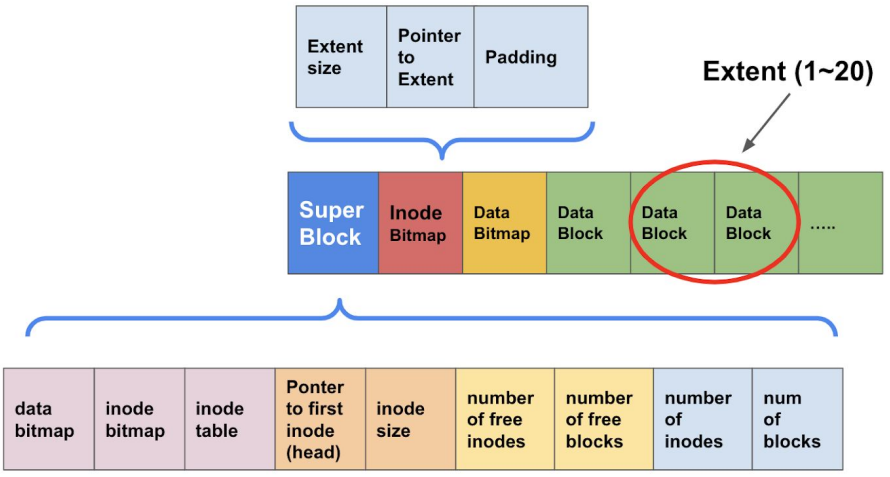
CSC369 A1a Proposal

Zhuolin Hou, Qi Zhu

Feb 1st, 2020

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.**

Disk Image Diagram



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, and how you determine which blocks are used for inodes.**

   We use the superblock to record the location of the data bitmap, the inode bitmap, and the inode table. It also stores the number of inodes in total, the number of data blocks in total, the size of inodes, free blocks count, and free inodes count. We keep track of free inodes and data blocks using bitmaps. Each inode records the extents used by the file, and each extent records its starting data block and its length.

3. **A description of how you will store the information about the extents that belong to a file.**

The extents are stored in the inode that represents the file and we store a pointer to the extents of the file. Each inode has an array list which can store up to 20 extents. And each extent stores its starting block and length.

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?**

1. When a file is extended, we first decide how many data blocks the extension needs.
2. We use the bitmap to see if there's enough available data blocks in the extents of the file that we have already allocated. If there is, we extend the extent, and modify the inode. If there isn't:
    a. We locate the available consecutive data blocks of that number.
    b. If there's no available consecutive data blocks of that number, we can split the number and use multiple extents.

        (We always search from the beginning of the data blocks)

    c. We store the starting block and length to the new extent(s), and add this extent to the file inode.
3. Then we modify the data block bitmap accordingly.

Return an error if space needed to store the extended data is larger than space available in the file system, or if the number of extents used by the file exceeds the limit 20.

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

By first checking if it's possible to extend an extent, we keep the file fragmentation low. Then we check if there's available consecutive data blocks of the size needed, trying to allocate a single extent. And we always search from the beginning of the data blocks when finding the available space, thus improving the fragmentation for future write requests.

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

As mentioned in q3,the extents are stored in the inode that represents the truncated file . For each extent used by the file, it records its starting data block and its length. In order to free disk blocks, we'll first need to free all data blocks inside the extent and we can further proceed to free the inode block. Correspondingly, the file

info in the located directory , the corresponding data block bitmaps and inode should also be updated.

7. **Describe the algorithm to seek a specific byte in a file.**
    1. Compute block number and offset in block for byte offset.

       blocknum = offset / blocksize, blockoff = offset % blocksize

    2. Check if blocknum is pointed by an extent stored in the file inode
    3. If so, read the extent to get the data block and use blockoff to seek to byte within the block.
    4. Else, the byte is not stored in the file system, raise error.
8. **Describe how to allocate and free inodes.**

    Note: when performing inode allocation and free i-node, we need to update the number of free i-nodes “num_free_inodes” in superblock.For i-node allocation, we’ll find the first zero entry of the bitmap by searching i-node bitmap. In the i-node table, the new inode will be allocated to the available free location.To free the i-node, adjust the bit of i-node bitmap entry to 0;

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

    We can store the directory entries as struct “a1fs_dentry”, with attributes including the inode number, directory entry length, file type, and file names under this directory. In this case, allocation and free operation are the same as the operations on file.

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

    Given the full path of the file, we’ll look up the file recursively. Repeat the following steps until we find it. We can search the file that matched the target name in the root directory firstly. Then we will look up the i-node of the file to pair up its corresponding data blocks.