Cristiam David Martin Jackson **22.01.2019**

Laboratory Nr. 6 (Report)

"Konstrukcja systemu plików"
"Construction of a file system"

1. Purpose

Write in the Minix system, a program in C language and a script showing how to use a file system implemented in a virtual disk of a given size.

The structure of directories should be single-leveled, basic file operations functions should be implemented.

2. Overview

The presented solution will be structured as follows:

- As first block, a 'super-block', containing all important information belonging to the file system, such as: size of the disk, number of files, number of blocks and parameters to generate the file allocation system.
- The file allocation system, contains information about the files, their name, size and address where the file first block of the required file can be found.
- Blocks, they consist of nodes belonging to a linked list with two links, the first link connects the physical block, the second link connects the different blocks belonging to the same file, the end of the file is denoted by a NULL when attempting to link the next node.

3. Solution

Create virtual disk: this step initializes the main characteristics of the disk, as well as creates the blocks where the files will be stored. The blocks consist of a linked list structure, the most important parameters are: block id, next block (by id), next block (by file). The disk is initialized with the number of blocks, calculated from the constant BLOCKSIZE, which happens to be 8 bytes. Every block is assigned a single bit in the 'available' space, in order to check if a file can be stored or not in there.

```
struct vd{
        int size;/*In bytes*/
        int files;
        int firstaddr;
        int blocks;
        int freeblocks;
        int blockid;
        int available;
        struct vd * realnxt;
        struct vd * filenext;
};
struct vd *Sblock=NULL;
struct vd *current=NULL;
struct vd *end=NULL;
struct filesList{
        char name[15];
        int size;
        int address;
}fl[500];


void createList(void){
        int i;
        printf("Size assigned: %d Bytes\n", Sblock-> size);
        printf("Creating %d blocks\n", Sblock-> blocks);
        for (i=2;i<Sblock->blocks+2;i++){
                struct vd *new = (struct vd*) malloc(sizeof(struct vd));
                current->realnxt=new;
                current->available=1;
                current=new;
                end=new;
                current->blockid=i;
        }
        current=Sblock->realnxt;
        printf("First block's ID: %d\n",current->blockid);
        printf("Last block's ID: %d\n",end->blockid);
}

int availableblocks(int _size){
        int blocks=(_size-sizeof(Sblock))/BLOCKSIZE;
        return blocks;

void createVD(int _size){
        int blocks;
        struct vd *link = (struct vd*) malloc(sizeof(struct vd));
        link->size = _size*1024;
        link->files =0;
        link->firstaddr=2;
        link->blockid=1;
        blocks=availableblocks(_size*1024);
        link->blocks=blocks;
        link->freeblocks=blocks;
        link->available=0;

        Sblock = link;
        current = link;
        end = link;
        printf("\n\t\t------ PREPARING DISK ------\n");
        createList();
}
```

Create a file in the virtual disc: There are two ways to create files, manual (the name of the file can be selected) and automatic, which generates automatically names to the files, so that the test can be run in a loop. The size of every file is variable, between 8 bytes and 500 bytes. Before creating a file, the system validates if there is already a file in the table with the same name, in that case will send and error message, otherwise, proceeds to calculate the number of blocks required for the new incoming file, this amount of blocks is compared to the amount of available blocks present in super-block's characteristics. After checking the likeability for the file to be on the system, the assign blocks function is called. This function traverses the disk, looking for the first available block (through the realnxt node), once the first block is found, its state turns to no available and set a pointer waiting for the next free block, the algorithm continues traversing until the next available block is found, the new link between the pointer on the previous block and the current block is established, this process continues until all blocks are assigned, finishing the block's list with a NULL link, this way it is easily to verify where is the end of the file in the virtual disk.

```
char* generName(void){
        char * file;
        char temp[15];
        strcpy(file,"test");
        sprintf(temp,"%d",(rand() % (10 - 1 + 1)) + 1);
        strcat(file,temp);/*(rand() % (max - min + 1)) + min*/
        strcat(file,".y");
        return file;
}
int randomSize(void){

        /*Size of the files are between 8 and 500 bytes*/
        int max = 500;
        int min = 10;
        int num = (rand() % (max - min + 1)) + min;
        return num;
}
int assignBlocks(int blocks);
void createfileVD(char * file){
        int blocks;
        int size;
        int pos;
        int val;
        printf("Creating... %s\n",file);
        val=findAllocationbyName(file);
        /*printf("VALVALVALVAL %d\n",val);*/
        if(val == -1 ) {
                /*printf("File size: %ld\n",buffer.st_size);*/
                size=randomSize();
                blocks=calculateBlocks(size);
                if (blocks<Sblock->freeblocks){
                        printf("File: %s\tsize: %d\trequires: %d blocks\n",file,size,blocks);
                        /*printf("STRUCTURE FILES BEFORE ASSIGN BLOCKS:\n");*/
                        /*printf("%d\n",fl[Sblock->files-1].address);*/
                        pos=assignBlocks(blocks);
                        printf("Address in file table: %d\t",pos+1);
                        printf("Address in disk: %d\n\n",fl[pos].address);/*Assign values in
File Table*/
                        strcpy(fl[pos].name,file);
                        fl[pos].size=size;
                }
                else{
                        perror("Not enough free space in disk\n");
                }
        }
        else{
                if (val >= 0){printf("File: %s, already exists\n",file);}
                else {printf("File: %s, does not exist\n",file);}
        }
}

void automaticfilevd(void){
        createfileVD(generName());
```

Copy a file to Minix: This process checks the path where the files should be allocated in the Minix system, looks for the position of the file and the address where it is allocated in the virtual disk, the blocks are traversed until the end of the file, those blocks are not modified, the file is copy into the Minix system, with the same name and size as it was on the virtual disk.

```
int findAllocationbyName(char * file){
        int i;
        int j=0;
        int val=-2;
        for (i=0;j<Sblock->files;i++){
                if(fl[i].size=='\0'){
                /*printf("fl[%d]= NULL\n",i);*/
                }
                else{
                /*printf("fl[%d]: %d B\taddress: %d\tfile:
%s\n",i+1,fl[i].size,fl[i].address,fl[i].name);*/
                /*printf("fl[%d]= %s, file= %s\n",i,fl[i].name,file);*/
                                if (strcmp(fl[i].name, file)==0){
                                printf("File: %s, is in the system\n",file);
                                val=i;
                                }
                j++;
                }
        }
        val = (val>=0)?val:-1;
        return val;
}

void copytoMinix(char * file){
        int fd;
        int result;
        char str[80];
        int temp;
        printf("FLAGGLAL\n");
        temp=findAllocationbyName(file);
        printf("File: %s, size: %d\n",fl[temp].name,fl[temp].size);
        strcpy(str,MINIXPATH);
        strcat(str,file);
        printf("\nCopying to... %s\n",str);
        fd = open(str, O_WRONLY | O_CREAT | O_EXCL, (mode_t)0600);
        if (fd == -1) {
                perror("Error opening file for writing");
                /*return 1;*/
        }
        result = lseek(fd, fl[temp].size, SEEK_SET);
        if (result == -1) {
                close(fd);
                perror("Error calling lseek() to 'stretch' the file");
                /*return 1;*/
        }

        /* write just one byte at the end */
        result = write(fd, "", 1);
        if (result < 0) {
                close(fd);
                perror("Error writing a byte at the end of the file");
                /*return 1;*/
        }
        /* do other things here */
        close(fd);
}
```

Copy from Minix: It uses the same functions in order to get the number of blocks required, validates if there is already a file with the same name and if there is enough free space in the disk in order to copy the file. It is also used the first available space in the file table, so that there won’t be free spaces after deleting a file and adding a new one.

```
int calculateBlocks(int size){
        int blocks;
        blocks=size/BLOCKSIZE;
        return blocks;
}


int getPosFileTable(void){
        int i=0;
        while (fl[i].size!='\0'){
```

```
		i++;/*Look for first free space in table*/
	}
	return i;
}


int assignBlocks(int blocks){
	int i;
	int pos;/*First allocation space the file table available*/
	struct vd *temp = (struct vd*) malloc(sizeof(struct vd));
	Sblock->files+=1;
	current=Sblock->realnxt;
	temp=current;
	/*printf("Current block's ID: %d\n",current->blockid);*/
	/*Look for next free node*/
	for(i=0;i<blocks;i++){
		if(current != temp){
			current->filenext=temp;
			current=temp;
		}
		while(temp->available!=1){
			temp=temp->realnxt;/*Find closest available block*/
		}
		/*Assign address for file 1*/
		if(i==0){/*Save address in file table*/
		current=temp;/*current goes to the first available block found by 'temp'*/
		pos=getPosFileTable();/*Look for first free space in table*/
		fl[pos].address=temp->blockid;
		}
		temp->available=0;
		Sblock->freeblocks-=1;
	}/*for*/
	current->filenext=temp;
	current=temp;
	current->filenext=NULL;/*End of file*/
	printf("Last block's ID: %d\tBlocks assigned!\n",current->blockid);
	return pos;
}

void copyfromMinix(char * file){
	struct stat buffer;
	int       status;
	int blocks;
	int size;
	int pos;
	int val;
	char str[80];
	strcpy(str,MINIXPATH);
	strcat(str,file);/*Concatenate path*/
	printf("Copying... %s\n",str);
```

```
        status = stat(str, &buffer);
        val=findAllocationbyName(file);
        /*printf("VALVALVALVAL %d\n",val);*/
        if(status == 0 && val == -1 ) {
                /*printf("File size: %ld\n",buffer.st_size);*/
                size=buffer.st_size;
                blocks=calculateBlocks(size);
                if (blocks<Sblock->freeblocks){
                        printf("File: %s\tsize: %d\trequires: %d blocks\n",file,size,blocks);
                        /*printf("STRUCTURE FILES BEFORE ASSIGN BLOCKS:\n");*/
                        /*printf("%d\n",fl[Sblock->files-1].address);*/
                        pos=assignBlocks(blocks);
                        printf("Address in file table: %d\t",pos+1);
                        printf("Address in disk: %d\n\n",fl[pos].address);/*Assign values in
File Table*/
                        strcpy(fl[pos].name,file);
                        fl[pos].size=size;
                }
                else{
                        perror("Not enough free space in disk\n");
                }
        }
        else{
                if (val >= 0){printf("File: %s, already exists\n",file);}
                else {printf("File: %s, does not exist\n",file);}
        }
}
```

Delete files: this process first identifies the file from the table, retrieving its address, once got his address, traverses the disk until the starting point of the file, once there, traverses the file through the node ‘filenext’ until it reaches ‘NULL’, deleting all properties assigned to the nodes, hence, deleting the file.

```
/*Once validated, unassign the assigned blocks*/
void unassignBlocks(int address){
        current=Sblock->realnxt;
        printf("Current address: %d\n",address);
        /*Traverse until find address of file*/
        while (current->blockid!=address){
                /*printf("Current blockid: %d address:%d\n",current->blockid,address);*/
                current=current->realnxt;
        }
        do{
        current->available=1;
        Sblock->freeblocks+=1;
        printf("deleting block: %d\n",current->blockid);
        current=current->filenext;
        }while(current->filenext!=NULL);
        current->available=1;
        Sblock->freeblocks+=1;
        printf("deleting block: %d\n",current->blockid);
}
void deletefile(char * file){
        int temp,addr;
        char str[80];
        strcpy(str,MINIXPATH);
        strcat(str,file);/*Concatenate path*/

        printf("\nDeleting... %s\n",str);
        addr=findAllocationbyName(file);
        printf("Allocation in table: %d\n",temp);
        temp=fl[addr].address;
        if (temp<0){
                printf("File does not exist\n");
        }
        else{
                unassignBlocks(temp);
        }
        /*printf("befname%s\n",fl[addr].name);
        printf("befname%d\n",fl[addr].size);
        printf("befname%d\n",fl[addr].address);
        printf("TEMP%d\n",addr);*/
        Sblock->files-=1;
        fl[addr].size='\0';
        fl[addr].address=0;
        strcpy(fl[addr].name,"");

        /*printf("name%s\n",fl[addr].name);
        printf("name%d\n",fl[addr].size);
        printf("name%d\n",fl[addr].address);*/
}
```

Check current status of the file system: retrieve information of the super block, file table and blocks from the file system.

```
void exploreVDFiles(void){
        int i;
        int j=0;
        printf("\n\t\t------ FILE TABLE ------\n");
        for (i=0;j<Sblock->files;i++){
        /*for(i=0;i<20;i++){*/
                if(fl[i].size=='\0'){
                printf("fl[%d]: NULL\n",i+1);
                }
```

```
                    else{
                    printf("fl[%d]: %d B\taddress: %d\tfile:
%s\n",i+1,fl[i].size,fl[i].address,fl[i].name);
                    j++;
                    }
          }
}
void Sblockchars(void){
          printf("\n\t\t------ VIRTUAL DISK ------\n");
          printf("-Size: %d Bytes\n",Sblock->size);
          printf("-Number of blocks: %d\n",Sblock->blocks);
          printf("-Available blocks: %d\n",Sblock->freeblocks);
          printf("-Available space: %d Bytes\n",Sblock->freeblocks*BLOCKSIZE);
          printf("-Files: %d\n\n",Sblock->files);
}
void mapVD(void){
          int i,j,val;
          printf("\n\t\t------ DISK MAPPER ------\n");
          printf("*S ");
          current=Sblock->realnxt;
          for(i=2;i<Sblock->blocks+1;i++){
                    if(current->blockid%20==1){val=i/10;}
                    else{val=current->blockid%20;}
                    if(current->available==1){printf("-%d ",val);}
                    else{printf("*%d ",current->blockid%20);}
                    if (i%20==0){printf("\n");}
                    current=current->realnxt;
          }
          printf("\n");
          current = Sblock->realnxt;
}
```

Utilities: create files with the specified size and name in the specified path, those files do not contain any useful information.

```
void createFile(char * file, int sel){
          int fd;
          int result;
          char str[80];
          /*0 for VD else for MINIX*/
          if (sel==0)
                    strcpy(str,FILEPATH);
          else
                    strcpy(str,MINIXPATH);
          strcat(str,file);
          fd = open(str, O_WRONLY | O_CREAT | O_EXCL, (mode_t)0600);
```

```
	if (fd == -1) {
		perror("Error opening file for writing");
		/*return 1;*/
	}
	result = lseek(fd, randomSize()-1, SEEK_SET);
	if (result == -1) {
		close(fd);
		perror("Error calling lseek() to 'stretch' the file");
		/*return 1;*/
	}

	/* write just one byte at the end */
	result = write(fd, "", 1);
	if (result < 0) {
		close(fd);
		perror("Error writing a byte at the end of the file");
		/*return 1;*/
	}

	/* do other things here */

	close(fd);
}

int getSize(char * file,int sel){
	struct stat buffer;
	int         status;
	int size;
	char str[80];
	/*0 for VD else for MINIX*/
	if (sel==0)
		strcpy(str,FILEPATH);
	else
		strcpy(str,MINIXPATH);
	strcat(str,file);
	printf("Getting file's size: %s\n",str);
	status = stat(str, &buffer);
	if(status == 0) {
		printf("File size: %ld\n",buffer.st_size);
		size=buffer.st_size;
	}
	else{
		printf("File: %s, does not exist\n",file);
	}
	return size;
}

void createmanualFile(void){
	char str[15];
	int sel;/*0 for VD*/
```

```
        printf("Enter name of file: ");
        scanf("%s",str);
        printf("Create in VD (0): ");
        scanf("%d",&sel);
        printf("Input: %s\nSel: %d\n",str,sel);
        createFile(str,sel);
        getSize(str,sel);
}
```

4. Testing

In order to show all the tests, the following table contains all information obtained when running the file system on Minix. The test was also checked by using a loop until cause a error alarm, in the cases where the files do not exist, or if there is no enough free space in the disk in order to save the next incoming file.
Plus, the fragmentation can be seen when a file is added after a deleted file (red), those spaces are covered with the next file asking to be stored in the file system.

```
                ------ PREPARING DISK ------
Size assigned: 4096 Bytes
Creating 511 blocks
First block's ID: 2
Last block's ID: 512

                ------ VIRTUAL DISK ------
-Size: 4096 Bytes
-Number of blocks: 511
-Available blocks: 511
-Available space: 4088 Bytes
-Files: 0

Creating... first.y
File: first.y       size: 413       requires: 51 blocks
Last block's ID: 52     Blocks assigned!
Address in file table: 1            Address in disk: 2

Creating... second.y
File: second.y  size: 493       requires: 61 blocks
Last block's ID: 113    Blocks assigned!
Address in file table: 2            Address in disk: 53

Creating... test1.y
File: test1.y       size: 64        requires: 8 blocks
Last block's ID: 121    Blocks assigned!
Address in file table: 3            Address in disk: 114
```

```
Creating... test2.y
File: test2.y     size: 326         requires: 40 blocks
Last block's ID: 161    Blocks assigned!
Address in file table: 4            Address in disk: 122

Creating... test3.y
File: test3.y     size: 301         requires: 37 blocks
Last block's ID: 198    Blocks assigned!
Address in file table: 5            Address in disk: 162

Creating... test4.y
File: test4.y     size: 398         requires: 49 blocks
Last block's ID: 247    Blocks assigned!
Address in file table: 6            Address in disk: 199

Creating... test5.y
File: test5.y     size: 421         requires: 52 blocks
Last block's ID: 299    Blocks assigned!
Address in file table: 7            Address in disk: 248

Creating... test6.y
File: test6.y     size: 311         requires: 38 blocks
Last block's ID: 337    Blocks assigned!
Address in file table: 8            Address in disk: 300

FLAGGLAL
File: first.y, is in the system
File: first.y, size: 413

Copying to... /usr/usr/first.y
Copying... /usr/usr/testa.x
File: testa.x     size: 159         requires: 19 blocks
Last block's ID: 356    Blocks assigned!
Address in file table: 9            Address in disk: 338

Copying... /usr/usr/testb.x
File: testb.x     size: 126         requires: 15 blocks
Last block's ID: 371    Blocks assigned!
Address in file table: 10           Address in disk: 357

Copying... /usr/usr/testc.x
File: testc.x     size: 137         requires: 17 blocks
Last block's ID: 388    Blocks assigned!
Address in file table: 11           Address in disk: 372

Copying... /usr/usr/testd.x
File: testd.x     size: 169         requires: 21 blocks
Last block's ID: 409    Blocks assigned!
Address in file table: 12           Address in disk: 389
```

```
Copying... /usr/usr/teste.x
File: teste.x     size: 445         requires: 55 blocks
Last block's ID: 464    Blocks assigned!
Address in file table: 13          Address in disk: 410

Copying... /usr/usr/testf.x
File: testf.x      size: 157         requires: 19 blocks
Last block's ID: 483    Blocks assigned!
Address in file table: 14          Address in disk: 465

Copying... /usr/usr/testg.x
Copying... /usr/usr/testh.x
Copying... /usr/usr/testi.x
File: testi.x      size: 15           requires: 1 blocks
Last block's ID: 484    Blocks assigned!
Address in file table: 15          Address in disk: 484

Copying... /usr/usr/testj.x
Copying... /usr/usr/testk.x
File: testk.x     size: 149         requires: 18 blocks
Last block's ID: 502    Blocks assigned!
Address in file table: 16          Address in disk: 485

Copying... /usr/usr/testl.x
FLAGGLAL
File: test1.y, is in the system
File: test1.y, size: 64

Copying to... /usr/usr/test1.y

Deleting... /usr/usr/second.y
File: second.y, is in the system

                    ------ DISK MAPPER ------
*S *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 -13 -14 -15 -16 -17 -18 -19 -0
-6 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -0
-8 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -0
-10 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
```

```
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 -3 -4 -5 -6 -7 -8 -9 -10 -11

                ------ FILE TABLE ------
fl[1]: 413 B    address: 2      file: first.y
fl[2]: NULL
fl[3]: 64 B     address: 114    file: test1.y
fl[4]: 326 B    address: 122    file: test2.y
fl[5]: 301 B    address: 162    file: test3.y
fl[6]: 398 B    address: 199    file: test4.y
fl[7]: 421 B    address: 248    file: test5.y
fl[8]: 311 B    address: 300    file: test6.y
fl[9]: 159 B    address: 338    file: testa.x
fl[10]: 126 B   address: 357    file: testb.x
fl[11]: 137 B   address: 372    file: testc.x
fl[12]: 169 B   address: 389    file: testd.x
fl[13]: 445 B   address: 410    file: teste.x
fl[14]: 157 B   address: 465    file: testf.x
fl[15]: 15 B    address: 484    file: testi.x
fl[16]: 149 B   address: 485    file: testk.x

                ------ VIRTUAL DISK ------
-Size: 4096 Bytes
-Number of blocks: 511
-Available blocks: 71
-Available space: 568 Bytes
-Files: 15


Deleting... /usr/usr/testb.x
File: testb.x, is in the system


                ------ DISK MAPPER ------
*S *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 -13 -14 -15 -16 -17 -18 -19 -0
-6 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -0
-8 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -0
-10 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
```

```
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 -17 -18 -19 -0
-36 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 -3 -4 -5 -6 -7 -8 -9 -10 -11

                  ------ FILE TABLE ------
fl[1]: 413 B      address: 2      file: first.y
fl[2]: NULL
fl[3]: 64 B       address: 114    file: test1.y
fl[4]: 326 B      address: 122    file: test2.y
fl[5]: 301 B      address: 162    file: test3.y
fl[6]: 398 B      address: 199    file: test4.y
fl[7]: 421 B      address: 248    file: test5.y
fl[8]: 311 B      address: 300    file: test6.y
fl[9]: 159 B      address: 338    file: testa.x
fl[10]: NULL
fl[11]: 137 B     address: 372    file: testc.x
fl[12]: 169 B     address: 389    file: testd.x
fl[13]: 445 B     address: 410    file: teste.x
fl[14]: 157 B     address: 465    file: testf.x
fl[15]: 15 B      address: 484    file: testi.x
fl[16]: 149 B     address: 485    file: testk.x

                  ------ VIRTUAL DISK ------
-Size: 4096 Bytes
-Number of blocks: 511
-Available blocks: 86
-Available space: 688 Bytes
-Files: 14


==============================================================
         ------ VIRTUAL DISK ------
-Size: 4096 Bytes
-Number of blocks: 511
-Available blocks: 511
```

```
-Available space: 4088 Bytes
-Files: 0

Creating... first.y
File: first.y      size: 462        requires: 57 blocks
Last block's ID: 58      Blocks assigned!
Address in file table: 1            Address in disk: 2

Creating... second.y
File: second.y size: 298        requires: 37 blocks
Last block's ID: 95      Blocks assigned!
Address in file table: 2            Address in disk: 59

Creating... test1.y
File: test1.y      size: 332        requires: 41 blocks
Last block's ID: 136    Blocks assigned!
Address in file table: 3            Address in disk: 96

Creating... test2.y
File: test2.y      size: 484        requires: 60 blocks
Last block's ID: 196    Blocks assigned!
Address in file table: 4            Address in disk: 137

Creating... test3.y
File: test3.y      size: 27          requires: 3 blocks
Last block's ID: 199    Blocks assigned!
Address in file table: 5            Address in disk: 197

Creating... test4.y
File: test4.y      size: 143        requires: 17 blocks
Last block's ID: 216    Blocks assigned!
Address in file table: 6            Address in disk: 200

Creating... test5.y
File: test5.y      size: 313        requires: 39 blocks
Last block's ID: 255    Blocks assigned!
Address in file table: 7            Address in disk: 217

Creating... test6.y
File: test6.y      size: 200        requires: 25 blocks
Last block's ID: 280    Blocks assigned!
Address in file table: 8            Address in disk: 256

Creating... test8.y
File: test8.y      size: 14          requires: 1 blocks
Last block's ID: 281    Blocks assigned!
Address in file table: 9            Address in disk: 281

Creating... test9.y
File: test9.y      size: 262        requires: 32 blocks
```

```
Last block's ID: 313    Blocks assigned!
Address in file table: 10          Address in disk: 282

Creating... test10.y
File: test10.y   size: 195         requires: 24 blocks
Last block's ID: 337    Blocks assigned!
Address in file table: 11          Address in disk: 314

Creating... test11.y
File: test11.y   size: 422         requires: 52 blocks
Last block's ID: 389    Blocks assigned!
Address in file table: 12          Address in disk: 338

Creating... test12.y
File: test12.y   size: 114         requires: 14 blocks
Last block's ID: 403    Blocks assigned!
Address in file table: 13          Address in disk: 390

Creating... test13.y
File: test13.y   size: 395         requires: 49 blocks
Last block's ID: 452    Blocks assigned!
Address in file table: 14          Address in disk: 404

Creating... test14.y
File: test14.y   size: 394         requires: 49 blocks
Last block's ID: 501    Blocks assigned!
Address in file table: 15          Address in disk: 453

Creating... test15.y
Not enough free space in disk
: Success
Creating... test16.y
Not enough free space in disk
: Success
Creating... test17.y
Not enough free space in disk
: Success

Deleting... /usr/usr/second.y
File: second.y, is in the system

           ------ DISK MAPPER ------
*S *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 -19 -0
-6 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -0
-8 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
```

```
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15 *16 *17 *18 *19 *0
*1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11

                ------ FILE TABLE ------
fl[1]: 462 B    address: 2      file: first.y
fl[2]: NULL
fl[3]: 332 B    address: 96     file: test1.y
fl[4]: 484 B    address: 137    file: test2.y
fl[5]: 27 B     address: 197    file: test3.y
fl[6]: 143 B    address: 200    file: test4.y
fl[7]: 313 B    address: 217    file: test5.y
fl[8]: 200 B    address: 256    file: test6.y
fl[9]: 14 B     address: 281    file: test8.y
fl[10]: 262 B   address: 282    file: test9.y
fl[11]: 195 B   address: 314    file: test10.y
fl[12]: 422 B   address: 338    file: test11.y
fl[13]: 114 B   address: 390    file: test12.y
fl[14]: 395 B   address: 404    file: test13.y
fl[15]: 394 B   address: 453    file: test14.y

                ------ VIRTUAL DISK ------
-Size: 4096 Bytes
-Number of blocks: 511
-Available blocks: 48
-Available space: 384 Bytes
-Files: 14
```