CSE-312 OPERATING SYSTEMS

SPRING 2024

HOMEWORK 2

REPORT

ONUR ATASEVER

210104004087

In this Project simplified FAT-12 system is implemented. There are 4096 entry since it is FAT-12 (2^12). According to input, block size changes between 0,5 and 1 KB. In this file system, files are stored in the disk, permissions are changed, password is changed.

In the makeFileSystem according to input, block size is written to the superblock and the root is created and it is written to the file. Rest of the file is filled with blanks to provide 4MB or 2MB storage.

In this Project structure each line in the file represents a block which is 1 or 0.5 KB. Each block can contain directory or file in it. When the directory or file is created, it's attributes are written to the file. Attributes are the name of the file, size, is directory bit, read and write permission bits, is protected bit fort he password, password and next block if it is connected with another block and date time.

First block of the file system is superblock which contains the size of a block. When the fileSystemOper program runs, it sets the global block size variable according to superblock.

In this file structure main idea is using delimeters. ';' is used fort his Project and some fixed sizes are used. 20 charachters are allocated for file name and 10 charachters are allocated for password. And the block size is fixed 0,5 KB or 1 KB.

Main logic:

```
init(fileSystem);
if (strcmp(operation, "mkdir") == 0)
{
    int file_block_num;
    // önce verilen path var mı kontrol et
    if ((return_block_number = isExist(fileSystem, splitted_path, number_of_path - 1, path_counter, block_number)) == -2)
    {
        // printf("Path not found return block number: %d\n", return_block_number);
        printf("Path not found\n");
        return 1;
    }
    else
    {
        // printf("Path found return block number: %d\n", return_block_number);
        if ((file_block_num = isSameName(fileSystem, splitted_path[number_of_path - 1], return_block_number)) != -1 )
        {
            printf("Same name found\n");
        }
        else
        {
            // printf("Same name not found\n");
            // Burda ekleme yapılcak
            mkdir_operation(fileSystem, splitted_path[number_of_path - 1], return_block_number, 1);
        }
    }
}
```

It checks if given path is exist. For example if given path is /dira/dirb/dirc first it checks if dira exist then if it is, it recursively checks if the rest of them exist. İf it is not exist it returns error code. Then if path is available, it checks if the given name desired to create exists, if there is a file or directory with same name it returns an error code. If ther eis not anyhing with same name it calls mkdir function.

```
int isExist(const char *fileSystem, char **splitted_path, int number_of_path, int path_counter, int block_numbe
    while ((readByte = read(file, &read_char, 1)) > 0)
    {
        if (read_char == '\n')
        {
            i = -1;
            current_line++;
            return -2;
        }
        if (read_char != ';' && read_char != '\n')
        {
            asd[i] = read_char;
        }
        else if (read_char == ';' && read_char != '\n')
        {
            asd[i] = '\0';
            //  printf("tokenize gitmeden önce: i:%d *%s*\n", i, asd);

            tokenize(tokens, asd);
            if (strcmp(tokens[0], splitted_path[path_counter]) == 0)
            {
                path_counter++;
                if (path_counter == number_of_path)
                {
                    close(file);
                    return atoi(tokens[1]);
                }
                int return_value = isExist(fileSystem, splitted_path, number_of_path, path_counter, atoi(tokens[1])
                close(file);
                // printf("return_value: %d\n", return_value);
                return return_value;
            }
            i = -1;
        }
```

isExist function finds directories according to file structure. İt saves from file until see ‘;’. When ’;’ is reached it means a directory informations are obtained one by one. When the informations come, it tokenizes all the things according to delimeter ‘,’. And it checks if the names are matches. İf it returns -2 it means it could not find the path which means path is not exist. It checks directories and sub directories recursively.

In the mkdir function it calls findFirstAvailableBlock function and it searchs FAT table. When it finds first empty places it returns it’s block number.

```
int mkdir_operation(const char *fileSystem, char *name, int block_number, int isDirectory)
    char *tokens[3];
    char read_char;
    int readByte;

    //write attributes to the beginning of the block
    write(file, buffer, ATTRIBUTES_LENGTH);
    go_to_line(file, block_number); // parent directory'e git
    int semicolonNumber;
    off_t semicolon_offset = find_last_semicolon(file, block_number, &semicolonNumber);

    //go to beginning of the file and move to last semicolon
    go_to_line(file, block_number);
    go_to_offset(file, semicolon_offset);
    int count = 0;
    int number = firstAvailableBlock;

    while (number != 0)
    {
        number /= 10; // Sayıyı 10'a böl
        count++;
    }
    char space2[4];
    for (i = 0; i < 4 - count; i++)
    {
        space2[i] = ' ';
    }
    space2[i] = '\0';
    sprintf(buffer, "%s%s,%d%s,1;", name, space, firstAvailableBlock, space2);

    write(file, buffer, 28);
    close(file);
    return firstAvailableBlock;
```

It goes end of the directory and writes new file or directory after semicolon. Then it goes the beginning of the it’s own block number and writes it’s attributes according to fixed sizes.

Fort the dir function it checks if the given path is exist. İf it is exist it calls dir_operation function with it’s block number.

```c
int dir_operation(const char *fileSystem, int block_number)
    {
        if (read_char == '\n') //Satır sonuna geldi demek ki file ve directoryler bitti
        {
            i = -1;
            current_line++;
            return -2;
        }
        if (read_char != ';' && read_char != '\n')
        {
            asd[i] = read_char;
        }
        else if (read_char == ';' && read_char != '\n')
        {
            asd[i] = '\0';

            tokenize(tokens, asd);
            off_t position = lseek(file, 0, SEEK_CUR);
            if (position == (off_t)-1)
            {
                perror("lseek");
                close(file);
                return EXIT_FAILURE;
            }
            char *attributesTokens[20];
            getAttributes(atoi(tokens[1]), file, attributesTokens);
            printf("name: %s, ", attributesTokens[0]);
            printf("drw: %s, ", attributesTokens[2]);
            printf("protected: %s, ", attributesTokens[3]);
            printf("creation date time: %s, ", attributesTokens[6]);
            printf("modification date time: %s\n", attributesTokens[7]);
            lseek(file, position, SEEK_SET);
            i = -1;
        }
```

It read from file. It goes directly specified block number. ‘;’ is a delimeter fort his structure. Each ; means an directory or file informations. Then it tokenizes and prints informations.

Addpw function checks if the path is exist. If it is exist it goes the given file’s block number. According to structure it moves fixed size and it changes protected bit from 0 to 1 and it moves and it changes password.

Each write and read operation checks protected bit before operation. If it is 1, it means it is protected and it gets password input from user. If it matches, it continues to the operation.

Write function checks if the given path is exist. If it is exist it goes to block number with FAT table. To write to the data section it finds the last ‘;’ delimeter. And it writes after. While writing it checks if the blocks size is exceed. If it is exceed, it finds new available block and it updates FAT tablet o set a connection. And it continues on new block. Before this operations it checks if the file is protected.

```
int write_operation(const char *fileSystem, int block_number, char *file_name)
    while ((readByte = read(readFile, &read_char, 1)) > 0)
    {
        if (difference == BLOCK_SIZE - 1) // yeni block bul ve ordan devam et
        {
            int firstAvailableBlock = findFirstAvailableBlock(file);
            // printf("firstAvailableBlock: %d\n", firstAvailableBlock);
            // printf("difference: %ld\n", difference);
            int number = firstAvailableBlock;
            if (firstAvailableBlock == -1)···
            go_to_line(file, current_block);
            go_to_offset(file, 49);
            char block_index[6]; // 20 byte genellikle int değeri için yeterli olacaktır
            int count = 0;
            while (number != 0)···
            char space2[5];
            for (i = 0; i < 5 - count; i++)···
            space2[i] = '\0';
            sprintf(block_index, "%d%s", firstAvailableBlock, space2);
            // printf("block_index: *%s*\n", block_index);
            write(file, block_index, 5);
            go_to_line(file, block_number);
            char file_attributes[ATTRIBUTES_LENGTH + 1];
            read(file, &file_attributes, ATTRIBUTES_LENGTH);
            file_attributes[49] = '-';
            file_attributes[50] = '1';
            file_attributes[51] = ' ';
            file_attributes[52] = ' ';
            file_attributes[53] = ' ';
            go_to_line(file, firstAvailableBlock);
            write(file, file_attributes, ATTRIBUTES_LENGTH);
            current_block = firstAvailableBlock;
            difference = 95;
```

It checks block size. If it is exceed it finds new available block and it copies all the attributes to the new block which is connected with old block. Old block's Fat table is updated and it goes until all the file is written to the file system.

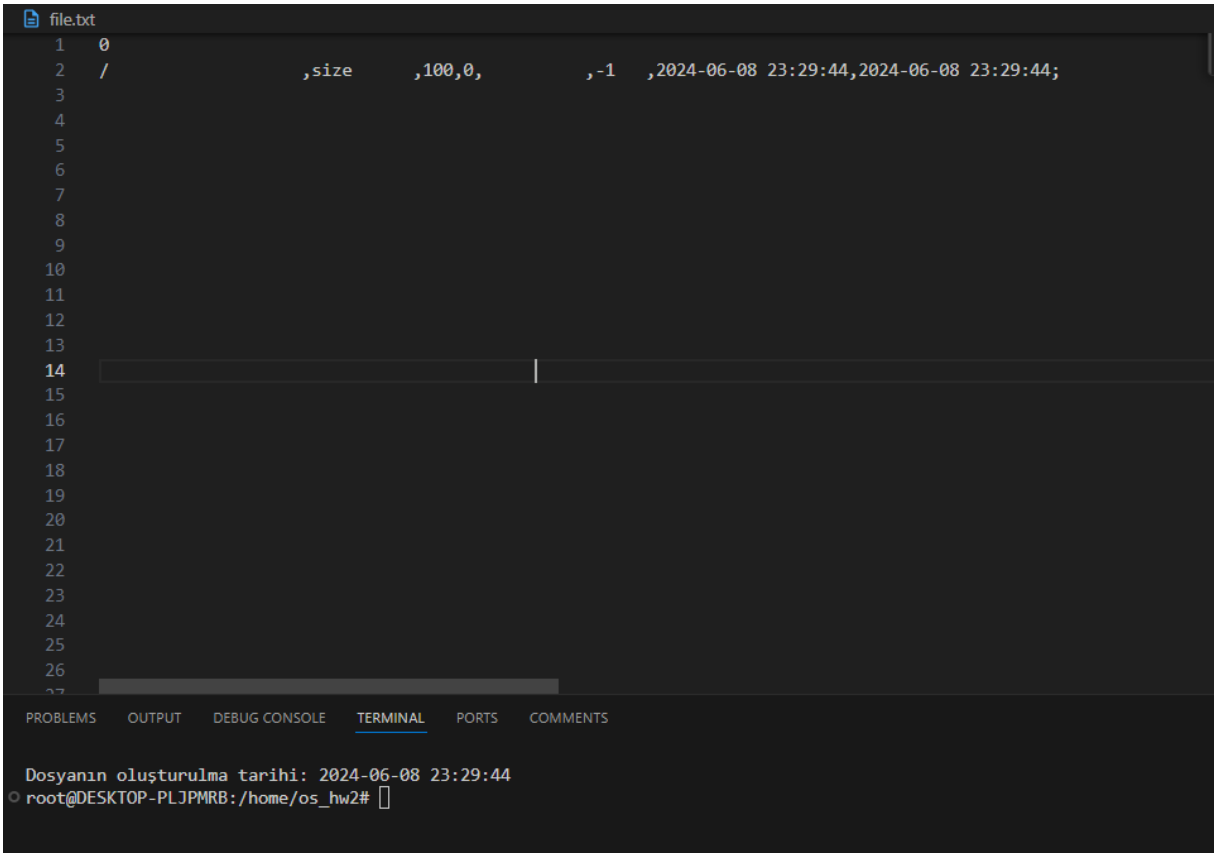
Chmod operation checks given parameter. If the first element of the parameter is '+' it gives read and write permission. It sets rw bits to 1. If it is '-', it sets read and write permissions to '0'.

```
int chmod_operation(const char *fileSystem, int return_block_number, char *permission)
{
    int file = open(fileSystem, O_RDWR);
    if (file == -1)
    {
        perror("open");
        exit(-1);
    }

    int i = 0;
    go_to_line(file, return_block_number);
    off_t offset = lseek(file, 33, SEEK_CUR);
    if(permission[0] == '+')
    {
        char entry = '1';
        write(file, &entry, 1);
        write(file, &entry, 1);
    }
    if(permission[0] == '-')
    {
        char entry = '0';
        write(file, &entry, 1);
        write(file, &entry, 1);
    }

}
```

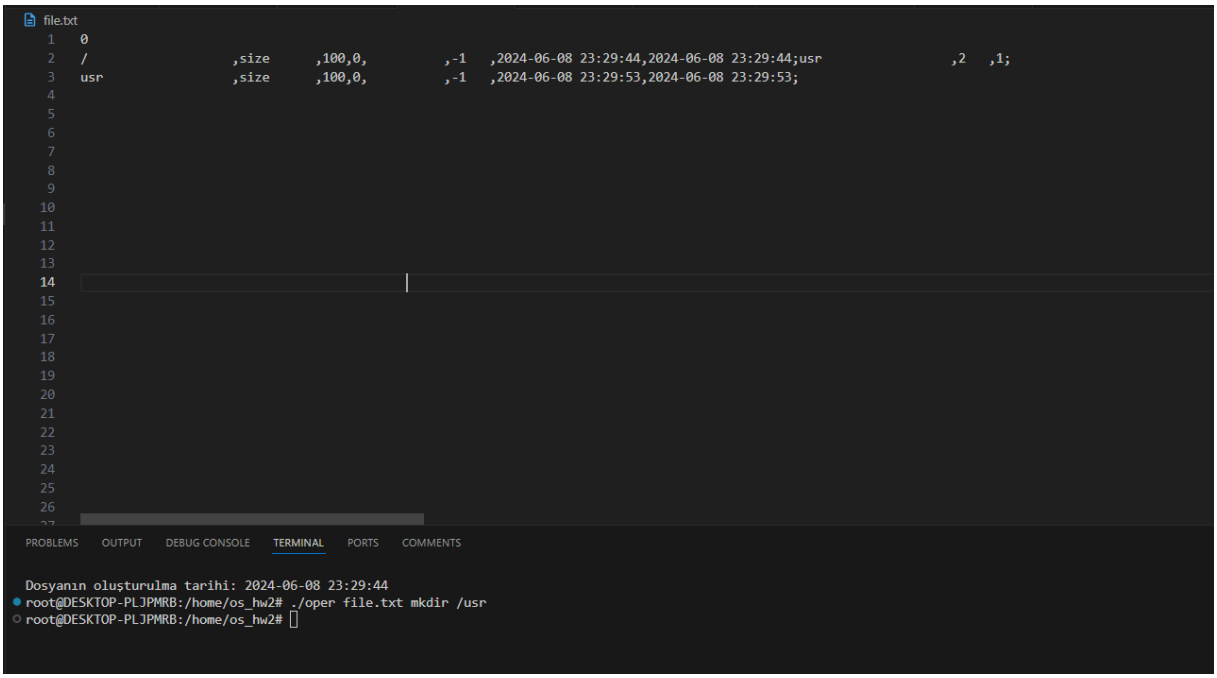
In the read function it checks if given path is exist. If it is, it goes the specified block and it goes beginning of the data section by using fixed size feature. It starts to read and write to the file until it reads ‘\n’. If it reads new line it checks fat table entry to determine if there is connection between this block and another block. If the value is different than -1 it means there is a connection and it goes to new block and it moves to data section. It read until see ‘;’. Because ‘;’ indicates that it is end of the data.

```
int read_operation(const char *fileSystem, int block_number, char *file_name)
    char read_char;
    int readByte;
    int current_block = block_number;
    while ((readByte = read(file, &read_char, 1)) > 0)
    {
        if (read_char == ';')
            break;
        if (read_char == '\n') // yeni block bul ve ordan devam et
        {
            //Fixed size olarak data section başına git
            go_to_line(file, current_block);
            go_to_offset(file, 49);
            char next_block[6];
            read(file, next_block, 5);
            int new = atoi(next_block);
            go_to_line(file, new);
            char file_attributes[ATTRIBUTES_LENGTH + 1];
            //attribute'ları oku
            read(file, &file_attributes, ATTRIBUTES_LENGTH);
            current_block = new; //current block u güncelle
            if (current_block == -1)
                break;
        }
        else
        {
            write(writeFile, &read_char, 1);
        }
    }
    close(file);
    close(writeFile);
    return 0;
}
```

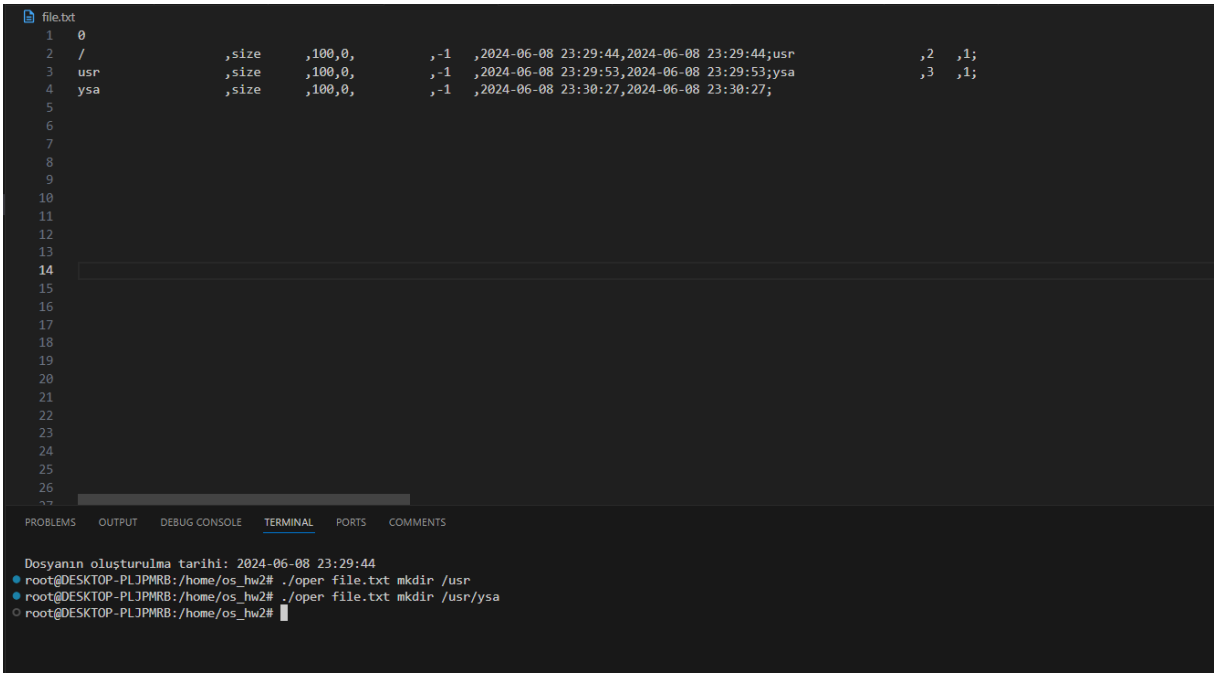
makeFileSystem creates file system with 0.5 KB size with name file.txt:



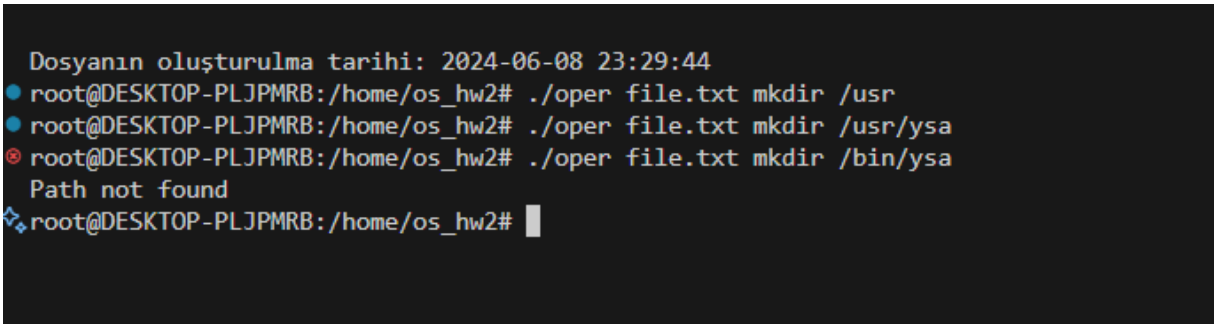
It creates new directory. It adds under the root directory with block number and it add new directory with attributes to the first available block:



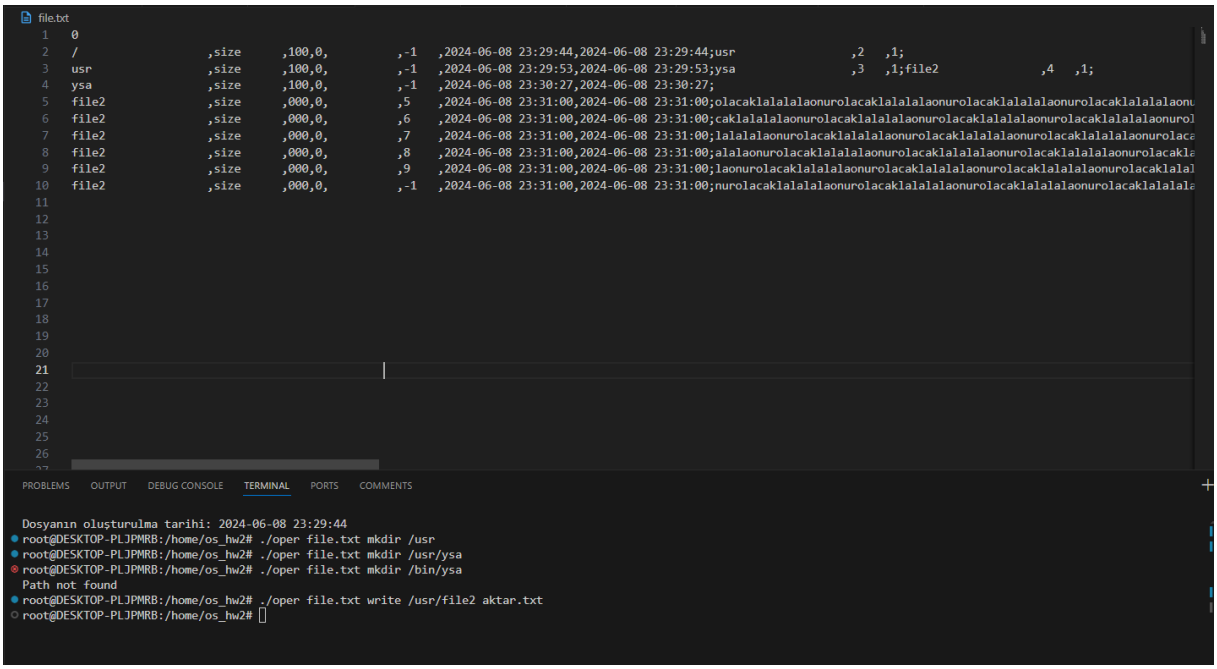
It adds ysa directory under usr directory:



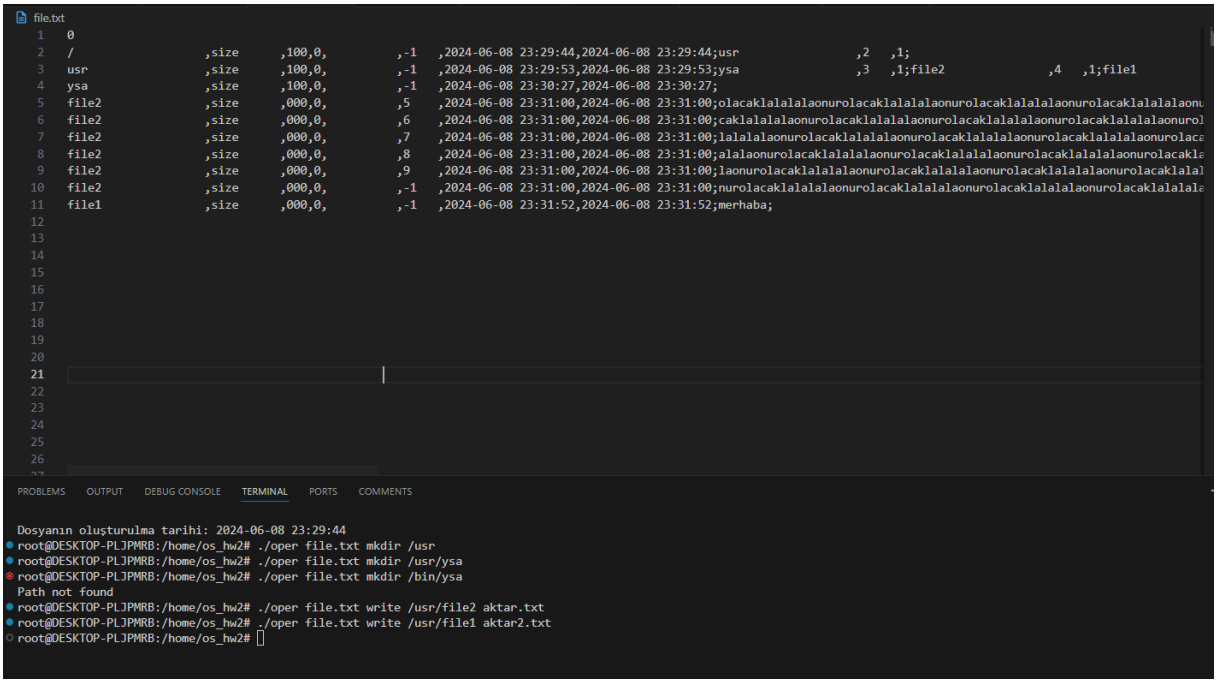
It tries to add to a path which is not exist:



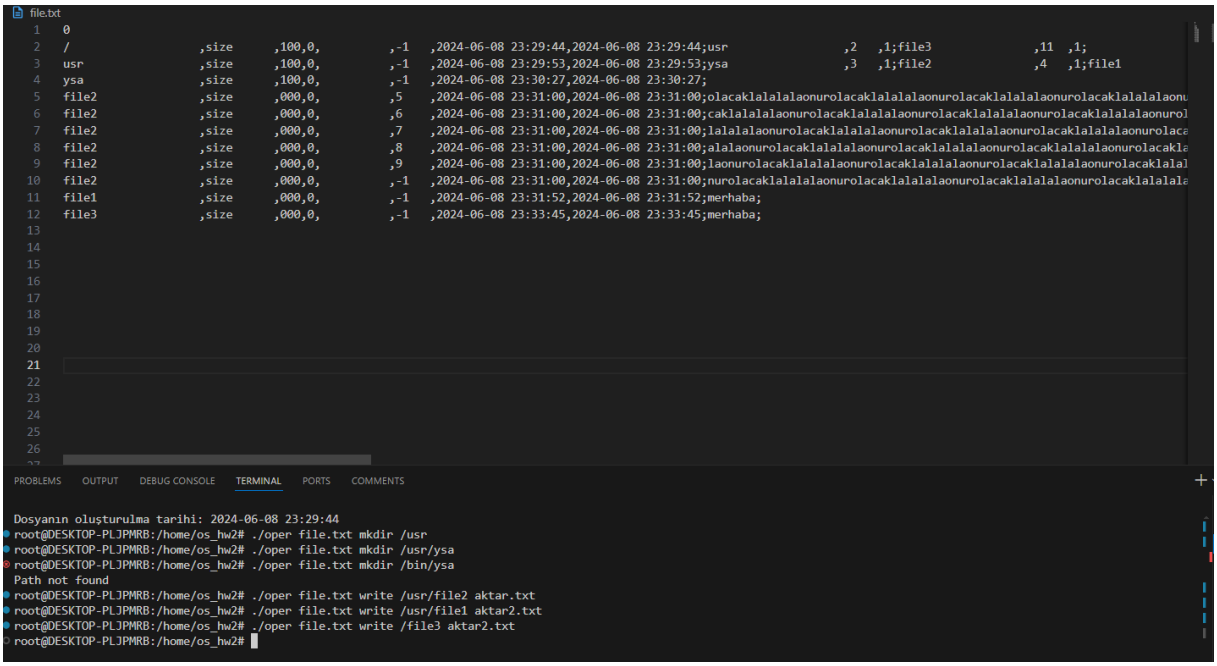
It tries to write from file which has 2484 byte in it to the file system. Since block size is 0.5 kb it uses more than one block:



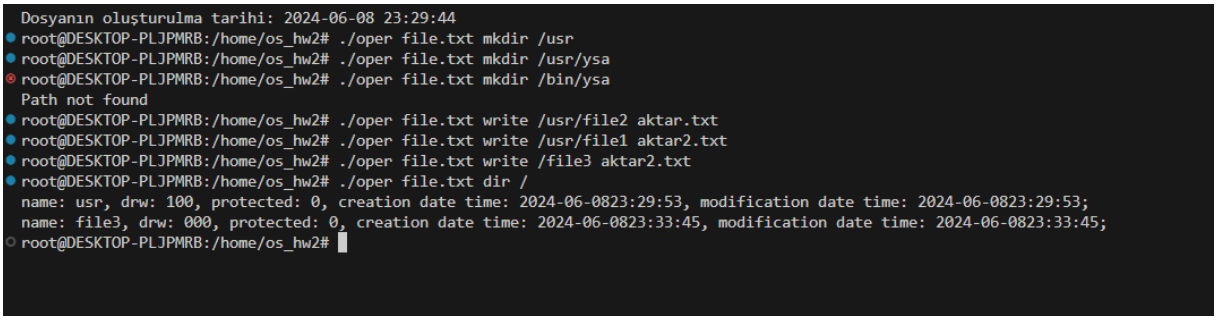
It writes one more block by using another file:



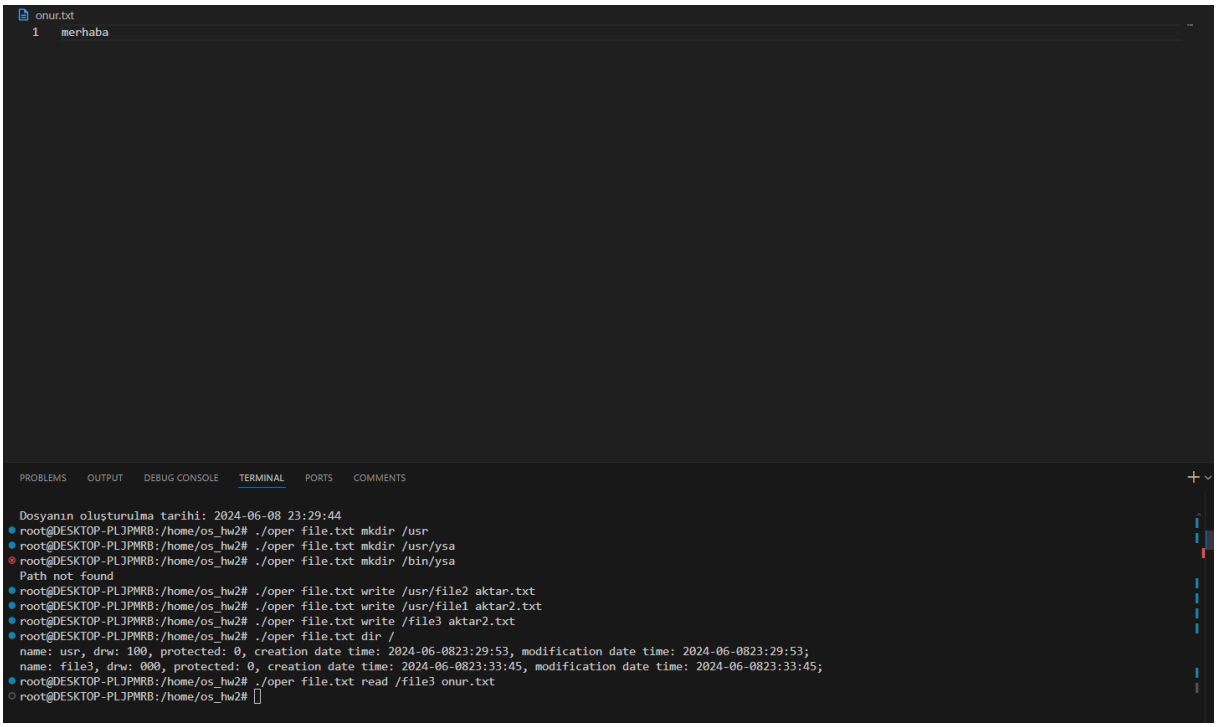
It writes directly under root:



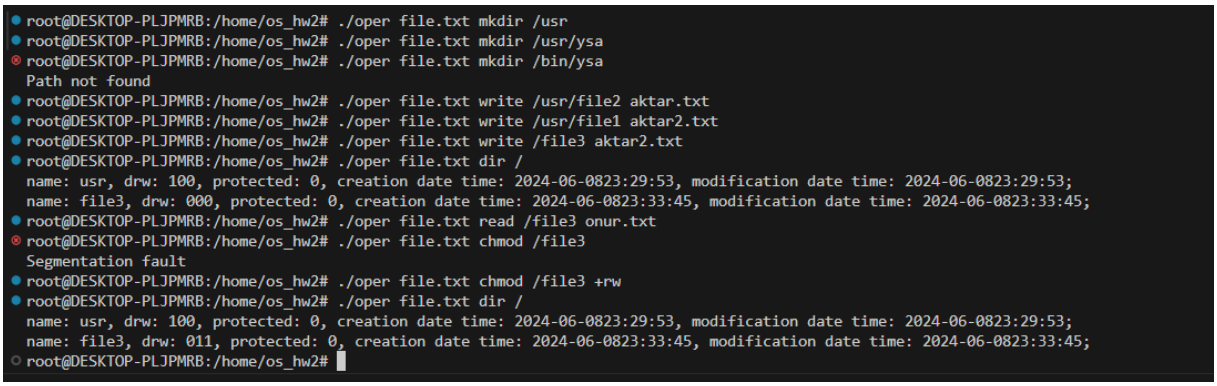
İt lists all the file and directories:



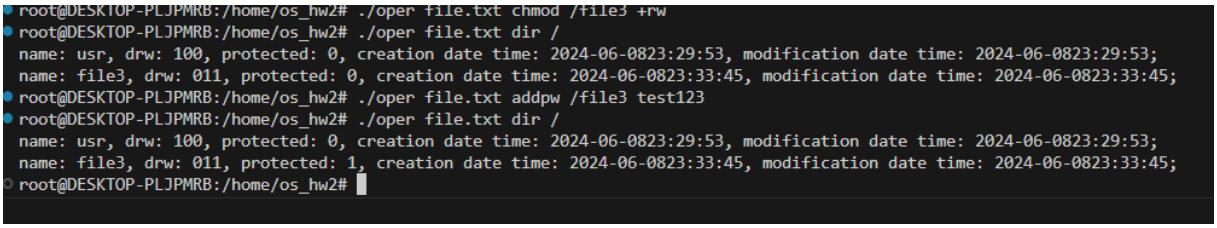
It reads from file system and writes regular file:



With chmod as you see permission bits are changed:



Protected bit is changed:



It asks password: