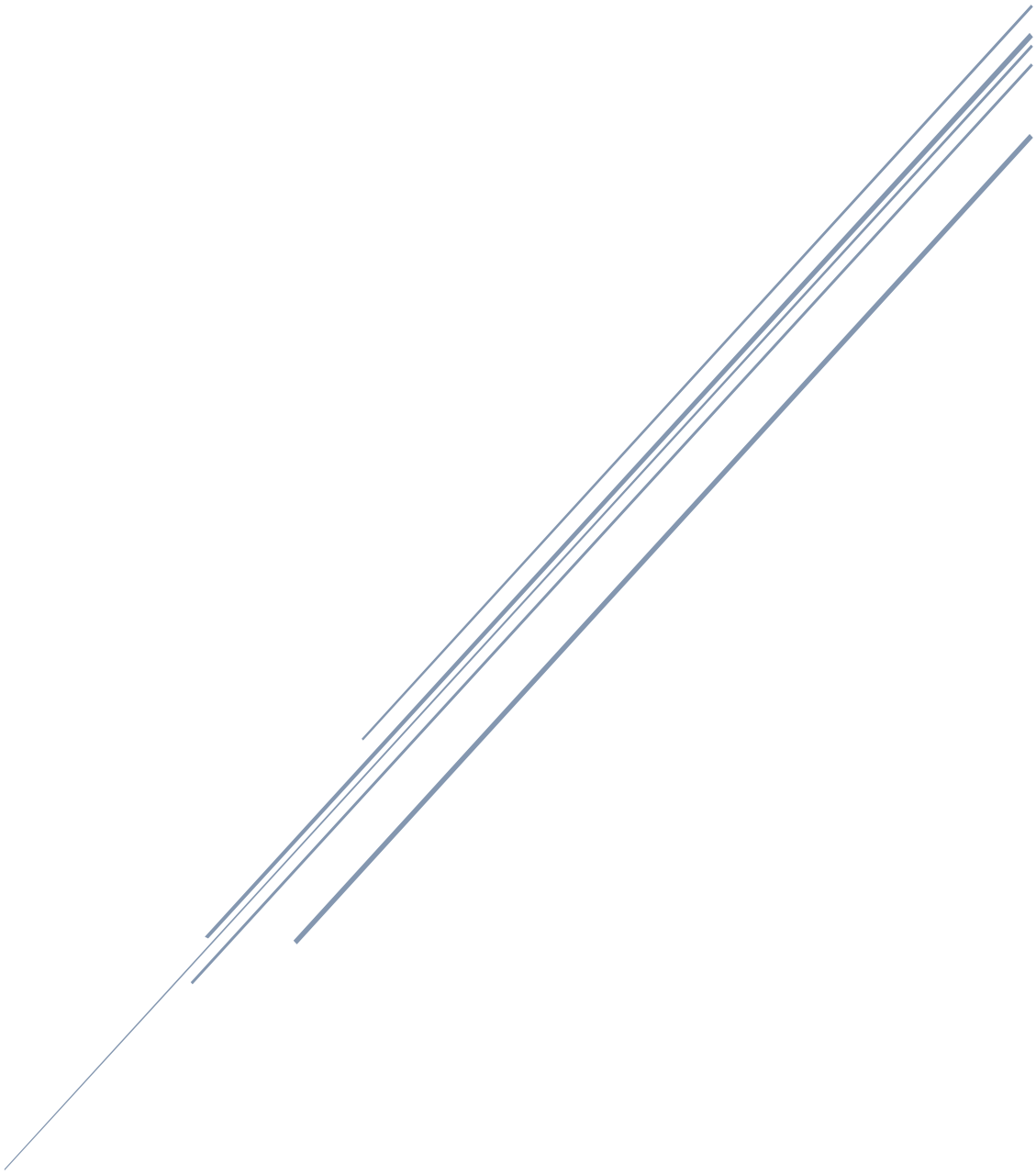
CSE312 MIDTERM DESIGN REPORT

Student : Fatih Selim Yakar

Instructor : Yusuf Sinan Akgül



Gebze Technical University
Computer Engineering

CSE 312 Operating System Midterm Design Report

Definition of directory structure and directory entries

In the file system, I reserved 5% of the entire file system size to keep directory entries and this block starts with “files and directories” title. So it takes approximately 51 kilobytes. And I reserved 1 block (it changes according to block size of file system) to keep root directory also this block starts with “rootdir” title. Each directory entry occupies approximately 110 bytes and root directory entry occupies 66 bytes . A directory contains at most three file and two directory in itself. And a directory entry includes some file specification. These features are as follows:

1. **Directory number/address :** Specifies the order of the directory.
2. **Directory name :** Specifies the name of the directory. The name size can be 20 characters/bytes at most
3. **File 1 :** It contains the inode number/address of the file contained in the directory. The number size can be 7 characters/bytes at most.
4. **File 2 :** It contains the inode number/address of the file contained in the directory. The number size can be 7 characters/bytes at most.
5. **File 3 :** It contains the inode number/address of the file contained in the directory. The number size can be 7 characters/bytes at most.
6. **Directory 1 :** It contains the another directory number/address of the file contained in the directory. The number size can be 7 characters/bytes at most.
7. **Directory 2 :** It contains the another directory number/address of the file contained in the directory. The number size can be 7 characters/bytes at most.

A directory entry looks like as follows in the filesystem disk (mySystem.data):

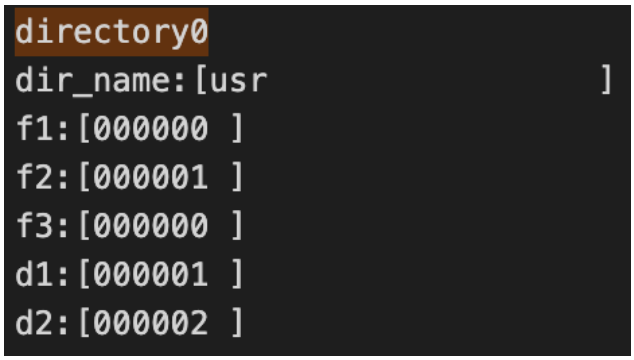


Figure 1- A directory entry indexed/numbered 0 and named usr.

One directory entry is created for each file created in file system. In other words, even if there is a file in the root directory, it must have a directory entry in the blocks reserved for directory entries. Since the root directory is a special directory, I have reserved a separate block for the files and directories in it. Only files created with "/ filename" or "/ directoryname" path are added here.. Root directory entries in this block contain the following features:

1. **Rootdir number/address :** Specifies the order of the root directory. Note that this number different than file entry's directory number. It exist next to the "d_name" or "f_name" titles.
2. **Directory name :** Specifies the name of the root directory. The name size can be 20 characters/bytes at most
3. **File name :** Specifies the name of the root file. The name size can be 20 characters/bytes at most

A root directory entry looks like as follows in the filesystem disk (mySystem.data):

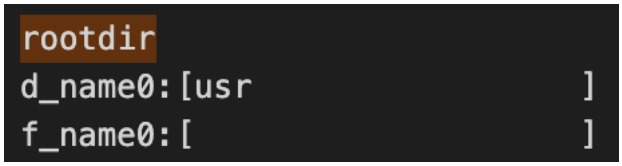


Figure 2- A root directory entry indexed/numbered 0 and named usr (the "usr" is same file in previous file system)

For nested file formation, the directory1 parameter in a parent directory entry or the directory2 parameter if it is full is written in the index of the internal directory. So it gets linked.

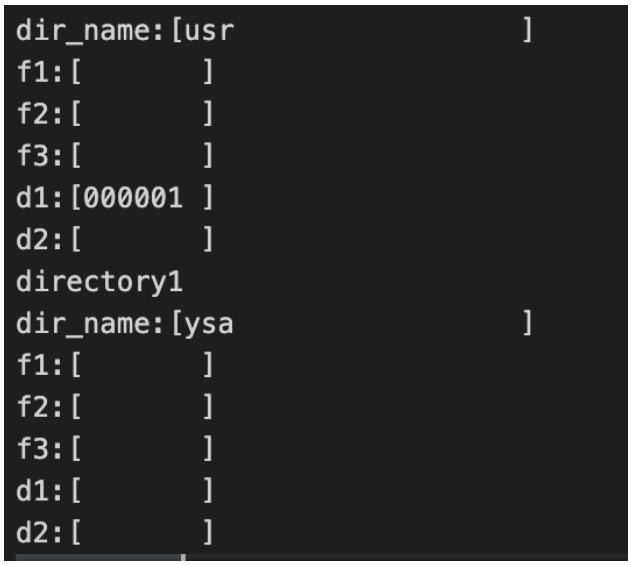


Figure 3- Nested directory example for "/usr/ysa"

In order to link a file to the directory, the inode index/number of the file is saved in the f1 or f2 parameters.

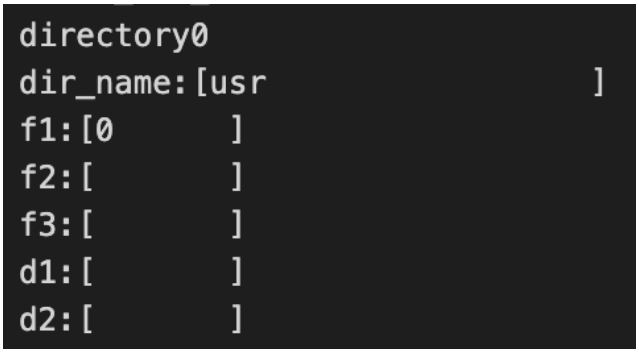


Figure 4- File linking in a directory

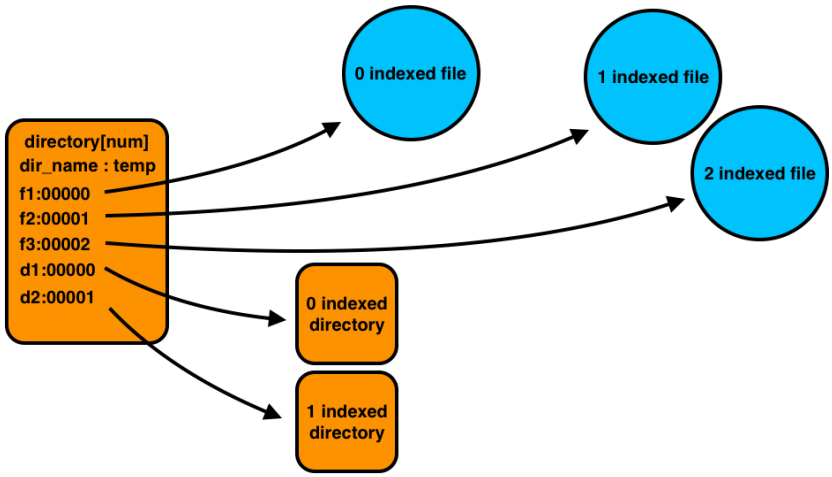


Figure 5-Simple Directory System Layout

Definition of keeping free blocks and free i-nodes

In the file system, I reserved 1 block space under the name of free_space_management to track free blocks and free i-nodes. I initialize the the initial value with all of free space management parameters. These parameters are as follows:

1. **Free inodes :** Holds the current number of free inodes in the file system.
2. **Free blocks :** Holds the current number of free blocks in the file system.
3. **Free directories :** Holds the current number of free directories in the file system.
4. **First free inode :** Holds the first free inode's index/number.
5. **First free directory :** Holds the first free directory's index/number.
6. **First free data block :** Holds the first free data block's index/number.
7. **First free root file :** Holds the first free root file's index/number.
8. **First free root directory :** Holds the first free root directory's index/number.

First of all, these parameters are initialized by the makeFileSystem program when the file system occurs. The initialized parameters are updated as the changes come according to the file operations performed in the fileSystemOper program.

```
free_space_management
free_inodes:399
free_blocks:432
free_directories:231
first_free_inode:000002
first_free_directory:000002
first_free_data_block:000004
root_files:000000
root_directories:000002
```

Figure 6- Free space management block in myFileSystem.data

If the file or directory is deleted, its name is deleted and the necessary global variables change. Since it will be costly to completely delete the contents of the file or directory, the content (overwrites when a new file or directory is created) is not deleted. And when it is requested to be accessed again, it cannot be accessed because it controls by name occupancy.

Definition of i-node structure

In the file system, I reserved 10% of the entire file system size to keep inode entries and this block starts with “inode_block” title. So it takes approximately 102 kilobytes. Each inode entry occupies approximately 210 bytes. A inode points at most three data block and double indirect or triple indirect inodes in itself. And a inode entry includes some specification. These features are as follows:

1. **Inode number/address :** Specifies the order of the inode.
2. **file name :** Specifies the name of the containing file. The name size can be 20 characters/bytes at most
3. **Data 1 :** It contains the data block number/address of the file in the inode. The number size can be 7 characters/bytes at most.
4. **Data 2 :** It contains the data block number/address of the file in the inode. The number size can be 7 characters/bytes at most.
5. **Data 3 :** It contains the data block number/address of the file in the inode. The number size can be 7 characters/bytes at most.
6. **Single indirect :** It contains the single indirect block number/address. The number size can be 7 characters/bytes at most.
7. **Double indirect :** It contains the double indirect block number/address. The number size can be 7 characters/bytes at most.
8. **Triple indirect :** It contains the triple indirect block number/address. The number size can be 7 characters/bytes at most.
9. **Size :** It contains size of the file.
10. **Date :** It contains last modification date and time.

When creating a new file, depending on the size of the copied file, the files up to 3 blocks are written into the data blocks indicated by the data1, data2 and data3 parameters in 1 inode. For files larger than 3 blocks, indirect parameters are used.

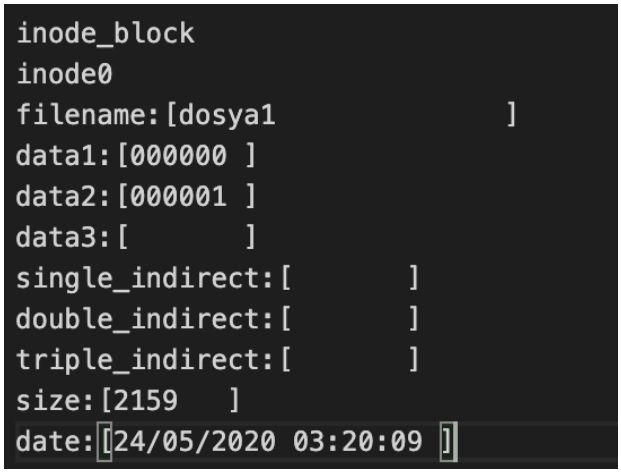


Figure 7- Sample inode entry that contains "dosya1" file

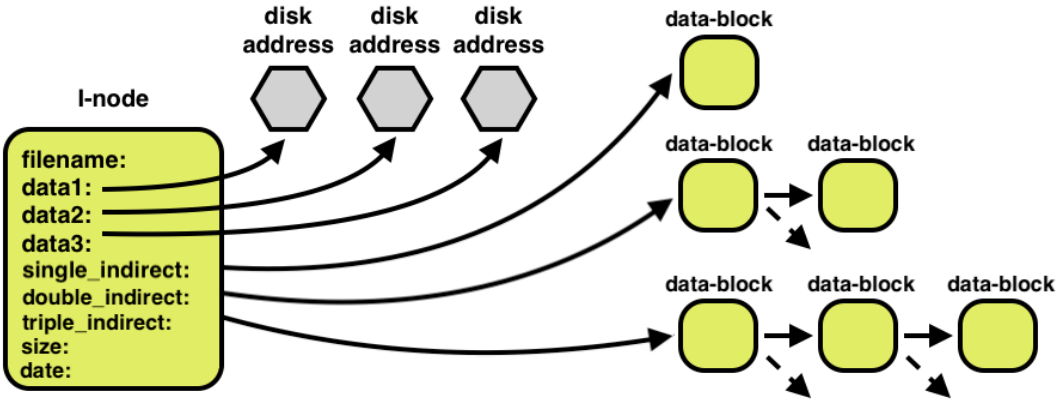


Figure 8- Simple Inode System Layout

Definition of Superblock

In the file system, I reserved one in first block of the entire file system size to keep inode entries and this block starts with “superblock” title. Also it creates by makeFileSystem program according the block size and number of inodes. There are crucial informations for file system in it.These informations are as follows:

1. **Magic Number :** Referred to as a file signature. In my file system magic number is 0.
2. **Number of blocks :** Holds the total total number of blocks in the file system.
3. **Number of directory blocks :** Holds the total number of directory blocks in the file system.
4. **Number of inodes :** Holds the total number of inodes (according to user input in makeFileSystem) in the file system.
5. **Number of directories :** Holds the total number of directories.(It calculated according to directory entry size and number of directory entry block)
6. **Block size :** The size of a block (Given by makeFileSystem)
7. **Inode block starts :** Holds the byte where the inode block starts.
8. **Free space management block starts :** Holds the byte where the free space management block starts.
9. **Root directory block starts :** Holds the byte where the root directory block starts.
10. **Directory block starts :** Holds the byte where the directory block starts.
11. **Data block starts :** Holds the byte where the data block starts.

All of these informations provided by makeFileSystem program. And they are not changeable.

```
superblock
magic_number:0
nof_blocks:512
nof_directory_blocks:25
nof_inodes:400
nof_directories:232
block_size:2048
inode_block_starts:2048
free_space_mgmt_block_starts:106496
root_directory_block_starts:108544
directory_block_starts:110592
data_block_starts:161792
```

Figure 9- Superblock for 400 inode and 2KB block size

Total File System Placement

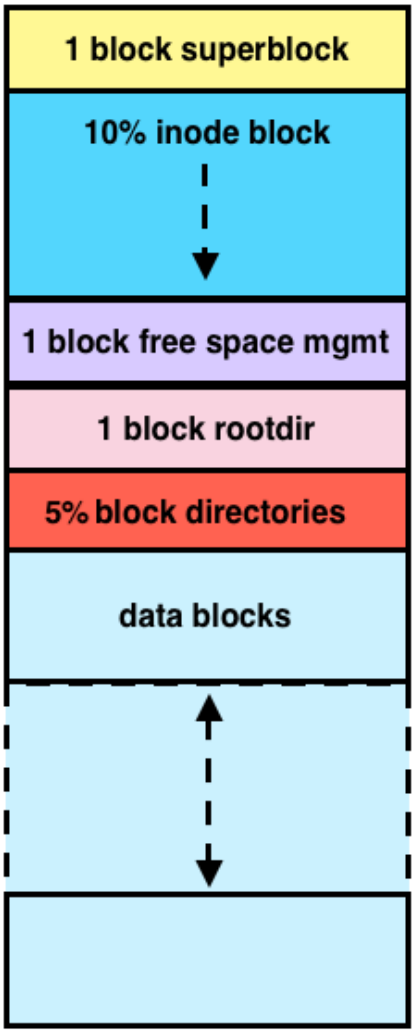
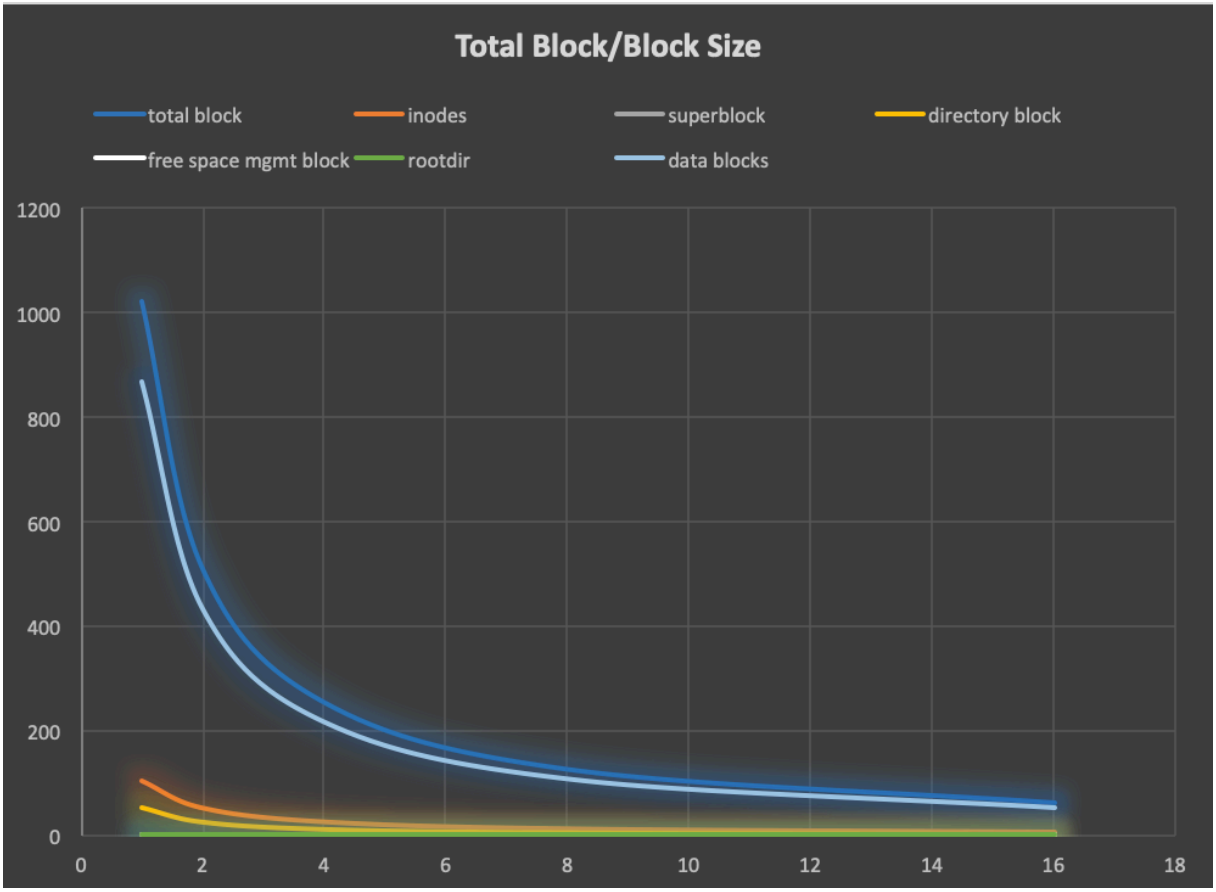


Figure 10-Block placement of entire file system

Distribution of block sizes by total size

block size	total block	inodes	Superblock	directory block	free space mgmt block	rootdir	data blocks
1KB	1024	102	1	51	1	1	868
2KB	512	51	1	25	1	1	433
4KB	256	25	1	12	1	1	216
8KB	128	12	1	6	1	1	107
16KB	64	6	1	3	1	1	52

Plot of Distribution



Function names in Part 3

Helper functions

- int get_inode_or_dir_num(FILE *fp,char *path,int is_file,int loop_reduce);
- int search_string_in_file(FILE *fp,int lower_bound,int upper_bound,char* str);
- int number_of_char_in_string(char ch,char* str);
- void get_value_of_string(FILE *fp,char* search_string,char* output);
- void get_value_of_string_range(FILE *fp,char* search_string,int lower_bound,int upper_bound,char* output);
- int change_the_value_of_string(FILE *fp,char* search_string,int change_value);
- void change_the_name_of_string(FILE *fp,char* main_string,char* search_string,char* name,int range);
- void add_rootdir(FILE *fp,int is_file,char* name);

Main functions

- void mkdir(FILE *fp,char path[]);
- void rm_dir(FILE *fp,char* path);
- void write_disk(FILE *fp,char* path,char* filename);
- void read_disk(FILE *fp,char* path,char* filename);
- void list(FILE *fp,char* path);
- void del(FILE *fp,char* path);
- void dumpe2fs(FILE *fp);
- int main(int argc,char *argv[]);