Kwabena Ohemeng, Terence Williams, Omer Baldo

CS416: Assignment 3 Simple File System

During this following project we defined the following structs in the params.h file:

```
typedef struct inode{
    char type;
    uid_t user_id;          //User ID  Number                    4
    gid_t group_id;         //Group ID Number                    4
    long fileSize;          //File Size In Bytes                 8
    time_t lastAccess;      //Time of last access.               4
    time_t created;         //When was it created                4
    time_t modified;        //When was it last modified          4
    int block_amount;       //amount of blocks it owns           4
    int pointers[12];       //pointers to blocks the             32
    directoryRow * dir;     //directory pointer if file is directory
    mode_t mode;
    //file owns
    int inode_number;
    char buffer[4];
}inode;
```

```
typedef struct directoryRow{
    int inodeNumber;//-1 if it is free. otherwise inode# of directory
    char fileName[10];
}directoryRow;
```

```
typedef struct directory{
    directoryRow table[31];
    struct directory* indirect;
}directory;
```

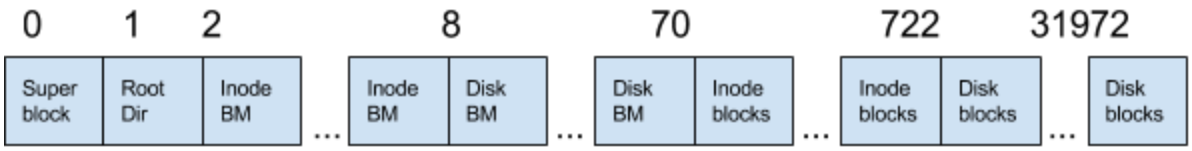
```
typedef struct super_block {
    int init;                           //has the file system been
initialized before ?
    unsigned long  s_magic;             /* filesystem's magic number*/
    unsigned long long  s_maxbytes;     /* max file size 6kb        */
```

```
    unsigned long  s_blocksize;         /* block size in bytes 512  */
    unsigned char s_blocksize_bits;     /* block size in bits 512*8 */
}super_block;
```

The following structures have the following sizes

Struct	**Size**	**How many we are fitting in a 512 block**
Inode	128	4
Directory	500	1
Superblock	40	1

Thus our disk file has the following look



This will give us the limitation of having a max of 31 files in our root directory. We designed our file system around the fact that the maximum file size we would have to implement would be 16MB or 16 * 10^12 byte. From this we deduced:

- 31,250 disk blocks(of length 512) are needed to store a 16mb file
- 31,250 bits to represent if those disk block are free or not. That's ≈ 62 blocks for disk bit map **good**
- 31,250/12 ≈ 2608 inodes to represent them(if we have 12 pointers to disk blocks in each inode and one indirect pointer to another array of 12 pointers) . That's ≈ 652 Blocks for inodes **good**
- 2608 bits needed to represent if those inodes are free or not. That's 6 blocks for inode bitmap. **good**

In our program we represent these values in global variables which we read and write back in when the system is mounted / unmounted so the file system can retain its state between calls. With this approach our program is not non volatile but we do not expect the grader to turn off their computer

while testing this file system. By using global variables it makes coding easier for us. Here is what we used

```
char inodeBM[2608]
char diskBM[31250]
inode inodeTable[2608]
```

Limitations

Originally we planned for directory support and memory indirection. Unfortunately, due to a lack of time, we could not implement these features in time. Here are restrictions of our system.

- A max of 6kb per file
- A max of 31 files

Running the code

When writing and reading to the system we did not use the block_read or block_write API provided. The reason for this is so we could write and read structs in with their direct size so we could maintain a constant file system between unmounting and mounting. Instead we read and wrote into the disk file directly. In block.c we added this method:

```
int getDiskFile(){
    return diskfile;
}
```

And in block.h

```
#define BLOCK_SIZE 512
#define maxDiskBlocks 12
#define amountOfINodes 2608
#define amountOfDiskBlocks 31250
int getDiskFile();
```

Do to this when running this project please use our version of block.c and block.h.