Trabalho Prático 1 - Parte 1
Implementação de um Sistema de Arquivos

Guilherme Philippi
Campus Blumenau
Universidade Federal de Santa Catarina
UFSC
guilherme.philippi@grad.ufsc.br

2 de julho de 2019

1 Introdução

A implementação que se segue tem como objetivo implementar um emulador de um sistema de arquivos FAT simplificado, que utiliza alocação de blocos encadeados de 512 bytes. Segue as especificações do software:

- O sistema de arquivos utiliza método de alocação de blocos encadeado (conforme explicado em aula).
- O sistema de arquivos deve considerar setores de 512 bytes.
- A tabela de diretório deve armazenar uma lista de nomes de arquivos, tamanho em bytes (0 em se tratando de diretório), ponteiro para o bloco inicial do arquivo ou ponteiro para diretório.
- Os setores são numerados de 0 a n.

  O sistema de arquivos utiliza 4 bytes (32 bits) para numeração dos blocos. Assim, são possíveis 2 32 blocos de 512 bytes cada, totalizando 2 terabytes de espaço total suportado pelo sistema de arquivos.
- O sistema de arquivos utiliza mapeamento de blocos livres por encadeamento.
- O setor 0 contem o ponteiro para a lista de blocos livres.
- O diretório raíz ocupa o setor 1.

As estruturas que definem o formato dos dados a serem utilizados são definidos pelo cabeçalho filesystem.h, conforme é mostrado abaixo.

```
1 #define SECTOR_SIZE             512
2 #define NUMBER_OF_SECTORS       2048
3 #define FILENAME                "simul.fs"
4
5
```

```
6  /* Filesystem structures. */
7  /**
8   * Sector 0.
9   */
10 struct sector_0{
11         unsigned int free_sectors_list;
12         unsigned char unused[508];
13 };
14
15 /**
16  * File or directory entry.
17  */
18 struct file_dir_entry{
19         unsigned int dir;                /**< File or directory representation. Use 1
20         char name[20];                   /**< File or directorty name. */
21         unsigned int size_bytes;         /**< Size of the file in bytes. Use 0 for dir
22         unsigned int sector_start;       /**< Initial sector of the file ou sector of
23 };
24
25 /**
26  * Directory table.
27  */
28 struct table_directory{
29         struct file_dir_entry entries[16];       /**< List of file or directories. */
30 };
31
32 /**
33  * Sector data.
34  */
35 struct sector_data{
36         unsigned char data[508];         /**< File data. */
37         unsigned int next_sector;        /**< Next sector. Use 0 if it is the last sec
38 };
39
40
41 int fs_format();
42 int fs_create(char* input_file, char* simul_file);
43 int fs_read(char* output_file, char* simul_file);
44 int fs_del(char* simul_file);
45 int fs_ls(char *dir_path);
46 int fs_mkdir(char* directory_path);
47 int fs_rmdir(char *directory_path);
48 int fs_free_map(char *log_f);
```

Tendo estas estruturas definidas, é importante mencionar alguns pontos sobre a implementação:

- Quando uma entrada de diretório (directory_entry) possui nome, size e index iguais a 0, significa fim da lista de arquivos.
- Quando a estrutura directory_entry apresentar apenas start como 0, significa que aquele arquivo ou diretório foi excluído e seu nome não deve ser apresentando para o usuário, a entrada fica disponível para um novo arquivo.
- Quando o membro next_sector de sector_data for 0 significa fim da lista de blocos para o arquivo em questão.

- Como o sistema de arquivos suporta um disco de tamanho considerável (2TB) é necessário configurar um tamanho máximo para o arquivo de simulação. Use o comando -format (veja abaixo) para limitar o tamanho do arquivo de simulação.

A aplicação suporta as seguintes operações sobre o sistema de arquivos:

- Inicializar

  exemplo: simulfs -format <tamanho em megabytes>

- Criar (-create <arquivo original> <destino no sistema virtual>)

  exemplo: simulfs -create /home/user/classe.xls /<caminho>/alunos.xls

- Ler (-read <arquivo no disco> <caminho no sistema virtual>)

  exemplo: simulfs -read /home/user/classe.xls <caminho>/alunos.xls

- Apagar

  exemplo: simulfs -del <caminho>/aluno.xls

- Listar arquivos ou diretórios.

  exemplo: simulfs -ls <caminho>

  f paisagem.jpg 2048 bytes

  d viagem

  f lobo.jpg 5128 bytes

- Criar diretório

  exemplo: simulfs -mkdir <caminho>/aulas

- Apagar diretório - Somente apaga se o diretório estiver vazio.

  exemplo: simulfs -rmdir <caminho>/aulas

Nota: <caminho> refere-se ao diretório onde a ação será realizada. Cada nome de diretório é separado por "/". Exemplo: /home/user/aulas

Nota[2]: O diretório raiz é indicado apenas por "/". Por exemplo, listar os arquivos e diretórios do diretório raiz: simulfs -ls /

2 Implementação

Nesta seção apresenta-se a implementação.

2.1 $\mathbf{fs}_simul.c$

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "filesystem.h"
5
6 void usage(char *exec){
```

```
7           printf("%s -format\n", exec);
8           printf("%s -create <disk file> <simulated file>\n", exec);
9           printf("%s -read <disk file> <simulated file>\n", exec);
10      printf("%s -ls <absolute directory path>\n", exec);
11          printf("%s -del <simulated file>\n", exec);
12          printf("%s -mkdir <absolute directory path>\n", exec);
13          printf("%s -rmdir <absolute directory path>\n", exec);
14  }
15
16
17  int main(int argc, char **argv){
18
19          if(argc<2){
20                  usage(argv[0]);
21          } else if( !strcmp(argv[1], "-format")){
22                  fs_format();
23          } else if(argc<3){
24                  usage(argv[0]);
25          } else if( !strcmp(argv[1], "-del")){
26                  fs_del(argv[2]);
27          } else if( !strcmp(argv[1], "-ls")){
28                  fs_ls(argv[2]);
29          } else if( !strcmp(argv[1], "-mkdir")){
30                  fs_mkdir(argv[2]);
31          } else if( !strcmp(argv[1], "-rmdir")){
32                  fs_rmdir(argv[2]);
33          } else if (argc<4){
34                  usage(argv[0]);
35          } else if( !strcmp(argv[1], "-read")){
36                  fs_read(argv[2], argv[3]);
37          } else if( !strcmp(argv[1], "-create")){
38                  fs_create(argv[2], argv[3]);
39          } else{
40                  usage(argv[0]);
41          }
42
43          /* Create a map of used/free disk sectors. */
44          fs_free_map("log.dat");
45
46          return  0;
47  }
```

2.2 libdisksimul.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <sys/stat.h>
6  #include <sys/types.h>
7  #include "libdisksimul.h"
8
9  /* Simple library to simul read/write access to disk sectors. */
10
11 static FILE* simulfile = NULL;
12
13 /**
14  * @brief Disk Simulator Init.
```

```
15  *
16  * Create or open (if it already exist) the simulation file.
17  *
18  * @param filename Name of the input/output file.
19  * @param sector_size Sector size in number of bytes.
20  * @param number_sector Total number of sectors.
21  * @param format Force create new file.
22  * @return Return 0 on success, otherwise error.
23  */
24 int ds_init(char* filename, int sector_size, int number_sectors, int format){
25         struct stat b;
26
27         if(format == 0){
28                 /* Check if the file already exists */
29                 if( stat(filename, &b) == 0){
30                         /* File exists, open for read/write. */
31                         if( (simulfile = fopen(filename, "r+b")) == NULL){
32                                 /* error openning the file */
33                                 perror("fopen: ");
34                                 return 1;
35                         }
36                         return 0;
37                 }
38                 return 1;
39         }
40
41         /* File doesn't exist initialize it. */
42
43         /* Create file  */
44         if( (simulfile = fopen(filename, "w")) == NULL){
45                 /* error openning the file */
46                 perror("fopen: ");
47                 return 1;
48         }
49
50         /* Set file size */
51         ftruncate(fileno(simulfile), (sector_size*number_sectors));
52
53         fclose(simulfile);
54
55         /* Reopen the file for input/output */
56         if( (simulfile = fopen(filename, "r+b")) == NULL){
57                 /* error openning the file */
58                 perror("fopen: ");
59                 return 1;
60         }
61
62         return 0;
63 }
64
65 /**
66  * Disk Simulator Read Sector.
67  *
68  * Read a sector and load the data to the memory in data.
69  *
70  * @param sector_number Number of the sector.
71  * @param data Pointer to buffer to store the data.
72  * @param sector_size Sector size in bytes.
```

```
 * @return 0 if success, otherwise error.
 */
int ds_read_sector(int sector_number, void *data, int sector_size){
        int ret;
        /* locate the sector .*/
        if ( (ret = fseek(simulfile, sector_number*sector_size, SEEK_SET)) != 0){
                return ret;
        }

        /* read the sector the memory buffer pointed by data. */
        if ( (ret = fread(data, sizeof(char), sector_size, simulfile)) == 0){
                return ret;
        }

        return 0;
}

/**
 * Disk Simulator Write Sector.
 *
 * Write a sector from data in memory.
 *
 * @param sector_number Number of the sector.
 * @param data Pointer to buffer to store the data.
 * @param sector_size Sector size in bytes.
 * @return 0 if success, otherwise error.
 */
int ds_write_sector(int sector_number, void *data, int sector_size){
        int ret;
        /* locate the sector .*/
        if ( (ret = fseek(simulfile, sector_number*sector_size, SEEK_SET)) != 0){
                return ret;
        }

        /* load the sector to the memory buffer pointed by data. */
        if ( (ret = fwrite(data, sizeof(char), sector_size, simulfile)) == 0){
                return ret;
        }

        return 0;
}

/**
 * Disk Simulator Stop.
 *
 * Stop disk simulation.
 *
 * @param fp File pointer to the I/O file.
 */
void ds_stop(){
        fclose(simulfile);
        simulfile = NULL;
}
```

2.3 filesystem.c

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <sys/wait.h>
#include "libdisksimul.h"
#include "filesystem.h"
#define DATA_LENGTH 508
#define DIR_LENGTH 16

/**
 * @brief Format disk.
 *
 */
int fs_format(){
        int ret, i;
        struct table_directory root_dir;
        struct sector_0 sector0;
        struct sector_data sector;

        if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 1))){
                return ret;
        }

        memset(&sector0, 0, sizeof(struct sector_0));

        /* first free sector. */
        sector0.free_sectors_list = 2;

        ds_write_sector(0, (void*)&sector0, SECTOR_SIZE);

        memset(&root_dir, 0, sizeof(root_dir));

        ds_write_sector(1, (void*)&root_dir, SECTOR_SIZE);

        /* Create a list of free sectors. */
        memset(&sector, 0, sizeof(sector));

        for(i=2;i<NUMBER_OF_SECTORS;i++){
                if(i<NUMBER_OF_SECTORS-1){
                        sector.next_sector = i+1;
                }else{
                        sector.next_sector = 0;
                }
                ds_write_sector(i, (void*)&sector, SECTOR_SIZE);
        }

        ds_stop();

        printf("Disk size %d kbytes, %d sectors.\n", (SECTOR_SIZE*NUMBER_OF_SECTORS)/1

        return 0;
}

/**
 * @brief Localize a directory.
 * @param dir Directory struct reference.
 * @param absoluteDir Destination file path on the simulated file system.
 * @param file File name residue in absoluteDir
 * @param section Reference to section locate of directory localized
 * @param showLabel Set 1 to print directory name or 0 to not print
```

```
61  * @return 0 on success.
62  */
63 int locateDir(struct table_directory* dir, char *absoluteDir, char **file, unsigned in
64         int i, newDir = 0, j;
65         char *dirBuff = (char*)calloc(20, 1);
66         for (i = 0; i < strlen(absoluteDir); i++)
67         {
68                 if (absoluteDir[i] == '/'){
69                         if (i+1 >= strlen(absoluteDir)) return 1;
70                         newDir = 1;
71                         break;
72                 }else{
73                         if(i>20) return 1;
74                         dirBuff[i] = absoluteDir[i];
75                 }
76         }
77         if (strlen(dirBuff) < 1) return 1;
78         if (newDir){
79                 for (j = 0; j < DIR_LENGTH; j++){
80                         if (!dir->entries[j].sector_start){
81                                 continue;
82                         }
83                         if (!strcmp(dirBuff, dir->entries[j].name)){
84                                 if (dir->entries[j].dir){
85                                         if (section) *section = dir->entries[j].sector
86                                         int ret;
87                                         if ((ret = ds_read_sector(dir->entries[j].sect
88                                                 ds_stop();
89                                                 return ret;
90                                         }
91                                         free(dirBuff);
92                                         return locateDir(dir, absoluteDir+i+1, file, s
93                                 } else return 1;
94                         }
95                 }
96         } else {
97                 if (file) *file = dirBuff;
98                 return 0;
99         }
100        return 1;
101
102 }
103
104 /**
105  * @brief Create a new file on the simulated filesystem.
106  * @param input_file Source file path.
107  * @param simul_file Destination file path on the simulated file system.
108  * @return 0 on success.
109  */
110 int fs_create(char* input_file, char* simul_file){
111         int ret;
112         struct stat b;
113
114         if (strlen(simul_file)<2 || simul_file[0] != '/'){
115                 /* Param error */
116                 perror("simul_file is not valid: ");
117                 return 1;
118         }
```

```
119
120         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
121                 return ret;
122         }
123         FILE* file = NULL;
124         /* Check if the input_file exists and open it */
125         if( stat(input_file, &b) == 0){
126                 if( (file = fopen(input_file, "rb")) == NULL){
127                         /* error openning the file */
128                         perror("fopen: ");
129                         ds_stop();
130                         return 1;
131                 }
132         }else{
133                 /* error file not exist */
134                 perror("fileNotExist: ");
135                 ds_stop();
136                 return 1;
137         }
138
139         // check the existence of space for new data
140         struct sector_0 sector0;
141     memset(&sector0, 0, sizeof(sector0));
142         if ( (ret = ds_read_sector(0, (void*)&sector0, SECTOR_SIZE))){
143                 fclose(file);
144                 ds_stop();
145                 return ret;
146         }
147         if (sector0.free_sectors_list == 0){
148                 /* error end of memory */
149                 perror("EndOfMemory: ");
150                 fclose(file);
151                 ds_stop();
152                 return 1;
153         }
154         struct sector_data freeSector;
155     memset(&freeSector, 0, sizeof(freeSector));
156         if ( (ret = ds_read_sector(sector0.free_sectors_list, (void*)&freeSector, SECT
157                 fclose(file);
158                 ds_stop();
159                 return ret;
160         }
161
162         // check the directory existence and create the structure for save
163         struct table_directory dir;
164     memset(&dir, 0, sizeof(dir));
165         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
166                 fclose(file);
167                 ds_stop();
168                 return ret;
169         }
170         char *filename;
171         unsigned int sec = 1;
172         if ( (ret = locateDir(&dir, simul_file+1, &filename, &sec, 0))){
173                 perror("Not is possible localize directory: ");
174                 fclose(file);
175                 ds_stop();
176                 return ret;
```

```
177         }
178         if (strlen(filename) < 1){
179                 perror("Invalid file name: ");
180                 fclose(file);
181                 ds_stop();
182                 return 1;
183         }
184         int i;
185         char hasSpace = 0;
186         for (i = 0; i < DIR_LENGTH; i++){
187                 if (!dir.entries[i].sector_start){
188                         dir.entries[i].dir = 0;
189                         strcpy(dir.entries[i].name, filename);
190                         dir.entries[i].size_bytes = b.st_size;
191                         dir.entries[i].sector_start = sector0.free_sectors_list;
192                         hasSpace = 1;
193                         break;
194                 }
195         }
196         if (!hasSpace){
197                 perror("Directory is loted: ");
198                 fclose(file);
199                 ds_stop();
200                 return 1;
201         }
202
203         // save the new file
204         int n;
205         unsigned int last = 0;
206     while((n = fread(freeSector.data, sizeof(char), DATA_LENGTH, file)) == DATA_LENGTH
207         if ( (ret = ds_write_sector(sector0.free_sectors_list, (void*)&freeSector, SEC
208                         fclose(file);
209                         ds_stop();
210                         return ret;
211                 }
212                 last = sector0.free_sectors_list;
213                 sector0.free_sectors_list = freeSector.next_sector;
214                 if (sector0.free_sectors_list == 0){
215                         /* error end of memory */
216                         perror("EndOfMemory: ");
217                         fclose(file);
218                         ds_stop();
219                         return 1;
220                 }
221         memset(&freeSector, 0, sizeof(freeSector));
222                 if ( (ret = ds_read_sector(sector0.free_sectors_list, (void*)&freeSect
223                         fclose(file);
224                         ds_stop();
225                         return ret;
226                 }
227     }
228         unsigned int aux;
229         if(n != 0){ // until exists data for saving
230                 aux = freeSector.next_sector;
231                 freeSector.next_sector = 0;
232                 if ( (ret = ds_write_sector(sector0.free_sectors_list, (void*)&freeSec
233                         fclose(file);
234                         ds_stop();
```

```
235                         return ret;
236                 }
237         }else if(last != 0){ // set the final of file in last section saved
238                 memset(&freeSector, 0, sizeof(freeSector));
239                 if ( (ret = ds_read_sector(last, (void*)&freeSector, SECTOR_SIZE))){
240                         fclose(file);
241                         ds_stop();
242                         return ret;
243                 }
244                 aux = freeSector.next_sector;
245                 freeSector.next_sector = 0;
246                 if ( (ret = ds_write_sector(last, (void*)&freeSector, SECTOR_SIZE))){
247                         fclose(file);
248                         ds_stop();
249                         return ret;
250                 }
251         }else {
252                 /* error input_file void */
253                 perror("Input_file is Void: ");
254                 fclose(file);
255                 ds_stop();
256                 return 1;
257         }
258         sector0.free_sectors_list = aux;
259         if ( (ret = ds_write_sector(0, (void*)&sector0, SECTOR_SIZE))){ // write the
260                 fclose(file);
261                 ds_stop();
262                 return ret;
263         }
264         if ( (ret = ds_write_sector(sec, (void*)&dir, SECTOR_SIZE))){ // write the di
265                 fclose(file);
266                 ds_stop();
267                 return ret;
268         }
269     fclose(file);
270
271         ds_stop();
272
273         return 0;
274 }
275
276 /**
277  * @brief Read file from the simulated filesystem.
278  * @param output_file Output file path.
279  * @param simul_file Source file path from the simulated file system.
280  * @return 0 on success.
281  */
282 int fs_read(char* output_file, char* simul_file){
283         int ret;
284
285         if (strlen(simul_file)<2 || simul_file[0] != '/'){
286                 /* Param error */
287                 perror("simul_file is not valid: ");
288                 return 1;
289         }
290
291         if (strlen(output_file)<1 || output_file[strlen(output_file)-1] == '/'){
292                 /* Param error */
```

```
293                 perror("output_file is not valid: ");
294                 return 1;
295         }
296
297         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
298                 return ret;
299         }
300
301         FILE* file = NULL;
302         /* Create the file for output */
303         if( (file = fopen(output_file, "wb")) == NULL){
304                 /* error creating the file */
305                 perror("fopen: ");
306                 ds_stop();
307                 return 1;
308         }
309
310         // check the file existence in simul
311         struct table_directory dir;
312     memset(&dir, 0, sizeof(dir));
313         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
314                 fclose(file);
315                 ds_stop();
316                 return ret;
317         }
318         char *filename;
319         unsigned int sec = 1;
320         if ( (ret = locateDir(&dir, simul_file+1, &filename, &sec, 0))){
321                 perror("Not is possible localize directory: ");
322                 fclose(file);
323                 ds_stop();
324                 return ret;
325         }
326         if (strlen(filename) < 1){
327                 perror("Invalid input_file name: ");
328                 fclose(file);
329                 ds_stop();
330                 return 1;
331         }
332
333         int i;
334         char flag = 0;
335         for (i = 0; i < DIR_LENGTH; i++){
336                 if (dir.entries[i].sector_start && !dir.entries[i].dir && !strcmp(dir.
337                         flag = 1;
338                         break;
339                 }
340         }
341         if (!flag) {
342                 perror("Not is possible find the input_file: ");
343                 fclose(file);
344                 ds_stop();
345                 return 1;
346         }
347         int lastPiece = dir.entries[i].size_bytes%DATA_LENGTH;
348
349         // read input file and write in output file
350         struct sector_data data;
```

```
351         memset(&data, 0, sizeof(data));
352         if ( (ret = ds_read_sector(dir.entries[i].sector_start, (void*)&data, SECTOR_
353                 perror("error when read the file: ");
354                 fclose(file);
355                 ds_stop();
356                 return ret;
357         }
358
359         while (data.next_sector){
360                 if ( !(ret = fwrite(data.data, sizeof(char), DATA_LENGTH, file))){
361                         perror("error when write in the outpu_file: ");
362                         fclose(file);
363                         ds_stop();
364                         return ret;
365                 }
366                 if ( (ret = ds_read_sector(data.next_sector, (void*)&data, SECTOR_SIZE
367                         perror("error when read the file: ");
368                         fclose(file);
369                         ds_stop();
370                         return ret;
371                 }
372         }
373         if (lastPiece){
374                 if ( !(ret = fwrite(data.data, sizeof(char), lastPiece, file))){
375                         perror("error when write last piece in the outpu_file: ");
376                         fclose(file);
377                         ds_stop();
378                         return ret;
379                 }
380         }
381
382         fclose(file);
383         ds_stop();
384
385         return 0;
386 }
387
388 /**
389  * @brief Delete file from file system.
390  * @param simul_file Source file path.
391  * @return 0 on success.
392  */
393 int fs_del(char* simul_file){
394         int ret;
395
396         if (strlen(simul_file)<2 || simul_file[0] != '/'){
397                 /* Param error */
398                 perror("simul_file is not valid: ");
399                 return 1;
400         }
401
402         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
403                 return ret;
404         }
405
406         // check the file existence in simul
407         struct table_directory dir;
408     memset(&dir, 0, sizeof(dir));
```

```
409         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
410                 ds_stop();
411                 return ret;
412         }
413         char *filename;
414         unsigned int sec = 1;
415         if ( (ret = locateDir(&dir, simul_file+1, &filename, &sec, 0))){
416                 perror("Not is possible localize directory: ");
417                 ds_stop();
418                 return ret;
419         }
420         if (strlen(filename) < 1){
421                 perror("Invalid input_file name: ");
422                 ds_stop();
423                 return 1;
424         }
425
426         int i;
427         char flag = 0;
428         for (i = 0; i < DIR_LENGTH; i++){
429                 if (dir.entries[i].sector_start && !dir.entries[i].dir && !strcmp(dir
430                         flag = 1;
431                         break;
432                 }
433         }
434         if (!flag) {
435                 perror("Not is possible find the input_file: ");
436                 ds_stop();
437                 return 1;
438         }
439
440         // Delete input file
441         struct sector_data data;
442         memset(&data, 0, sizeof(data));
443         if ( (ret = ds_read_sector(dir.entries[i].sector_start, (void*)&data, SECTOR_
444                 perror("error when read the file: ");
445                 ds_stop();
446                 return ret;
447         }
448
449         struct sector_0 sector0;
450     memset(&sector0, 0, sizeof(sector0));
451         if ( (ret = ds_read_sector(0, (void*)&sector0, SECTOR_SIZE))){
452                 ds_stop();
453                 return ret;
454         }
455         unsigned int last;
456         do{
457                 last = data.next_sector;
458                 if ( (ret = ds_read_sector(data.next_sector, (void*)&data, SECTOR_SIZ
459                         perror("error when read the file: ");
460                         ds_stop();
461                         return ret;
462                 }
463         } while (data.next_sector);
464         data.next_sector = sector0.free_sectors_list;
465         if ((ret = ds_write_sector(last, &data, SECTOR_SIZE))){
466                 perror("error when write: ");
```

```
467                 ds_stop();
468                 return ret;
469         }
470
471         sector0.free_sectors_list = dir.entries[i].sector_start;
472         if ((ret = ds_write_sector(0, &sector0, SECTOR_SIZE))){
473                 perror("error when write: ");
474                 ds_stop();
475                 return ret;
476         }
477
478         dir.entries[i].sector_start = 0;
479         if ((ret = ds_write_sector(sec, &dir, SECTOR_SIZE))){
480                 perror("error when write: ");
481                 ds_stop();
482                 return ret;
483         }
484
485         ds_stop();
486
487         return 0;
488 }
489
490 /**
491  * @brief List files from a directory.
492  * @param simul_file Source file path.
493  * @return 0 on success.
494  */
495 int fs_ls(char *dir_path){
496         int ret;
497
498         if (strlen(dir_path)<1 || dir_path[0] != '/'){
499                 /* Param error */
500                 perror("dir_path is not valid: ");
501                 return 1;
502         }
503
504         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
505                 return ret;
506         }
507
508         struct table_directory dir;
509     memset(&dir, 0, sizeof(dir));
510         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
511                 ds_stop();
512                 return ret;
513         }
514         char *filename;
515         if (strlen(dir_path) > 1)
516         if ( (ret = locateDir(&dir, dir_path+1, &filename, NULL, 0))){
517                 perror("Not is possible localize directory: ");
518                 ds_stop();
519                 return ret;
520         }
521
522         int i;
523         for (i = 0; i < DIR_LENGTH; i++){
524                 if (dir.entries[i].sector_start && !strcmp(dir.entries[i].name, filena
```

```
525                 {
526                         if (!dir.entries[i].dir){
527                                 perror("Not is possible localize directory: ");
528                                 ds_stop();
529                                 return ret;
530                         }
531                         if ( (ret = ds_read_sector(dir.entries[i].sector_start, (void*
532                                 ds_stop();
533                                 return ret;
534                         }
535                         break;
536                 }
537         }
538 
539         for (i = 0; i < DIR_LENGTH; i++){
540                 if (dir.entries[i].sector_start)
541                 {
542                         printf(" => %c %s\t%d bytes\n", (dir.entries[i].dir)?'d':'f',
543                 }
544         }
545 
546         ds_stop();
547 
548         return 0;
549 }
550 
551 /**
552  * @brief Create a new directory on the simulated filesystem.
553  * @param directory_path directory path.
554  * @return 0 on success.
555  */
556 int fs_mkdir(char* directory_path){
557         int ret;
558         if (strlen(directory_path)<2 || directory_path[0] != '/'){
559                 /* Param error */
560                 perror("directory_path is not valid: ");
561                 return 1;
562         }
563 
564         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
565                 return ret;
566         }
567 
568         // check the directory
569         struct table_directory dir;
570     memset(&dir, 0, sizeof(dir));
571         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
572                 ds_stop();
573                 return ret;
574         }
575         unsigned int sec = 1;
576         char *filename;
577         if ( (ret = locateDir(&dir, directory_path+1, &filename, &sec, 0))){
578                 perror("Not is possible localize directory: ");
579                 ds_stop();
580                 return ret;
581         }
582         int i;
```

```
583         for (i = 0; i < DIR_LENGTH; i++){
584                 if (dir.entries[i].sector_start && !strcmp(dir.entries[i].name, filena
585                 {
586                         perror("Not is possible create this directory, alredy exist a
587                         ds_stop();
588                         return ret;
589                 }
590         }
591         char flag = 0;
592         for (i = 0; i < DIR_LENGTH; i++){
593                 if (!dir.entries[i].sector_start)
594                 {
595                         flag = 1;
596                         break;
597                 }
598         }
599         if (!flag){
600                 perror("Directory is loted: ");
601                 ds_stop();
602                 return 1;
603         }
604
605         // check the existence of space for new data
606         struct sector_0 sector0;
607     memset(&sector0, 0, sizeof(sector0));
608         if ( (ret = ds_read_sector(0, (void*)&sector0, SECTOR_SIZE))){
609                 ds_stop();
610                 return ret;
611         }
612         if (sector0.free_sectors_list == 0){
613                 /* error end of memory */
614                 perror("EndOfMemory: ");
615                 ds_stop();
616                 return 1;
617         }
618         struct sector_data freeSector;
619     memset(&freeSector, 0, sizeof(freeSector));
620         if ( (ret = ds_read_sector(sector0.free_sectors_list, (void*)&freeSector, SECT
621                 ds_stop();
622                 return ret;
623         }
624
625         struct table_directory newDir;
626         memset(&newDir, 0, sizeof(newDir));
627         if ((ret = ds_write_sector(sector0.free_sectors_list, &newDir, SECTOR_SIZE))){
628                 perror("error when write: ");
629                 ds_stop();
630                 return ret;
631         }
632         dir.entries[i].sector_start = sector0.free_sectors_list;
633         dir.entries[i].dir = 1;
634         strcpy(dir.entries[i].name, filename);
635         dir.entries[i].size_bytes = 0;
636         if ((ret = ds_write_sector(sec, &dir, SECTOR_SIZE))){
637                 perror("error when write: ");
638                 ds_stop();
639                 return ret;
640         }
```

```
641
642         sector0.free_sectors_list = freeSector.next_sector;
643         if ((ret = ds_write_sector(0, &sector0, SECTOR_SIZE))){
644                 perror("error when write: ");
645                 ds_stop();
646                 return ret;
647         }
648
649         ds_stop();
650
651         return 0;
652 }
653
654 /**
655  * @brief Remove directory from the simulated filesystem.
656  * @param directory_path directory path.
657  * @return 0 on success.
658  */
659 int fs_rmdir(char *directory_path){
660         int ret;
661         if (strlen(directory_path)<3 || directory_path[0] != '/' || directory_path[str
662                 /* Param error */
663                 perror("directory_path is not valid: ");
664                 return 1;
665         }
666
667         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
668                 return ret;
669         }
670
671         // check the file existence in simul
672         struct table_directory dir;
673     memset(&dir, 0, sizeof(dir));
674         if ( (ret = ds_read_sector(1, (void*)&dir, SECTOR_SIZE))){
675                 ds_stop();
676                 return ret;
677         }
678         char *filename;
679         unsigned int sec = 1;
680         char *labelAux = (char*)calloc(strlen(directory_path), 1);
681         strcpy(labelAux, directory_path+1);
682
683         labelAux[strlen(labelAux)-1] = 0;
684         if ( (ret = locateDir(&dir, labelAux, &filename, &sec, 0))){
685                 perror("Not is possible localize directory: ");
686                 ds_stop();
687                 return ret;
688         }
689
690         if (strlen(filename) < 1){
691                 perror("Invalid directory_path name: ");
692                 ds_stop();
693                 return 1;
694         }
695         int i;
696         char flag = 0;
697         for (i = 0; i < DIR_LENGTH; i++){
698                 if (dir.entries[i].sector_start && dir.entries[i].dir && !strcmp(dir.
```

```
699                         flag = 1;
700                         break;
701                 }
702         }
703         if (!flag) {
704                 perror("Not is possible find the directory_path: ");
705                 ds_stop();
706                 return 1;
707         }
708
709         // Delete input directory
710         struct table_directory dirDel;
711         memset(&dirDel, 0, sizeof(dir));
712         if ( (ret = ds_read_sector(dir.entries[i].sector_start, (void*)&dirDel, SECTO
713                 perror("error when read the file: ");
714                 ds_stop();
715                 return ret;
716         }
717
718         int j;
719         for (j = 0; j < DIR_LENGTH; j++)
720         {
721                 if(dirDel.entries[j].sector_start)
722                 {
723                         perror("Directory not void: ");
724                         ds_stop();
725                         return ret;
726                 }
727         }
728
729         struct sector_0 sector0;
730     memset(&sector0, 0, sizeof(sector0));
731         if ( (ret = ds_read_sector(0, (void*)&sector0, SECTOR_SIZE))){
732                 ds_stop();
733                 return ret;
734         }
735
736         struct sector_data aux;
737         aux.next_sector = sector0.free_sectors_list;
738         if ((ret = ds_write_sector(dir.entries[i].sector_start, &aux, SECTOR_SIZE))){
739                 perror("error when write: ");
740                 ds_stop();
741                 return ret;
742         }
743         sector0.free_sectors_list = dir.entries[i].sector_start;
744         if ((ret = ds_write_sector(0, &sector0, SECTOR_SIZE))){
745                 perror("error when write: ");
746                 ds_stop();
747                 return ret;
748         }
749
750         dir.entries[i].sector_start = 0;
751         if ((ret = ds_write_sector(sec, &dir, SECTOR_SIZE))){
752                 perror("error when write: ");
753                 ds_stop();
754                 return ret;
755         }
756
```

```
757         ds_stop();
758
759         return 0;
760 }
761
762 /**
763  * @brief Generate a map of used/available sectors.
764  * @param log_f Log file with the sector map.
765  * @return 0 on success.
766  */
767 int fs_free_map(char *log_f){
768         int ret, i, next;
769         //struct root_table_directory root_dir;
770         struct sector_0 sector0;
771         struct sector_data sector;
772         char *sector_array;
773         FILE* log;
774         int pid, status;
775         int free_space = 0;
776         char* exec_params[] = {"gnuplot", "sector_map.gnuplot" , NULL};
777
778         if ( (ret = ds_init(FILENAME, SECTOR_SIZE, NUMBER_OF_SECTORS, 0))){
779                 return ret;
780         }
781
782         /* each byte represents a sector. */
783         sector_array = (char*)malloc(NUMBER_OF_SECTORS);
784
785         /* set 0 to all sectors. Zero means that the sector is used. */
786         memset(sector_array, 0, NUMBER_OF_SECTORS);
787
788         /* Read the sector 0 to get the free blocks list. */
789         ds_read_sector(0, (void*)&sector0, SECTOR_SIZE);
790
791         next = sector0.free_sectors_list;
792
793         while(next){
794                 /* The sector is in the free list, mark with 1. */
795                 sector_array[next] = 1;
796
797                 /* move to the next free sector. */
798                 ds_read_sector(next, (void*)&sector, SECTOR_SIZE);
799
800                 next = sector.next_sector;
801
802                 free_space += SECTOR_SIZE;
803         }
804
805         /* Create a log file. */
806         if( (log = fopen(log_f, "w")) == NULL){
807                 perror("fopen()");
808                 free(sector_array);
809                 ds_stop();
810                 return 1;
811         }
812
813         /* Write the the sector map to the log file. */
814         for(i=0;i<NUMBER_OF_SECTORS;i++){
```

```
815                 if(i%32==0) fprintf(log, "%s", "\n");
816                 fprintf(log, " %d", sector_array[i]);
817         }
818
819         fclose(log);
820
821         /* Execute gnuplot to generate the sector's free map. */
822         pid = fork();
823         if(pid==0){
824                 execvp("gnuplot", exec_params);
825         }
826         /* Wait gnuplot to finish */
827         wait(&status);
828
829         free(sector_array);
830
831         ds_stop();
832
833         printf("Free space %d kbytes.\n", free_space/1024);
834
835         return 0;
836 }
```