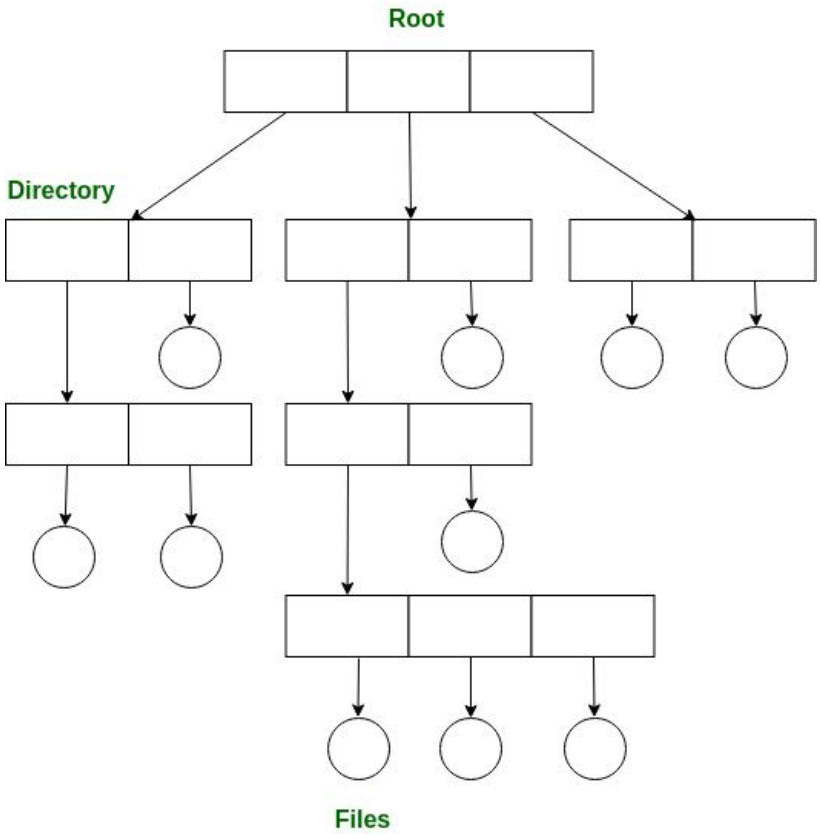
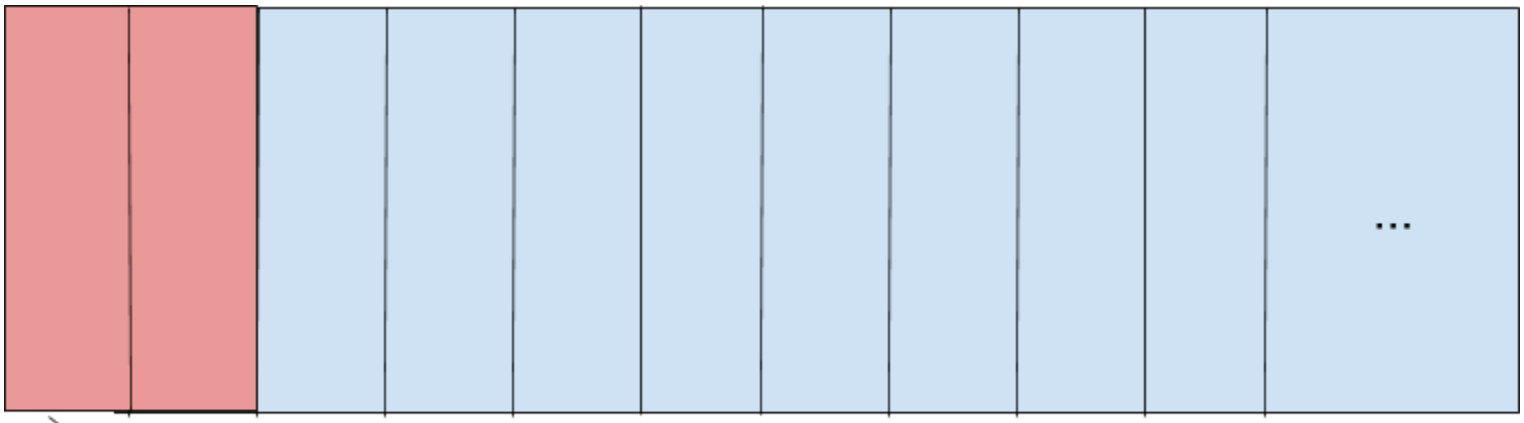
Dominic Taraska Tug76525
CIS 3207

Project 4A File System on a Virtual Disk, Design/Pseudocode

The goal of this project is to implement a simple file system on top of a virtual disk. To complete this, a library of file system function calls will be implemented, the data and meta-information will be stored on a virtual disk. The virtual disk is a single file that is stored on the file system provided by Linux OS. The virtual disk contains 16,384 blocks and each block will hold 4KB.

The file system will support a directory hierarchy where files are to be stored in directories. Files can be created and destroyed as well as directories.
The file system will store a maximum of 256 files, and half the space of the virtual disk will be reserved for data (files, directories) while the other half will be reserved for meta-information.



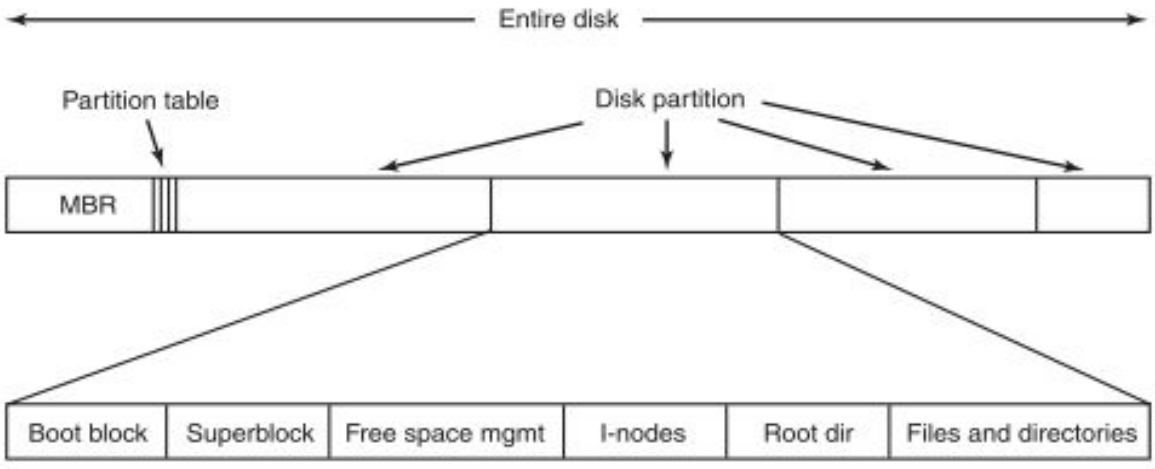


Partition of virtual disk file

-16384 blocks
-Block is 4096 bytes (4KB)

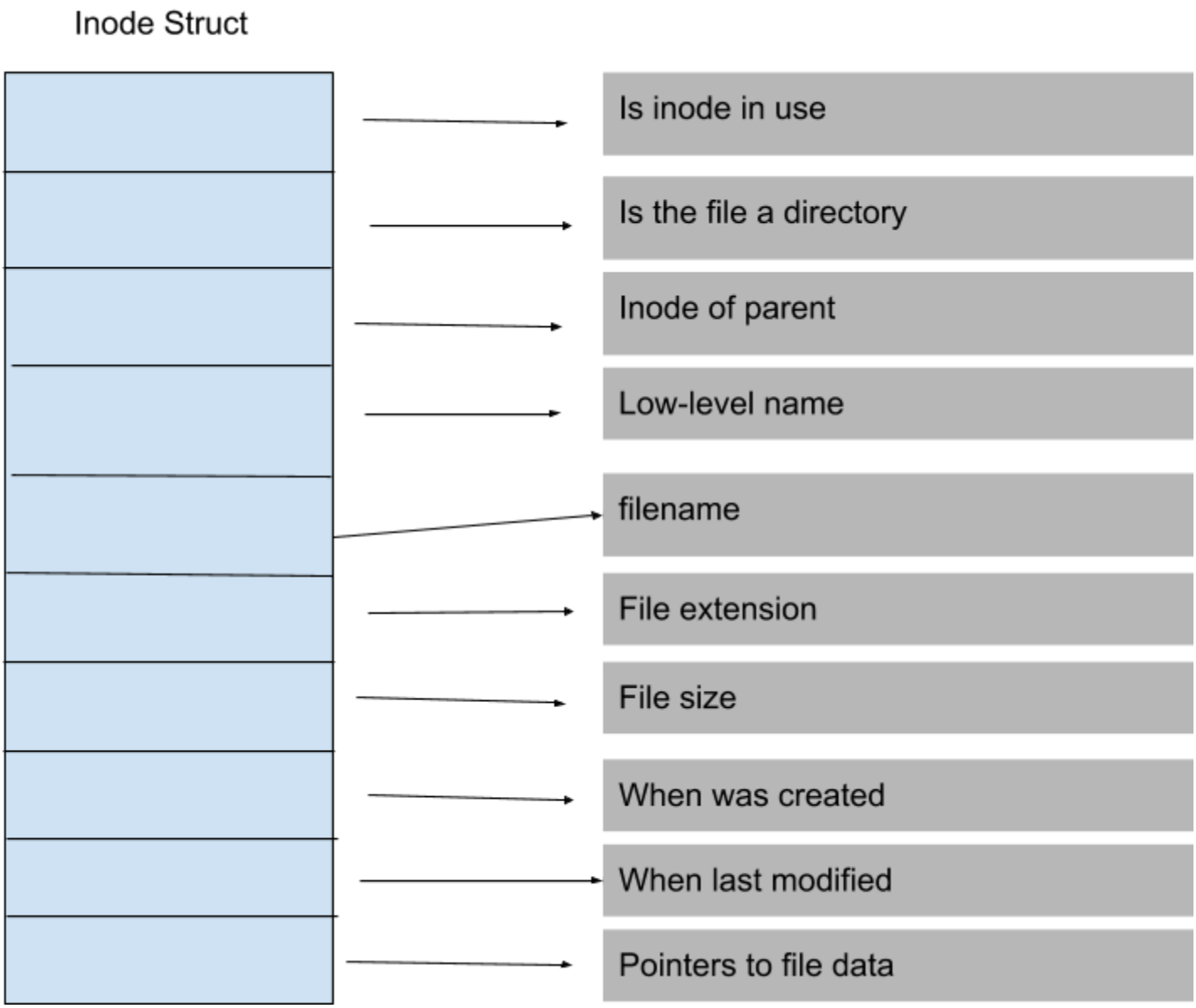
Entire virtual disk file is 64 Mb
32 Mb will be reserved for
data blocks

First blocks are bootblock, superblock



-Bootblock contains boot sector that holds the necessary data to start the system.

Superblock Struct


Inode Struct
Is inode in use
Is the file a directory
Inode of parent
Low-level name
filename
File extension
File size
When was created
When last modified
Pointers to file data

Free spaces are kept track by employing bitmap structs to indicate whether it is occupied

```
struct data_bmap {

        bool occupied[4096]

};
```

-Starting block of root directory is stored in superblock data structure
-Normal files are distinguished from directory files in the inode data structure with a bool variable.

Things like superblock struct and inode structs are mapped using calls to mmap:

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
(from <sys/mman.h>)

addr is starting address for new mapping
length specifies the length of the mapping
prot argument describes desired memory protection (PROT_READ, PROT_WRITE, …)
flags argument says whether updates to mapping are visible to other processes in same region
fd is file descriptor
offset is multiple of page size

Superblock data structure creation:

initialize_superblock (superblock *superB){
        sets all relevent member variables such as block size, number of blocks, max file size
}

Creation of data structures that keep track of occupied data blocks and inodes:

initialize_data_bmap(data_bmap *bm){

        for(i, i<4096)
                bm->occupied = 0;   (Sets each data block to unoccupied state)}

Creation of inode:

create_inode(char *name, int free_dataBlock, int free_inode, bool dir){

(name is name of file, free_dataBlock is index of free block, free_inode is index of free node, dir indicates whether file is a directory)

if (name is “root”), set parent directory to 0, or set parent directory to current directory

if(dir), set extension to “dir”, else set it to “txt”

set inodes data block pointer to free_dataBlock

}

I/O functions:

void makeDirectory(char *name){

find free inodes and data blocks
initialize inode with create_inode()
update directory
}

void createFile(char *name){

find free inodes and datablocks
call to create_inode()
update directory
}

int openFile(char *filename) {

for(i=0; i < # of inodes; i++)
check if inode name matches filename
if(inode name = filename), now check if the file is a valid directory
If not a directory, print error, return 1
If file not found, print error, Returns 0 if successful }

```
int closeFile(){

        *make sure it is not the root directory, if it is then return 1*
        *find parent directory, and set current directory to parent directory*
        *return 0 if successful, 1 if failure*

}

int readFile(char* filename){

        for(i=0; i < # of inodes; i++)
                *check if inode name matches filename*
                        if(inode name = filename)
                                *open file and print contents of inode, return 0 on success*    }

int writeFile(char *filename){

        for(i=0; i < # of inodes; i++)
                *check if inode name matches filename*
                        if(inode name = filename)
                                *open file for writing and get user input*
                                *write user input to file*
                                Return 0 on success, 1 on failure

}

int deleteFile(char *filename){

        for(i=0; i < # of inodes; i++)
                *check if inode name matches filename*
                        if(inode name = filename)
                        *change inode state in bitmap to unoccupied*
                        *free the data blocks in data bitmap*
                        Return 0 on success, 1 on failure

}

Main driver of program will essentially be a while loop:
while (notDone){
        printPrompt();
        *get user input*
        *perform appropriate action using above functions*
}
```

Testing:

Possible bugs include:

-trying to access block number outside range (>16383)

-trying to write data that exceeds block size (>4095)

-exceeding space provided by virtual disk

-not writing data to blocks correctly (data written does not match data provided by user)

-data being written or read to/from the wrong blocks

-improper parsing of user input

Testing would also include making sure that partition of virtual disk is performed correctly as well as creation of necessary data structures (superblock, bitmaps, data blocks, …) And also the collection of user input for writing data.

Test program will be written that demonstrates creation of virtual disk and file system. It will use all the functions required of the filesystem, and it will demonstrate copying a file from the OS file system to my filesystem, and my filesystem to the OS file system.