: d KiB, the number of inodes: n, d and n are input by the user):

a. Each blocks = 4KiB = 4096 bytes = 4096 * 8 bits.

b. We allocate first block as Super Block, which stores the necessary information (metadata) of the file system such as the size of the file system, the address of the starting block of inode bitmap, data block bitmap, inode table, and data blocks, the index after the last used data block, number of inodes/data blocks, number of free inodes/data blocks, etc. The total size of the super block is 4KiB.

c. The next $n_{ib}$ blocks (where $n_{ib} = \lceil \frac{n}{4096*8} \rceil$) will be allocated as Inode Bitmap (one bit represents the status of usage of one inode), which keeps track of whether an inode is in use or not. (An inode is in use if the corresponding index of bit in bitmap is set to be 1, otherwise 0). The total size of inode bitmap part is $n_{ib}$ * 4KiB.

d. The next $n_{db}$ blocks will be allocated as Data Block Bitmap (one bit represents the status of usage of one data block), which keeps track of whether a data block is in use or not. (A data block is in use if the corresponding index of bit in bitmap is set to be 1, otherwise 0). The total size of data block bitmap is $n_{db}$ * 4KiB. The value of $n_{db}$ will be discussed in part f).

e. The next $n_i$ (where $n_i = \lceil n * \frac{inode\_size}{4096} \rceil$) blocks will be allocated as Inode Table, which stores all the inodes.

f. The rest of $n_d$ blocks are Data Blocks. Since one data block of bitmap can track 4096 * 8 data blocks, then we define $n_{db}: n_d$ should be $1: 4096 * 8$. Thus let $N_{left} = {}^{d}/_{4KiB} - 1 - n_{ib} - n_i$ blocks left for data block bitmap and data blocks. Then $n_{db} = \lceil \frac{1}{1+4 \quad * 8} N_{left} \rceil$, $n_d = N_{left} - n_{db}$. The total size of data block is $n_d$ * 4KiB.

## 3. How to store the information about the extents that belongs to a file:

a. The size of an extent is 8 Byte, since it stores two integer values (the starting address of contiguous blocks and the number of blocks it possesses).

b. Each inode will store a pointer to a data block (we call it extent table below) which is used to store all the extents it possesses, the maximum of extents it can possess is 4096 / 8 = 512 which satisfies the restriction. The extent table of an inode i is assigned to the first empty data block. This extent table can be visited easily by the pointer from the inode it serves for.

4. **How to allocate disk blocks to a file when it is extended:**
Inode Keeps track of how many extents it possesses. If a file needs to be extended, calculate enough data blocks (i.e. k blocks) that can be used to store all the extending size.
    a. If the following k blocks of the last extent are free, set the bits of these k data blocks to 1 in the data block bitmap, and modify this extent from (a, b) to (a, b + k).
    b. Search in the data block bitmap where the k blocks can be set consecutively as an extent. With the attribute s_next_free_data_block_addr in the superblock, we can find the address of the first block that all blocks after it remain free (for the "yes" case).
        i. If yes: Set those bits in data block bitmap to 1, create a new extent for the inode.
        ii. If no: search for m longest contiguous data blocks piece and set them as new extents (where $\sum_{i=1}^{m} length(piece_i) = k$) to minimize number of extents used for a file. Flip the corresponding bits in the data block bitmap (0 to 1).

5. **How to free disk blocks to a file When a file is truncated/deleted:**
    a. Truncating (Free part of the data blocks):
        i. Locate the inode of the modified file and look for the extents that need to be truncated.
        ii. If there are k blocks need to be truncated from an extent with length l, set the length of the extent to l – k and write the untruncated block consecutively from the same starting address. For example, blocks 4, 6 need to be truncated from extent (3, 5), that is, data block 3, 4, 5, 6, 7, the extent will be modified to (3, 3), the data block 4, 6 will be released and the data in 5, 7 will be moved to 4, 5 to maintain a contiguous extent.
        iii. Flip the bits that correspond to the truncated data block in the data block bitmap. (Set those bits from 1 to 0)
    b. Deleting (Free all data blocks):
        i. Locate the inode of the file that needs to be deleted.
        ii. For every extent in this inode, get all the block address that is in use. Flip the bits that correspond to those data block in the data block bitmap. (Set those bits from 1 to 0). Then the file becomes empty. If the file is also deleted by the user, Flip the corresponding inode bitmap from 1 to 0 (details discussed below in part 7).

6. **How to seek to a specific byte in a file:**
Want to locate the k-th byte in a file (k starts at 0):
    a. If k > the size of the file, then the byte is beyond the size of the file, an error will be raised.
    b. Calculate the t-th (t =floor[k/4096]) block that the byte is in (t starts at 0).
    c. Locate the inode of this file.
    d. Loop through each extent in the address that the inode points to, keep track of which block currently at by adding up the length of each extent, and the total length is sum_len. Once sum_len > t, we have found the extent that contains the t-th data block. So, the byte is located at (t – (sum_len – current_extent_length))-th block in the extent, and (k % 4096)-th byte of this block.

7. **How to allocate/Free Inodes:**
    a. Allocate Inodes:
        i. Loop through inode bitmap, find the nearest un-reserved index.
        ii. Flip the bit at this index of the bitmap. (from 0 to 1).

   iii. Create an inode at the same index of inode table, initialize it with information of the inode including mode, size e.t.c.

b. Free Inodes:
   i. Locate the inode of the file that needs to be freed.
   ii. Free all the data block that this inode possesses: For every extent in this inode, get all the block address that is in use. Flip the bits that correspond to those data block in the data block bitmap. (Set those bits from 1 to 0).
   iii. Flip the bit of the inode bitmap at this index of this inode (From 1 to 0) to indicate that this index is free.

**8. How to allocate/Free directory entries:**

a. Allocate directory entry:
   i. Iterate through all the directory entries of the directory block to find the first directory entry with inode_number = -1.
   ii. Initialize the directory entry with inode number and file name.

b. Free a directory entry:
   i. Locate the directory entry that need to be freed follow the path.
   ii. Set the inode number to -1, which indicates that this is a free directory entry.

**9. How to lookup a file given a full path (i.e. path: /a/b/c.txt):**

a. Use Super Block to locate the root inode (root directory) (Take the root inode from the inode table pointed by the super block).

b. Search the directory entry with name “a” in root directory, locate the inode that this directory points to with the inode number in the directory entry, according to the file_type in the inode of “a”, “a” is a directory. (directory a).

c. Search the directory entry with name “b” in directory a, locate the inode that this directory points to with the inode number in the directory entry, according to the file_type in the inode of “b”, “b” is a directory. (directory b).

d. Search the directory entry with name “c.txt” in directory b, locate the inode that this directory points to with the inode number in the directory entry, according to the file_type in the in the inode of “c.txt”, “c.txt” is a file.

e. Thus, we successfully found the file and got its inode.

   Note: this search succeeds if each file (regular file or directory) can be found based on the given path at the given directory, or the search fails and an error will be raised.