CSC369 A1 Proposal

Zhongliu Liu
Yifan Liu

1. 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.

Super block	Inode Bitmap	Block Bitmap	Inodes	…
Data Blocks	…			

2. 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.

We have four general parts in the file system which are super block, Inode Bitmap, Block Bitmap, Inodes, and Data Blocks. The super block, Inode Bitmap, Block Bitmap will be one block each, the number of blocks for inodes and data will be determined when creating the file system. The size of the inode is fixed so giving the number of inodes required by the user we can reserve enough blocks for inodes and the bites in Inode Bitmap is also determined by that time. The rest of the blocks are for data and the bites in Block Bitmap will also be determined.

3. 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?

We have an array attribute called i_extents in our inode struct which stores the tuples for one extent of the file. The size of array i_extents is 27, if a file or a directory takes more than 27 extents we will use a pointer that pointers to a data block and store the extra 485 extents in that data blocks.

4. 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?

We first find the inode of the file and try to find free blocks in data block sections by checking blocks bitmap, find the available extent that can fit the extended data of the file and then allocate it. If such extent does not exist, that means we do not have one big extent for our data, in this case we will try to find the biggest extent available and allocate part of the extend data then looking for the next biggest extent, continually doing this until all the extend has been allocated. In extrema case that there is no available extent for new data or the file running out of all 512 extents we just report error and do nothing to the system. After we extend the file successfully, we need to update the attributes in inode of the files, the number of free blocks in super block and also the blocks bitmap and data bitmap.

5. Explain how your allocation algorithm will help keep fragmentation low.

The extent we are using can be any number of data blocks, is such file cannot be fitted into one extent, we will try to find many separate extents that can be used. In this way we fill up the fragmentation and keep using all the available data blocks.

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

When truncating a file, after we get the inode of the file we will delete all the extents related to the truncated data and reduce the size of some extent by changing the count attribute in struct tuple if necessary. In some case we also need to free the block for extra extent. Then we just set the corresponding bit in the bitmap to 0 to show that this block is free and ready to write new data in. If we want to delete a file, all the data blocks related to the file and the inode will be freed by setting the corresponding bit in the bitmap to 0. The new inode and blocks of data will just write over the already freed ones. We also need to check all the directories that contain this file and delete it from the directory. In the last we also need to update the variables in super block.

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

First, we will look through the array of extents in the inode, calculate the size of extents and find out which block of data stores the byte we need, since the file was stored sequentially we can find the exact byte we want. Then go to that block and we can easily locate the byte we want by doing some calculations.

8. Describe how to allocate and free inodes.

We allocate a new inode by looking up the inode bitmap to find the first 0 value in the bitmap, then we can allocate our new inode at that free location in inode table. When free an inode, we just change the corresponding bit in the inode bitmap, the new inode will just write over the already freed inode so we don’t need to delete the inode itself. After we complete allocate or free directory entries, we also need to update the variables in our super block.

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

We basically treat the directory entries as a normal file, but we store an array of structs of directory entry, each directory entry struct will contain the information of inode number of the files that are in the directory, the file name and the length of the struct itself. When we want to free the directory entries, we also need to delete the files that are only stored in this directory. We also need to free the inode of this directory and set the corresponding bit in the bitmap to 0. In the last, we have to update the variables in our super block.

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

First, we go to the root directory and find the file with the name we want and then get the inode number of that file, then we look up the inode and go to the data blocks. Repeating the steps above until we reach the file we want.