Gebze Technical University
Department of Computer Engineering
CSE 312 /CSE 504
Operating Systems Spring 2022
HW3

Mehmet Avni CELIK
1801042630

In this project, Simplified UNIX System 7 file system will be implemented in C by following Modern Operating System 4th edition.
Unfortunately some of the requirements could not be implemented.

I-Nodes:

It is a method of keeping track of which blocks belong to which file is to associate with each file a data structure called an i-node (index-node), which lists the attributes and disk addresses of the file's blocks.
If each i-node occupies n bytes and a maximum of k files may be open at once, the total memory occupied by the array holding the i-nodes for the open files is only kn bytes.

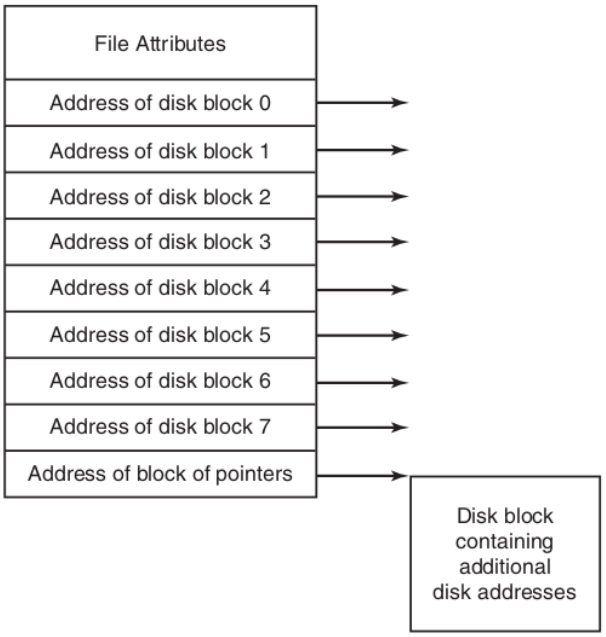

Figure 4-13. An example i-node.

In the first part of the assignment, following structures are required;

- I-node structure
- Directory structure
- Directory entries
- Free-block location
- Superblock structure

Implementation of the Structure:

1. **I-Node:**
   "Your file attributes will include size, file creation time, last modification date and time, and name of the file."
   I-node structure can include many more informations about files. But only required ones and some other important metadata are defined by considering textbook's definition.

```
typedef struct st_inode
{
    unsigned int file_size;

    unsigned int file_creation_second;
    unsigned int file_creation_minute;
    unsigned int file_creation_hour;

    unsigned int last_modify_second;
    unsigned int last_modify_minute;
    unsigned int last_modfiy_hour;

    unsigned int last_modify_day;
    unsigned int last_modify_month;
    unsigned int last_modfiy_year;

    unsigned int indirect_block_single;
    unsigned int indirect_block_double;
    unsigned int indirect_block_triple;

    unsigned int inode_location;    //keeps
    bool isFile;                    //Since


    unsigned file_page_number; // in i-nodes
    char *owner;

}_inode;
```

2. **Directory Structure:**
   Even though the file system does not include any special directory structure, since it is asked in the homework I tried to define it with some important data as well. Every file has an i-node and also a directory is some kind of a file. So it also has an i-node with other shown information.

```
typedef struct st_directory
{
    _inode inode;
    char *directory_path;
    unsigned int size;
    bool isEmpty;
}_directory;
```

3. **Directory Entries:**
   UNIX directory entry contains one entry for each file in that directory. Each entry is extremely simple because UNIX uses the i-node scheme illustrated in Fig. 4-13. A directory entry contains only two fields: the file name (14 bytes) and the number of the i-node for that file (2 bytes), as shown in Fig. 4-32. These pa-rameters limit the number of files per file system to 64K.

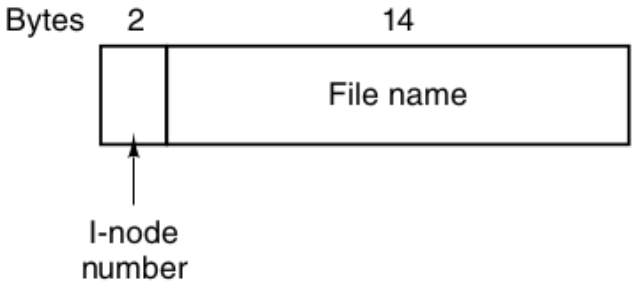

Figure 4-32. A UNIX V7 directory entry.

Also in my definition i-node size and file-name size are limited by the explained amounts.

```
typedef struct st_directoryEntry
{
    unsigned char file_name[14];
    int8_t inode_number;

}_directoryEntry;
```

4. **Free-Blocks:**
   Free-blocks can be kept in 2 different way in UNIX V7 File System. First one is to keep them in free-list by using linked-list structure. Second solution is keep them in fixed size bitmap array. In my definition both is defined in superblock structure.

```
//unsigned int *free_block_list;      //free blocks can be kept in free block list
unsigned int bitmap_blocks[TOTAL_BITMAP_BLOCKS_NUMBER];     //free blocks can be also kept in bitmap as array
```

5. **Superblock:**
   The file systems contain some of the items shown in Fig. 4-9. The first one is the **superblock**. It contains all the key parameters about the file system and is read into memory when the computer is booted or the file system is first touched. Typical information in the superblock includes a magic number to identify the file-system type, the number of blocks in the file system, and other key administrative information.

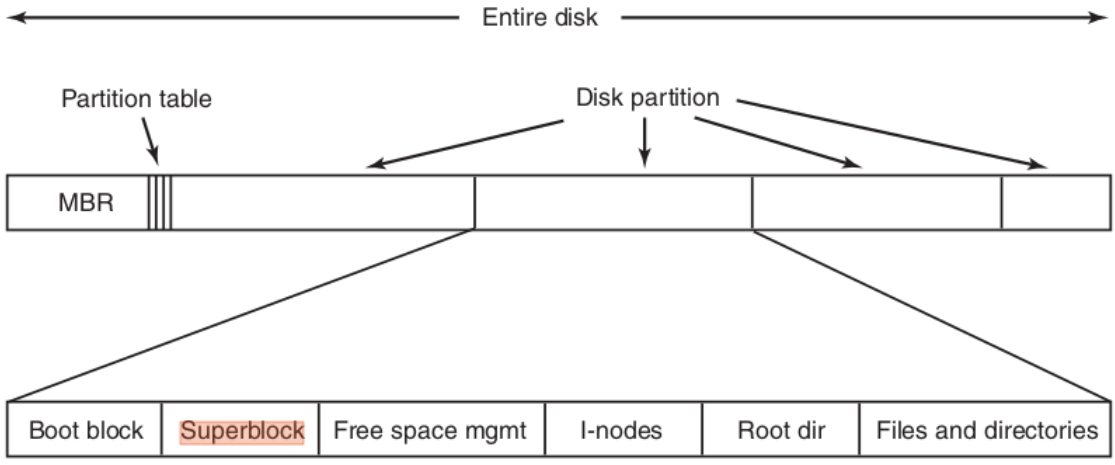

Figure 4-9. A possible file-system layout.

In my definition superblock is defined with the structure below:

```
typedef struct st_superblock          //superblock is essentially a system metadata
{
    unsigned int magic_number;        //the magic number to identify the system type (executable or not or something else).
    unsigned int number_of_blocks;    //keeps the total number of blocks.
    unsigned int number_of_datablocks; //keeps the total number of datablock.
    unsigned int number_of_inodes;    //keeps the total number of inodes.
    unsigned int block_size;          //keeps the block size.
    unsigned int root_directory_position;//keeps the root directory position.
    unsigned int data_block_position;    //keeps the position of datablocks.
    //unsigned int *free_block_list;     //free blocks can be kept in free block list
    unsigned int bitmap_blocks[TOTAL_BITMAP_BLOCKS_NUMBER];     //free blocks can be also kept in bitmap as array
    unsigned int first_block_address;
    unsigned int first_inode_address;
} superblock;
#endif
```

Execution of the Program:
The program should be executed with the command below:

>make
>./makeFileSystem 4 mySystem.dat

“4” is optional, it can be changed with any size but it should be changed as 16MB maximum.

Command line arguments are parsed.

- The program works with 3 command line arguments. Any less or more arguments will throw an exception and will be terminated.

```
if (argc!=3) {
    fprintf(stderr, "Program is terminated. Invalid arguaments!\n");
    exit(EXIT_FAILURE);
}
```

- The first command line argument is multiplied with 1024 to calculate the size in terms of MB. If it is larger, the program will throw an exception and will be terminated.

```
block_size = atoi(argv[1]) * 1024;
if(block_size>16384){       //to make it max 16 MB
    fprintf(stderr, "Program is terminated. A file can not be larger than 16MB!\n");
    exit(EXIT_FAILURE);
}
```

- The first command line argument is taking the file name and checking its size to see if the size is larger than it should be (14byte) or not. If it is larger, the program will throw an exception and will be terminated.

```
file_name = argv[2];
if(strlen(argv[2])>FILE_NAME_SIZE){
    fprintf(stderr, "The program is terminated. File name is too long!\n");
    exit(EXIT_FAILURE);
}
```

- The file is created, if the pointer that keeps the file is NULL, the program will throw an exception and will be terminated.

- Unfortunately I couldn’t implement the 2nd and 3rd part :(

Thank You for Your Time…

Mehmet Avni ÇELİK
1801042630