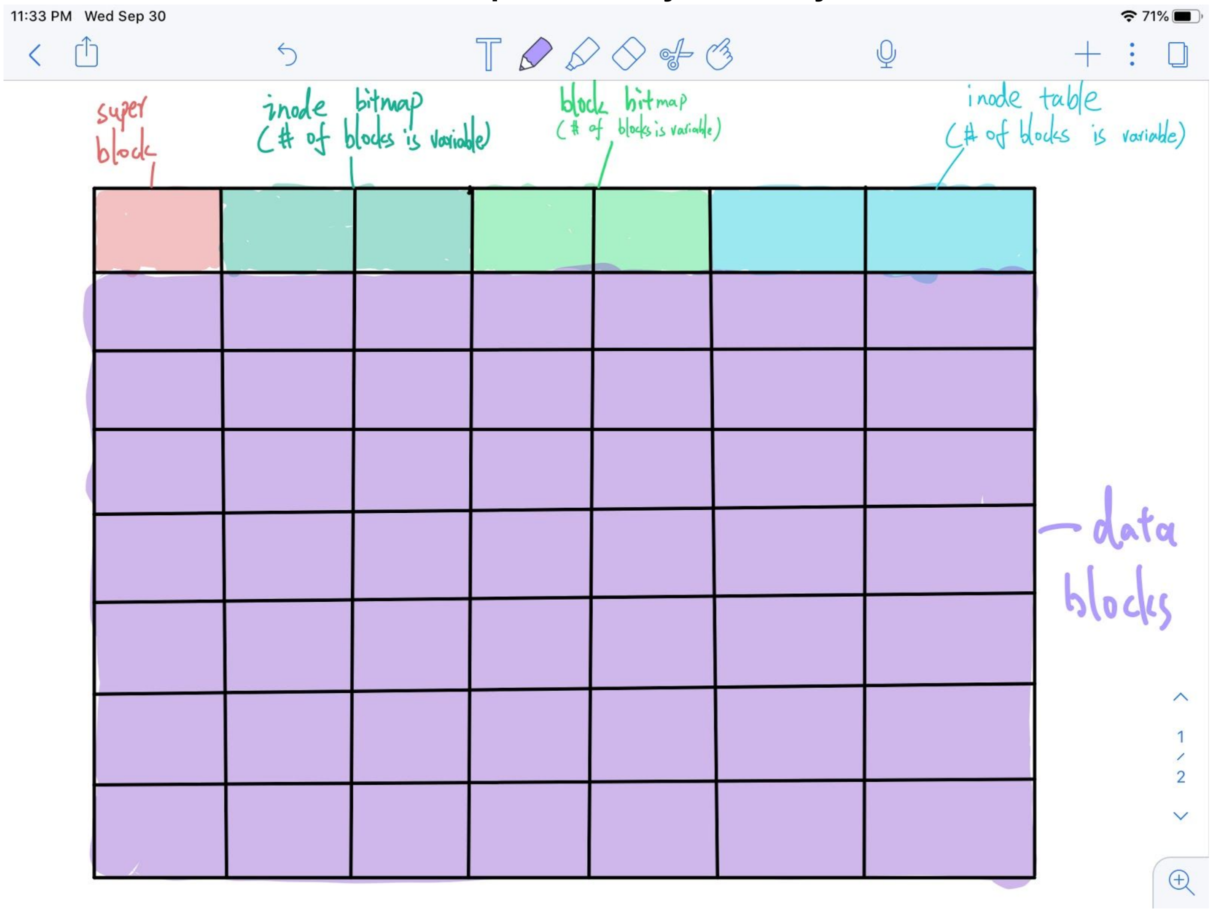
CSC369 Assignment 1 Proposal

Ruijie Sun, Wenjie Hao

Q1. A simple diagram of the layout of your disk image. The diagram will indicate where the different components of your file system will be stored.


11:33 PM Wed Sep 30
71%
super block
inode bitmap (# of blocks is variable)
block bitmap (# of blocks is variable)
inode table (# of blocks is variable)
data blocks
1
2

Q2. 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 file system splits the disk into blocks of 4096 bytes. The total # of blocks is determined by size of disk image following the formula total # of blocks = ceil(size of disk image / 4096)
The first segment is allocated for the super block. And the 1st block is allocated for the super block.
The second segment is allocated for inode bitmap. The inode bitmap starts from the 2nd block and the length of blocks is determined by the # of inodes in mkfs, which is an input parameter. Note that # of blocks = ceil(# of inodes / 32768)

The third segment is allocated for block bitmap. The block bitmap starts strictly after the end of the inode bitmap segment. The length of the block bitmap segment is determined by the size of disk image which are input parameters of mkfs.
Note that # of blocks of the block bitmap = ceil(size of disk image / (4096*4096*8))

The next segment is inode table which starts strictly after block bitmap segment and its length(# of blocks) is determined by parameter # of inodes in mkfs. Note that # of block for inode table = ceiling(# of inode * sizeof(inode struct) / 4096)).

All remaining blocks are data blocks.

The inode bitmap and block bitmap enable us to **track free inode** and **free data blocks**.

Q3. 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?

All extents for that file will be stored to an extent table.（Each extent table is stored in the data block segment.）
Each extent struct which contains the starting address and length. The file inode contains the **pointer which points to that extent table**. The inode also stores the **count of extents** belonging to the file.

Q4. 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?

Main idea is that the file system manages to update the last existing extent to allocate as much new data as possible. Then, according to the block bitmap, the file system will keep creating a new extent as long as possible to allocate the remaining data until all data of the existing extent is allocated.

Q5. Explain how your allocation algorithm will help keep fragmentation low.
The reason is that the file system is always trying to make extents as long as possible. In other words, when the file system looks for blocks in the bitmap to satisfy a write request, it looks for a contiguous region of bits large enough to fit the number of blocks that are requested.

Q6. Describe how to free disk blocks when a file is truncated (reduced in size) or deleted.

If a file is truncated:
- Set the corresponding data bitmap to be 0 for the data deleted.
- Update the related extents in the extent table.

- Update the size information, last updated time and extents count of the corresponding inode in the inode table.

If a file is deleted:
- Set the corresponding data bitmap to be 0 for the data deleted.
- Set the corresponding data bitmap to be 0 for the extent table deleted.
- Set the corresponding inode bitmap to 0.

Q7. Describe the algorithm to seek a specific byte in a file.

Firstly, the file system finds the extent table based on file inode. Then, based on extent's starting address and length, the file system iterates all extents to accumulate the data byte to match the specific byte number.

Q8. Describe how to allocate and free inodes.
Allocate:
● Find the smallest free inode index in inode bitmap, say index 1, set it from 0 to 1.
● Based on the inode index 1, calculate the starting address in the inode table. Initialize an inode struct in this starting address.
● Find the smallest free data block index in block bitmap, say index 2, set it from 0 to1. Calculate this data block address and set it in inode.extent_table.
● Initialize extent(s) in extent table, update corresponding data block bitmap, and update extent countin the inode.

Free:
● In the inodes_bitmap, change the value on the corresponding index as 0
● based on inode.extent_table, extent_count, and all information of each extent, find all related data block(block for extent table + blocks for real data) address, then find indices in block bitmap, set them as 0.
● if the inode type is directory, free all entry inode as well.

Q9. Describe how to allocate and free directory entries within the data block(s) that represent the directory.

Allocate:
In data block(s) that represent the directory, write allocated directory entries structs.

For each directory entry, based on the Inode bitmap, find a free inode number and write the Inode number and name of the directory entry in this directory entry struct.

Update entry count in the parent Inode struct.
Update extents count in the parent Inode struct if in need.

Free:
For each directory entry, based on Inode number, reset the corresponding bit in Inode bitmap as 0 and find the Inode struct. Then, based on the extent table and all related extents, reset all corresponding bits in block bitmap as 0. Lastly, free each directory entry in this directory.

Update entry count in the parent Inode struct.
Update extents count in the parent Inode struct if in need.

Q10. Describe how to lookup a file given a full path to it, starting from the root directory.

- Access super block get all metadata needed
- Access the root Inode struct
- From the inode struct, access the corresponding extent table and all real data blocks, then look into the directory entries to match the name and type of next part of the path.(The directory entry's type can be found by going back checking the corresponding inode struct in the inode table.)
- Repeat the above step and If we reach the end of the path, it means that we find the target file. Otherwise, this is an invalid path.